PHP Manual

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Zeev Suraski

Edited by

Stig Sæther Bakken

PHP Manual

by Stig Sæther Bakken, Alexander Aulbach, Egon Schmid, Jim Winstead, Lars Torben Wilson, Rasmus Lerdorf, and Zeev Suraski

Edited by Stig Sæther Bakken

Published 1999-11-03

Copyright

Dedication

For Christine and Rasmus

1999-11-03

Table of Contents

List of Tables

List of Examples

Preface

PHP is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly.

About this Manual

This manual is written in SGML using the DocBook DTD (http://www.ora.com/davenport/), using DSSSL (http://www.jclark.com/dsssl/) (Document Style and Semantics Specification Language) for formatting. The tools used for formatting HTML, TeX and RTF versions are Jade (http://www.jclark.com/jade/), written by James Clark (http://www.jclark.com/bio.htm) and The Modular DocBook Stylesheets (http://nwalsh.com/docbook/dsssl/) written by Norman Walsh (http://nwalsh.com/). PHP's documentation framework was assembled by Stig Sæther Bakken (mailto:stig@php.net).

I. Getting Started

Chapter 1. Introduction

What is PHP?

PHP is a server-side HTML-embedded scripting language.

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

Example 1-1. An introductory example

```
<html>
    <head>
        <title>Example</title>
    </head>
    <body>
        <?php echo "Hi, I’m a PHP script!"; ?>
    </body>
</html>
```

Notice how this is different from a CGI 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 a 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.

What can PHP do?

At the most basic level, PHP can do anything any other CGI program can do, such as collect form data, generate dynamic page content, or send and receive cookies.

Perhaps 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
dBase
Empress
FilePro
Informix
InterBase
mSQL
MySQL
Oracle
PostgreSQL
Solid
Sybase
Velocis
Unix dbm

PHP also has support for talking to other services using protocols such as IMAP, SNMP, NNTP, POP3, or even HTTP. You can also open raw network sockets and interact using other protocols.

A brief history of PHP

PHP was conceived sometime in the fall of 1994 by Rasmus Lerdorf (mailto:rasmus@lerdorf.on.ca). Early non-released versions were used on his home page to keep track of who was looking at his online resume. The first version used by others was available sometime in early 1995 and was known as the Personal Home Page Tools. It consisted of a very simplistic parser engine that only understood a few special macros and a number of utilities that were in common use on home pages back then. A guestbook, a counter and some other stuff. The parser was rewritten in mid-1995 and named PHP/FI Version 2. The FI came from another package Rasmus had written which interpreted html form data. He combined the Personal Home Page tools scripts with the Form Interpreter and added mSQL support and PHP/FI was born. PHP/FI grew at an amazing pace and people started contributing code to it.

It is difficult to give any hard statistics, but it is estimated that by late 1996 PHP/FI was in use on at least 15,000 web sites around the world. By mid-1997 this number had grown to over 50,000. Mid-1997 also saw a change in the development of PHP. It changed from being Rasmus' own pet project that a handful of people had contributed to, to being a much more organized team effort. The parser was rewritten from scratch by Zeev Suraski and Andi Gutmans and this new parser formed the basis for PHP Version 3. A lot of the utility code from PHP/FI was ported over to PHP3 and a lot of it was completely rewritten.

Today (mid-1999) either PHP/FI or PHP3 ships with a number of commercial products such as C2's StrongHold web server and RedHat Linux. A conservative estimate based on an extrapolation from numbers provided by NetCraft would be that PHP is in use on over 150,000 sites around the world. To put that in perspective, that is more sites than run Netscape's flagship Enterprise server on the Internet.

Also as of this writing, work is underway on the next generation of PHP, which will utilize the powerful Zend (http://www.zend.com) scripting engine to deliver higher performance, and will also support running under webservers other than Apache as a native server module.

Chapter 2. Installation

Downloading the latest version

The source code, and binary distributions for some platforms (including Windows), can be found at `http://www.php.net/`.

Installation on UNIX systems

This section will guide you through the configuration and installation of PHP. Prerequisite knowledge and software:

- Basic UNIX skills (being able to operate "make" and a C compiler)
- An ANSI C compiler
- A web server

Quick Installation Instructions (Apache Module Version)

```
1.  gunzip apache_1.3.x.tar.gz
2.  tar xvf apache_1.3.x.tar
3.  gunzip php-3.0.x.tar.gz
4.  tar xvf php-3.0.x.tar
5.  cd apache_1.3.x
6.  ./configure -prefix=/www
7.  cd ../php-3.0.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 -prefix=/www -activate-module=src/modules/php3/libphp3.a
13. make
14. make install

  Instead of this step you may prefer to simply copy the httpd binary
  overtop of your existing binary.  Make sure you shut down your
  server first though.

15. cd ../php-3.0.x
16. cp php3.ini-dist /usr/local/lib/php3.ini

  You can edit /usr/local/lib/php3.ini file to set PHP options.  If
  you prefer this file in another location, use
  -with-config-file-path=/path in step 8.

17. Edit your httpd.conf or srm.conf file and add:

          AddType application/x-httpd-php3 .php3

  You can choose any extension you wish here.  .php3 is simply the one
  we suggest.

18. 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.)
```

Configuration

There are two ways of configuring PHP.

- Using the "setup" script that comes with PHP. This script asks you a series of questions (almost like the "install" script of PHP/FI 2.0) and runs "configure" in the end. To run this script, type **./setup**.

  This script will also create a file called "do-conf", this file will contain the options passed to configure. You can edit this file to change just a few options without having to re-run setup. Then type **./do-conf** to run configure with the new options.

- Running configure by hand. To see what options you have, type **./configure –help**.

Details about some of the different configuration options are listed below.

Apache module

To build PHP as an Apache module, answer "yes" to "Build as an Apache module?" (the `-with-apache=DIR` option to configure) and specify the Apache distribution base directory. If you have unpacked your Apache distribution in `/usr/local/www/apache_1.2.4`, this is your Apache distribution base directory. The default directory is `/usr/local/etc/httpd`.

fhttpd module

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.

CGI version

The default is to build PHP as a CGI program. 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.

Database Support Options

PHP has native support for a number of databases (as well as ODBC):

Adabas D

```
-with-adabas=DIR
```

Compiles with Adabas D support. The parameter is the Adabas D install directory and defaults to `/usr/local/adabasd`.

Adabas home page (http://www.adabas.com/)

dBase

```
-with-dbase
```

Enables the bundled DBase support. No external libraries are required.

filePro

```
-with-filepro
```

Enables the bundled read-only filePro support. No external libraries are required.

mSQL

```
-with-msql=DIR
```

Enables mSQL support. The parameter to this option is the mSQL install directory and defaults to `/usr/local/Hughes`. This is the default directory of the mSQL 2.0 distribution. **configure** automatically detects which mSQL version you are running and PHP supports both 1.0 and 2.0, but if you compile PHP with mSQL 1.0, you can only access mSQL 1.0 databases, and vice-versa.

See also mSQL Configuration Directives in the configuration file.

mSQL home page (http://www.hughes.com.au)

MySQL

```
-with-mysql=DIR
```

Enables MySQL support. The parameter to this option is the MySQL install directory and defaults to `/usr/local`. This is the default installation directory of the MySQL distribution.

See also MySQL Configuration Directives in the configuration file.

MySQL home page (http://www.tcx.se)

iODBC

```
-with-iodbc=DIR
```

Includes iODBC support. This feature was first developed for iODBC Driver Manager, a freely redistributable ODBC driver manager which runs under many flavors of UNIX. The parameter to this option is the iODBC installation directory and defaults to `/usr/local`.

FreeODBC home page (http://users.ids.net/~bjepson/freeODBC/)

OpenLink ODBC

```
-with-openlink=DIR
```

Includes OpenLink ODBC support. The parameter to this option is the OpenLink ODBC installation directory and defaults to `/usr/local/openlink`.

OpenLink Software’s home page (http://www.openlinksw.com/)

Oracle

```
-with-oracle=DIR
```

Includes Oracle support. Has been tested and should be working at least with Oracle versions 7.0 through 7.3. The parameter is the ORACLE_HOME directory. You do not have to specify this parameter if your Oracle environment has been set up.

Oracle home page (http://www.oracle.com)

PostgreSQL

```
-with-pgsql=DIR
```

Includes PostgreSQL support. The parameter is the PostgreSQL base install directory and defaults to `/usr/local/pgsql`.

See also Postgres Configuration Directives in the configuration file.

PostgreSQL home page (http://www.postgreSQL.org/)

Solid

```
-with-solid=DIR
```

Includes Solid support. The parameter is the Solid install directory and defaults to `/usr/local/solid`.

Solid home page (http://www.solidtech.com)

Sybase

```
-with-sybase=DIR
```

Includes Sybase support. The parameter is the Sybase install directory and defaults to `/home/sybase`.

See also Sybase Configuration Directives in the configuration file.

Sybase home page (http://www.sybase.com)

Sybase-CT

```
-with-sybase-ct=DIR
```

Includes Sybase-CT support. The parameter is the Sybase-CT install directory and defaults to `/home/sybase`.

See also Sybase-CT Configuration Directives in the configuration file.

Velocis

```
-with-velocis=DIR
```

Includes Velocis support. The parameter is the Velocis install directory and defaults to `/usr/local/velocis`.

Velocis home page (http://www.raima.com)

A custom ODBC library

```
-with-custom-odbc=DIR
```

Includes support for an arbitrary custom ODBC library. The parameter is the base directory and defaults to `/usr/local`.

This option implies that you have defined CUSTOM_ODBC_LIBS when you run the configure script. You also must have a valid odbc.h header somewhere in your include path. If you don’t have one, create it and include your specific header from there. Your header may also require some extra definitions, particularly when it is multiplatform. Define them in CFLAGS.

For example, you can use Sybase SQL Anywhere on QNX as following:

```
CFLAGS=-DODBC_QNX
LDFLAGS=-lunix CUSTOM_ODBC_LIBS="-ldblib -lodbc" ./configure
-with-custom-odbc=/usr/lib/sqlany50
```

Unified ODBC

```
-disable-unified-odbc
```

Disables the Unified ODBC module, which is a common interface to all the databases with ODBC-based interfaces, such as Solid and Adabas D. It also works for normal ODBC libraries. Has been tested with iODBC, Solid, Adabas D and Sybase SQL Anywhere. Requires that one (and only one) of these modules or the Velocis module is enabled, or a custom ODBC library specified. This option is only applicable if one of the following options is used: –with-iodbc, –with-solid, –with-adabas, –with-velocis, or –with-custom-odbc,

See also Unified ODBC Configuration Directives in the configuration file.

LDAP

```
-with-ldap=DIR
```

Includes LDAP (Lightweight Directory Access Protocol) support. The parameter is the LDAP base install directory, defaults to `/usr/local/ldap`.

More information about LDAP can be found in RFC1777 (ftp://ftp.isi.edu/in-notes/rfc1777.txt) and RFC1778 (ftp://ftp.isi.edu/in-notes/rfc1778.txt).

Other configure options

–with-mcrypt=*DIR*

```
-with-mcrypt
```

Include support for the mcrypt library. See the mcrypt documentation for more information. If you use the optional *`DIR`* argument, PHP will look for mcrypt.h in *`DIR`*/include.

–enable-sysvsem

```
-enable-sysvsem
```

Include support for Sys V semaphores (supported by most Unix derivates). See the Semaphore and Shared Memory documentation for more information.

–enable-sysvshm

```
-enable-sysvshm
```

Include support for Sys V shared memory (supported by most Unix derivates). See the Semaphore and Shared Memory documentation for more information.

–with-xml

```
-with-xml
```

Include support for a non-validating XML parser using James Clark's expat library (http://www.jclark.com/xml/). See the XML function reference for details.

–enable-maintainer-mode

```
-enable-maintainer-mode
```

Turns on extra dependencies and compiler warnings used by some of the PHP developers.

–with-system-regex

```
-with-system-regex
```

Uses the system’s regular expression library rather than the bundled one. If you are building PHP as a server module, you must use the same library when building PHP as when linking the server. Enable this if the system’s library provides special features you need. It is recommended that you use the bundled library if possible.

–with-config-file-path

```
-with-config-file-path=DIR
```

The path used to look for the configuration file when PHP starts up.

–with-exec-dir

```
-with-exec-dir=DIR
```

Only allow running of executables in DIR when in safe mode. Defaults to `/usr/local/bin`. This option only sets the default, it may be changed with the safe_mode_exec_dir directive in the configuration file later.

–enable-debug

```
-enable-debug
```

Enables extra debug information. This makes it possible to gather more detailed information when there are problems with PHP. (Note that this doesn’t have anything to do with debugging facilities or information available to PHP scripts.)

–enable-safe-mode

```
-enable-safe-mode
```

Enables "safe mode" by default. This imposes several restrictions on what PHP can do, such as opening only files within the document root. Read the Security chapter for more more information. CGI users should always enable secure mode. This option only sets the default, it may be enabled or disabled with the safe_mode directive in the configuration file later.

–enable-track-vars

```
-enable-track-vars
```

Makes PHP keep track of where GET/POST/cookie variables come from in the arrays HTTP_GET_VARS, HTTP_POST_VARS and HTTP_COOKIE_VARS. This option only sets the default, it may be enabled or disabled with the track_vars directive in the configuration file later.

–enable-magic-quotes

```
-enable-magic-quotes
```

Enable magic quotes by default. This option only sets the default, it may be enabled or disabled with the magic_quotes_runtime directive in the configuration file later. See also the magic_quotes_gpc and the magic_quotes_sybase directives.

–enable-debugger

```
-enable-debugger
```

Enables the internal PHP debugger support. This feature is still in an experimental state. See also the Debugger Configuration directives in the configuration file.

–enable-discard-path

```
-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. Read the section in the security chapter about this option.

–enable-bcmath

```
-enable-bcmath
```

Enables **bc** style arbitrary precision math functions. See also the bcmath.scale option in the configuration file.

–enable-force-cgi-redirect

```
-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.

When using PHP as a CGI binary, PHP by default always first checks that it is used by redirection (for example under Apache, by using Action directives). This makes sure that the PHP binary cannot be used to bypass standard web server authentication procedures by calling it directly, like `http://my.host/cgi-bin/php/secret/doc.html`. This example accesses `http://my.host/secret/doc.html` but does not honour any security settings enforced by httpd for directory `/secret`.

Not enabling option disables the check and enables bypassing httpd security and authentication settings. Do this only if your server software is unable to indicate that a safe redirection was done and all your files under your document root and user directories may be accessed by anyone.

Read the section in the security chapter about this option.

–disable-short-tags

```
-disable-short-tags
```

Disables the short form `<? ?>` PHP tags. You must disable the short form if you want to use PHP with XML. With short tags disabled, the only PHP code tag is `<?php ?>`. This option only sets the default, it may be enabled or disabled with the short_open_tag directive in the configuration file later.

–enable-url-includes

```
-enable-url-includes
```

Makes it possible to run code on other HTTP or FTP servers directly from PHP with include(). See also the include_path option in the configuration file.

–disable-syntax-hl

```
-disable-syntax-hl
```

Turns off syntax highlighting.

CPPFLAGS and LDFLAGS

To make the PHP installation look for header or library files in different directories, modify the CPPFLAGS and LDFLAGS environment variables, respectively. If you are using a sensible shell, you should be able to do **LDFLAGS=-L/my/lib/dir CPPFLAGS=-I/my/include/dir ./configure**

Building

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

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 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.

Installation on Windows 95/98/NT systems

This install guide will help you install and configure PHP on your Windows 9x/NT webservers. This guide was compiled by Bob Silva (mailto:bob_silva@mail.umesd.k12.or.us). The latest revision can be found at http://www.umesd.k12.or.us/php/win32install.html.

This guide provides installation support for:

- Personal Web Server (Newest version recommended)
- Internet Information Server 3 or 4
- Apache 1.3.x
- Omni HTTPd 2.0b1

General Installation Steps

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:\PHP3\" is a good start.
- Copy the file, ’php3-dist.ini’ to your ’%WINDOWS%’ directory and rename it to ’php3.ini’. Your ’%WINDOWS%’ directory is typically:

  c:\windows for Windows 95/98
  c:\winnt or c:\winnt40 for NT servers

- Edit your ’php3.ini’ file:
  - You will need to change the ’extension_dir’ setting to point to your php-install-dir, or where you have placed your ’php3_*.dll’ files. ex: c:\php3

- If you are using Omni Httpd, do not follow the next step. Set the ’doc_root’ to point to your webservers document_root. ex: c:\apache\htdocs or c:\webroot
- Choose which modules you would like to load when PHP starts. You can uncomment the: ’extension=php3_*.dll’ lines to load these modules. Some modules require you to have additional libraries installed on your system for the module to work correctly. The PHP FAQ (http://www.php.net/FAQ.php3) has more information on where to get supporting libraries. You can also load a module dynamically in your script using: **dl("php_*.dll");**
- On PWS and IIS, you can set the browscap.ini to point to: ’c:\windows\system\inetsrv\browscap.ini’ on Windows 95/98 and ’c:\winnt\system32\inetsrv\browscap.ini’ on NT Server. Additional information on using the browscap functionality in PHP can be found at this mirror (http://www.netvision.net.il/browser-id.php3), select the "source" button to see it in action.

The DLLs for PHP extensions are prefixed with ’php3_’. This prevents confusion between PHP extensions and their supporting libraries.

Windows 95/98/NT and PWS/IIS 3

The recommended method for configuring these servers is to use the INF file included with the distribution (php_iis_reg.inf). 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. ex: `.php3`
- Double click on the new string value and enter the path to `php.exe` in the value data field. ex: `c:\php3\php.exe %s %s`. The ’%s %s’ is VERY important, PHP will not work properly without it.
- Repeat these steps for each extension you wish to associate with PHP scripts.
- Now 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: `.php3`
- 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:\php3\php.exe -q %1`. (don't forget the `%1`).
- Exit Regedit.

PWS and IIS 3 users now have a fully operational system. IIS 3 users can use a nifty tool (http://www.genusa.com/iis/iiscfg.html) from Steven Genusa to configure their script maps.

Windows NT and IIS 4

To install PHP on an NT Server running IIS 4, follow these instructions:

- In Internet Service Manager (MMC), select the Web site or the starting point directory of an application.
- Open the directory's property sheets (by right clicking and selecting properties), and then click the Home Directory, Virtual Directory, or Directory tab.
- Click the Configuration button, and then click the App Mappings tab.
- Click Add, and in the Executable box, type: `c:\path-to-php-dir\php.exe %s %s`. You MUST have the %s %s on the end, PHP will not function properly if you fail to do this.
- In the Extension box, type the file name extension you want associated with PHP scripts. (You must repeat step 5 and 6 for each extension you want accociated with PHP scripts. (`.php3` and `.phtml` are common)
- 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`.

Windows 9x/NT and Apache 1.3.x

You must edit your `srm.conf` or `httpd.conf` to configure Apache to work with the PHP CGI binary.

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

- `ScriptAlias /php3/ "c:/path-to-php-dir/"`
- `AddType application/x-httpd-php3 .php3`
- `AddType application/x-httpd-php3 .phtml`
- `Action application/x-httpd-php3 "/php3/php.exe"`

To use the source code highlighting feature, simply create a PHP script file and stick this code in: `<?php show_source ("original_php_script.php3"); ?>`. Substitute `original_php_script.php3` with the name of the file you wish to show the source of. (this is only one way of doing it). *Note:* On Win-Apache all back slashes in a path statement such as: "c:\directory\file.ext", must be converted to forward slashes.

Omni HTTPd 2.0b1 for Windows

This has got to be the easiest config there is:

Step 1: Install Omni 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 = .php3 | actual = c:\path-to-php-dir\php.exe`
Step 5: On the `Mime tab, enter: virtual = wwwserver/stdcgi | actual = .php3`
Step 6: Click `OK`

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

PHP Modules

Table 2-1. PHP Modules

php3_calendar.dll	Calendar conversion functions
php3_crypt.dll	Crypt functions

php3_dbase.dll	DBase functions
php3_dbm.dll	GDBM emulation via Berkely DB2 library
php3_filepro.dll	READ ONLY access to filepro databases
php3_gd.dll	GD Library functions for gif manipulation
php3_hyperwave.dll	HyperWave functions
php3_imap4r2.dll	IMAP 4 functions
php3_ldap.dll	LDAP functions
php3_msql1.dll	mSQL 1 client
php3_msql2.dll	mSQL 2 client
php3_mssql.dll	MSSQL client (requires MSSQL DB-Libraries
php3_mysql.dll	MySQL functions
php3_nsmail.dll	Netscape mail functions
php3_oci73.dll	Oracle functions
php3_snmp.dll	SNMP get and walk functions (NT only!)
php3_zlib.dll	ZLib functions

Problems?

Read the FAQ

Some problems are more common than others. The most common ones are listed in the PHP FAQ, found at http://www.php.net/FAQ.php3

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://www.php.net/bugs.php3.

Other problems

If you are still stuck, someone on the PHP 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 mailing list, send an empty mail to php3-subscribe@lists.php.net (mailto:php3-subscribe@lists.php.net). The mailing list address is `php3@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.

Chapter 3. 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 version, it happens on every invocation.

When using PHP as an Apache module, you can also change the configuration settings using directives in Apache configuration files and .htaccess files.

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_".

With PHP 4.0, there are just a few Apache directives that allow you to change the PHP configuration settings.

php_value *`name`* *`value`*

This sets the value of the specified variable.

php_flag *`name`* *`on|off`*

This is used to set a Boolean configuration option.

php_admin_value *`name`* *`value`*

This sets the value of the specified variable. "Admin" configuration settings can only be set from within the main Apache configuration files, and not from .htaccess files.

php_admin_flag *`name`* *`on|off`*

This is used to set a Boolean configuration option.

You can view the settings of the configuration values in the output of `phpinfo`. You can also access the values of individial configuration settings using `get_cfg_var`.

General Configuration Directives

`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.

`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.

`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.

`cgi_ext` string

`display_errors` boolean

This determines whether errors should be printed to the screen as part of the HTML output or not.

`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.

`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 **`php3_engine off`** in the appropriate places in the `httpd.conf` file, PHP can be enabled or disabled.

`error_log` string

Name of file where script errors should be logged. If the special value `syslog` is used, the errors are sent to the system logger instead. On UNIX, this means syslog(3) and on Windows NT it means the event log. The system logger is not supported on Windows 95.

`error_reporting` integer

Set the error reporting level. The parameter is an integer representing a bit field. Add the values of the error reporting levels you want.

Table 3-1. Error Reporting Levels

bit value	enabled reporting
1	normal errors
2	normal warnings
4	parser errors
8	non-critical style-related warnings

The default value for this directive is 7 (normal errors, normal warnings and parser errors are shown).

`open_basedir` string

Limit the files that can be opened by PHP to the specified directory-tree.

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 an Apache module, open_basedir paths from parent directories are now automatically inherited.

> **Note:** Support for multiple directories was added in 3.0.7.

The default is to allow all files to be opened.

`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.

`ignore_user_abort` string

Off by default. If changed to On scripts will run to completion even if the remote client disconnects in the middle. See also `ignore_user_abort`.

`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 3-1. UNIX include_path

```
include_path=.:/home/httpd/php-lib
```

Example 3-2. Windows include_path

```
include_path=".;c:\www\phplib"
```

The default value for this directive is `.` (only the current directory).

`isapi_ext` string

`log_errors` boolean

Tells whether script error messages should be logged to the server's error log. This option is thus server-specific.

`magic_quotes_gpc` boolean

Sets the magic_quotes state for GPC (Get/Post/Cookie) operations. When magic_quotes are on, all ' (single-quote), " (double quote), \ (backslash) and NUL's are escaped with a backslash automatically. If magic_quotes_sybase is also on, a single-quote is escaped with a single-quote instead of a backslash.

`magic_quotes_runtime` boolean

If *`magic_quotes_runtime`* is enabled, most functions that return data from any sort of external source including databases and text files will have quotes escaped with a backslash. If *`magic_quotes_sybase`* is also on, a single-quote is escaped with a single-quote instead of a backslash.

`magic_quotes_sybase` boolean

If *`magic_quotes_sybase`* is also on, a single-quote is escaped with a single-quote instead of a backslash if *`magic_quotes_gpc`* or *`magic_quotes_runtime`* is enabled.

`max_execution_time` integer

This sets the maximum time in seconds a script is allowed to take before it is terminated by the parser. This helps prevent poorly written scripts from tieing up the server.

`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.

`nsapi_ext` string

`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 have to disable this option. If disabled, you must use the long form of the open tag (**`<?php ?>`**).

`sql.safe_mode` boolean

`track_errors` boolean

If enabled, the last error message will always be present in the global variable $php_errormsg.

`track_vars` boolean

If enabled, GET, POST and cookie input can be found in the global associative arrays $HTTP_GET_VARS, $HTTP_POST_VARS and $HTTP_COOKIE_VARS, respectively.

`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.

`user_dir` string

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

`warn_plus_overloading` boolean

If enabled, this option makes PHP output a warning when the plus (`+`) operator is used on strings. This is to make it easier to find scripts that need to be rewritten to using the string concatenator instead (`.`).

Mail Configuration Directives

`SMTP` string

DNS name or IP address of the SMTP server PHP under Windows should use for mail sent with the `mail` function.

`sendmail_from` string

Which "From:" mail address should be used in mail sent from PHP under Windows.

`sendmail_path` string

Where the **sendmail** program can be found, usually `/usr/sbin/sendmail` or `/usr/lib/sendmail` **configure** does an honest attempt of locating this one for you and set a default, but if it fails, you can set it here.

Systems not using sendmail should set this directive to the sendmail wrapper/replacement their mail system offers, if any. For example, Qmail (http://www.qmail.org/) users can normally set it to `/var/qmail/bin/sendmail`.

Safe Mode Configuration Directives

`safe_mode` boolean

Whether to enable PHP's safe mode. Read the Security chapter for more more information.

`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.

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.

Extension Loading Directives

`enable_dl` boolean

This directive is really only useful in the Apache module version of PHP. You can turn dynamic loading of PHP extensions with `dl` on and off per virtual server or per directory.

The main reason for turning dynamic loading off is security. With dynamic loading, it's possible to ignore all the safe_mode and open_basedir restrictions.

The default is to allow dynamic loading, except when using safe-mode. In safe-mode, it's always imposible to use `dl`.

`extension_dir` string

In what directory PHP should look for dynamically loadable extensions.

`extension` string

Which dynamically loadable extensions to load when PHP starts up.

MySQL Configuration Directives

`mysql.allow_persistent` boolean

Whether to allow persistent MySQL connections.

`mysql.max_persistent` integer

The maximum number of persistent MySQL connections per process.

`mysql.max_links` integer

The maximum number of MySQL connections per process, including persistent connections.

mSQL Configuration Directives

`msql.allow_persistent` boolean

Whether to allow persistent mSQL connections.

`msql.max_persistent` integer

The maximum number of persistent mSQL connections per process.

`msql.max_links` integer

The maximum number of mSQL connections per process, including persistent connections.

Postgres Configuration Directives

`pgsql.allow_persistent` boolean

Whether to allow persistent Postgres connections.

`pgsql.max_persistent` integer

The maximum number of persistent Postgres connections per process.

`pgsql.max_links` integer

The maximum number of Postgres connections per process, including persistent connections.

Sybase Configuration Directives

`sybase.allow_persistent` boolean

Whether to allow persistent Sybase connections.

`sybase.max_persistent` integer

The maximum number of persistent Sybase connections per process.

`sybase.max_links` integer

The maximum number of Sybase connections per process, including persistent connections.

Sybase-CT Configuration Directives

`sybct.allow_persistent` boolean

Whether to allow persistent Sybase-CT connections. The default is on.

`sybct.max_persistent` integer

The maximum number of persistent Sybase-CT connections per process. The default is -1 meaning unlimited.

`sybct.max_links` integer

The maximum number of Sybase-CT connections per process, including persistent connections. The default is -1 meaning unlimited.

`sybct.min_server_severity` integer

Server messages with severity greater than or equal to sybct.min_server_severity will be reported as warnings. This value can also be set from a script by calling `sybase_min_server_severity`. The default is 10 which reports errors of information severity or greater.

`sybct.min_client_severity` integer

Client library messages with severity greater than or equal to sybct.min_client_severity will be reported as warnings. This value can also be set from a script by calling `sybase_min_client_severity`. The default is 10 which effectively disables reporting.

`sybct.login_timeout` integer

The maximum time in seconds to wait for a connection attempt to succeed before returning failure. Note that if max_execution_time has been exceeded when a connection attempt times out, your script will be terminated before it can take action on failure. The default is one minute.

`sybct.timeout` integer

The maximum time in seconds to wait for a select_db or query operation to succeed before returning failure. Note that if max_execution_time has been exceeded when am operation times out, your script will be terminated before it can take action on failure. The default is no limit.

`sybct.hostname` string

The name of the host you claim to be connecting from, for display by sp_who. The default is none.

Informix Configuration Directives

`ifx.allow_persistent` boolean

Whether to allow persistent Informix connections.

`ifx.max_persistent` integer

The maximum number of persistent Informix connections per process.

`ifx.max_links` integer

The maximum number of Informix connections per process, including persistent connections.

`ifx.default_host` string

The default host to connect to when no host is specified in `ifx_connect` or `ifx_pconnect`.

`ifx.default_user` string

The default user id to use when none is specified in `ifx_connect` or `ifx_pconnect`.

`ifx.default_password` string

The default password to use when none is specified in `ifx_connect` or `ifx_pconnect`.

`ifx.blobinfile` boolean

Set to true if you want to return blob columns in a file, false if you want them in memory. You can override the setting at runtime with `ifx_blobinfile_mode`.

`ifx.textasvarchar` boolean

Set to true if you want to return TEXT columns as normal strings in select statements, false if you want to use blob id parameters. You can override the setting at runtime with `ifx_textasvarchar`.

`ifx.byteasvarchar` boolean

Set to true if you want to return BYTE columns as normal strings in select queries, false if you want to use blob id parameters. You can override the setting at runtime with `ifx_textasvarchar`.

`ifx.charasvarchar` boolean

Set to true if you want to trim trailing spaces from CHAR columns when fetching them.

`ifx.nullformat` boolean

Set to true if you want to return NULL columns as the literal string "NULL", false if you want them returned as the empty string "". You can override this setting at runtime with `ifx_nullformat`.

BC Math Configuration Directives

`bcmath.scale` integer

Number of decimal digits for all bcmath functions.

Browser Capability Configuration Directives

`browscap` string

Name of browser capabilities file. See also `get_browser`.

Unified ODBC Configuration Directives

`uodbc.default_db` string

ODBC data source to use if none is specified in `odbc_connect` or `odbc_pconnect`.

`uodbc.default_user` string

User name to use if none is specified in `odbc_connect` or `odbc_pconnect`.

`uodbc.default_pw` string

Password to use if none is specified in `odbc_connect` or `odbc_pconnect`.

`uodbc.allow_persistent` boolean

Whether to allow persistent ODBC connections.

`uodbc.max_persistent` integer

The maximum number of persistent ODBC connections per process.

`uodbc.max_links` integer

The maximum number of ODBC connections per process, including persistent connections.

Chapter 4. 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 it gives 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. This chapter explains the different configuration option combinations and the situations they can be safely used.

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 (http://www.cert.org/advisories/CA-96.11.interpreters_in_cgi_bin_dir.html) 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.php3` 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.php3`. Unfortunately, if the request is originally given in this form, no access checks are made by web server for file `/secret/script.php3`, 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 –disable-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.php3` nor by redirection `http://my.host/dir/script.php3`.

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.php3`. 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 php3-script /cgi-bin/php
AddHandler php3-script .php3
```

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).

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.php3` does not result in opening a file under users home directory, but a file called `~user/doc.php3` 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.php3` will open a file called `doc.php3` 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.php3`.

`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.

Apache module

When PHP is used as an Apache module it inherits Apache’s user permissions (typically those of the "nobody" user).

II. Language Reference

Chapter 5. Basic syntax

Escaping from HTML

There are four ways of escaping from HTML and entering "PHP code mode":

Example 5-1. Ways of escaping from HTML

```
1.  <? echo ("this is the simplest, an SGML processing instruction\n"); ?>

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

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 is only available if short tags have been enabled. This can be done via the `short_tags` function, 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**.

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.

The closing tag for the block will include the immediately trailing newline if one is present.

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 <?# 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 */
 */
?>
```

Chapter 6. Types

PHP supports the following types:

- integer
- floating-point numbers
- string
- array
- object

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.

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 behave in different manners in certain situations, depending on what type it is at the time. For more information, see the section on Type Juggling.

Integers

Integers can be specified using any of the following syntaxes:

```
$a = 1234; # decimal number
$a = -123; # a negative number
$a = 0123; # octal number (equivalent to 83 decimal)
$a = 0x12; # hexadecimal number (equivalent to 18 decimal)
```

Floating point numbers

Floating point numbers ("doubles") can be specified using any of the following syntaxes:

```
$a = 1.234;
$a = 1.2e3;
```

Strings

Strings can be specified using one of two sets of delimiters.

If the string is enclosed in double-quotes ("), variables within the string will be expanded (subject to some parsing limitations). As in C and Perl, the backslash ("\") character can be used in specifying special characters:

Table 6-1. Escaped characters

sequence	meaning
`\n`	newline
`\r`	carriage
`\t`	horizontal tab
`\\`	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

You can escape any other character, but a warning will be issued at the highest warning level.

The second way to delimit a string uses the single-quote ("’") character, which does not do any variable expansion or backslash processing (except for "\\" and "\’" so you can insert backslashes and single-quotes in a singly-quoted string).

String conversion

When a string is evaluated as a numeric value, the resulting value and type are determined as follows.

The string will evaluate as a double 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.

When the first expression is a string, the type of the variable will depend on the second expression.

```
$foo = 1 + "10.5";              // $foo is double (11.5)
$foo = 1 + "-1.3e3";            // $foo is double (-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 = 1 + "10 Little Piggies"; // $foo is integer (11)
$foo = "10.0 pigs " + 1;        // $foo is integer (11)
$foo = "10.0 pigs " + 1.0;      // $foo is double (11)
```

For more information on this conversion, see the Unix manual page for strtod(3).

Arrays

Arrays actually act like both hash tables (associative arrays) and indexed arrays (vectors).

Single Dimension Arrays

PHP supports both scalar and associative arrays. In fact, there is no difference between the two. You can create an array using the `list` or `array` functions, or you can explicitly set each array element value.

```
$a[0] = "abc";
$a[1] = "def";
$b["foo"] = 13;
```

You can also create an array by simply adding values to the array.

```
$a[] = "hello"; // $a[2] == "hello"
$a[] = "world"; // $a[3] == "world"
```

Arrays may be sorted using the `asort`, `arsort`, `ksort`, `rsort`, `sort`, `uasort`, `usort`, and `uksort` functions depending on the type of sort you want.

You can count the number of items in an array using the `count` function.

You can traverse an array using `next` and `prev` functions. Another common way to traverse an array is to use the `each` function.

Multi-Dimensional Arrays

Multi-dimensional arrays are actually pretty simple. For each dimension of the array, you add another [key] value to the end:

```
$a[1]      = $f;               # one dimensional examples
$a["foo"]  = $f;

$a[1][0]     = $f;             # two dimensional
$a["foo"][2] = $f;             # (you can mix numeric and associa-
tive indices)
$a[3]["bar"] = $f;             # (you can mix numeric and associa-
tive indices)

$a["foo"][4]["bar"][0] = $f;   # four dimensional!
```

You can "fill up" multi-dimensional arrays in many ways, but the trickiest one to understand is how to use the `array` command for associative arrays. These two snippets of code fill up the one-dimensional array in the same way:

```
# Example 1:

$a["color"] = "red";
$a["taste"] = "sweet";
$a["shape"] = "round";
$a["name"] = "apple";
$a[3] = 4;


# Example 2:
$a = array(
     "color" => "red",
     "taste" => "sweet",
     "shape" => "round",
     "name"  => "apple",
     3       => 4
);
```

The `array` function can be nested for multi-dimensional arrays:

```
<?
```

```
$a = array(
     "apple"  => array(
          "color"  => "red",
          "taste"  => "sweet",
          "shape"  => "round"
     ),
     "orange"  => array(
          "color"  => "orange",
          "taste"  => "sweet",
          "shape"  => "round"
     ),
     "banana"  => array(
          "color"  => "yellow",
          "taste"  => "paste-y",
          "shape"  => "banana-shaped"
     )
);

echo $a["apple"]["taste"];    # will output "sweet"
?>
```

Objects

Object Initialization

To initialize an object, you use the new statement to instantiate the object to a variable.

```
class foo {
    function do_foo () {
        echo "Doing foo.";
    }
}

$bar = new foo;
$bar->do_foo();
```

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 double, then all operands are evaluated as doubles, and the result will be a double. 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.

```
$foo = "0";  // $foo is string (ASCII 48)
$foo++;      // $foo is the string "1" (ASCII 49)
$foo += 1;   // $foo is now an integer (2)
$foo = $foo + 1.3;  // $foo is now a double (3.3)
$foo = 5 + "10 Little Piggies"; // $foo is integer (15)
$foo = 5 + "10 Small Pigs";     // $foo is integer (15)
```

If the last two examples above seem odd, see String conversion.

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`.

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.

```
$foo = 10;   // $foo is an integer
$bar = (double) $foo;   // $bar is a double
```

The casts allowed are:

- (int), (integer) - cast to integer
- (real), (double), (float) - cast to double
- (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:

```
$foo = (int) $bar;
$foo = ( int ) $bar;
```

It may not be obvious exactly what will happen when casting between certain types. For instance, the following should be noted:

Chapter 7. Variables

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. 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:

```
$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:

```
$a = 1;
$b = 2;

Function Sum () {
    global $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:

```
$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.

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:

```
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:

```
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 are also essential when functions are called recursively. 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:

```
Function Test () {
    static $count = 0;

    $count++;
    echo $count;
    if ($count < 10) {
        Test ();
    }
    $count-;
}
```

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:

```
$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. ie.

```
$$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:

```
echo "$a ${$a}";
```

produces the exact same output as:

```
echo "$a $hello";
```

ie. 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.

Variables from outside PHP

HTML Forms (GET and POST)

When a form is submitted to a PHP script, any variables from that form will be automatically made available to the script by PHP. For instance, consider the following form:

Example 7-1. Simple form variable

```
<form action="foo.php3" method="post">
    Name: <input type="text" name="name"><br>
    <input type="submit">
</form>
```

When submitted, PHP will create the variable `$name`, which will will contain whatever what entered into the *Name:* field on the form.

PHP also understands arrays in the context of form variables, but only in one dimension. You may, for example, group related variables together, or use this feature to retrieve values from a multiple select input:

Example 7-2. More complex form variables

```
<form action="array.php" 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 value="guinness">Guinness
        </select>
    <input type="submit">
```

```
</form>
```

If PHP's track_vars feature is turned on, either by the track_vars configuration setting or the `<?php_track_vars?>` directive, then variables submitted via the POST or GET methods will also be found in the global associative arrays $HTTP_POST_VARS and $HTTP_GET_VARS as appropriate.

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 (http://www.netscape.com/newsref/std/cookie_spec.html). 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. Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data.

If you wish to assign multiple values to a single cookie, just add *[]* to the cookie name. For example:

```
SetCookie ("MyCookie[]", "Testing", time()+3600);
```

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 7-3. SetCookie Example

```
$Count++;
SetCookie ("Count", $Count, time()+3600);
SetCookie ("Cart[$Count]", $item, time()+3600);
```

Environment variables

PHP automatically makes environment variables available as normal PHP variables.

```
echo $HOME;  /* Shows the HOME environment variable, if set. */
```

Since information coming in via GET, POST and Cookie mechanisms also automatically create PHP variables, it is sometimes best to explicitly read a variable from the environment in order to make sure that you are getting the right version. The `getenv` function can be used for this. You can also set an environment variable with the `putenv` function.

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. They are `gettype`, `is_long`, `is_double`, `is_string`, `is_array`, and `is_object`.

Chapter 8. Constants

PHP defines several constants and provides a mechanism for defining more at run-time. Constants are much like variables, save for the two facts that constants must be defined using the `define` function, and that they cannot later be redefined to another value.

The predefined constants (always available) are:

__FILE__

The name of the script file presently being parsed. If used within a file which has been included or required, then the name of the included file is given, and not the name of the parent file.

__LINE__

The number of the line within the current script file which is being parsed. If used within a file which has been included or required, then the position within the included file is given.

PHP_VERSION

The string representation of the version of the PHP parser presently in use; e.g. ’3.0.8-dev’.

PHP_OS

The name of the operating system on which the PHP parser is executing; e.g. ’Linux’.

TRUE

A true value.

FALSE

A false value.

E_ERROR

Denotes an error other than a parsing error from which recovery is not possible.

E_WARNING

Denotes a condition where PHP knows something is wrong, but will continue anyway; these can be caught by the script itself. An example would be an invalid regexp in `ereg`.

E_PARSE

The parser choked on invalid syntax in the script file. Recovery is not possible.

E_NOTICE

Something happened which may or may not be an error. Execution continues. Examples include using an unquoted string as a hash index, or accessing a variable which has not been set.

The E_* constants are typically used with the `error_reporting` function for setting the error reporting level.

You can define additional constants using the `define` function.

Note that these are constants, not C-style macros; only valid scalar data may be represented by a constant.

Example 8-1. Defining Constants

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
?>
```

Example 8-2. Using __FILE__ and __LINE__

```
<?php
function report_error($file, $line, $message) {
    echo "An error occured in $file on line $line: $message.";
}

report_error(__FILE__,__LINE__, "Something went wrong!");
?>
```

Chapter 9. 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 it 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 before
                       the increment, 2*6 = 12 to $f */
$g = double(++$e);  /* assign twice the value of $e after
                       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 (PHP doesn't have a dedicated boolean type). The truth value of expressions in PHP is calculated in a similar way to perl. Any numeric non-zero numeric value is TRUE, zero is FALSE. Be sure to note that negative values are non-zero and are thus considered TRUE! The empty string and the string "0" are FALSE; all other strings are TRUE. With non-scalar values (arrays and objects) - if the value contains no elements it's considered FALSE, otherwise it's considered TRUE.

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 10. Operators

Arithmetic Operators

Remember basic arithmetic from school? These work just like those.

Table 10-1. Arithmetic Operators

example	name	result
$a + $b	Addition	Sum of $a and $b.
$a - $b	Subtraction	Remainder of $b subtracted from $a.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Dividend of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

String Operators

There is only really one string operator – the concatenation operator (".").

```
$a = "Hello ";
$b = $a . "World!"; // now $b = "Hello World!"
```

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:

```
$a = ($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!";
```

Bitwise Operators

Bitwise operators allow you to turn specific bits within an integer on or off.

Table 10-2. 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")

Logical Operators

Table 10-3. Logical Operators

example	name	result
$a and $b	And	True of both $a and $b are true.
$a or $b	Or	True if either $a or $b is true.
$a xor $b	Or	True if either $a or $b is true, but not both.
! $a	Not	True if $a is not true.
$a && $b	And	True of 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 below.)

Comparison Operators

Comparison operators, as their name imply, allow you to compare two values.

Table 10-4. Comparson Operators

example	name	result
$a == $b	Equal	True if $a is equal to $b.
$a != $b	Not equal	True if $a is not equal to $b.
$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 trinary) operator, which operates as in C and many other languages.

```
(expr1) ? (expr2) : (expr3);
```

This expression returns to *`expr2`* if *`expr1`* evalutes to true, and expr3 if *`expr1`* evaluates to false.

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.

The following table lists the precedence of operators with the lowest-precedence operators listed first.

Table 10-5. Operator Precedence

<table>
<tr><th>Associativity</th><th>Operators</th></tr>
<tr><td>left</td><td>,</td></tr>
<tr><td>left</td><td>or</td></tr>
<tr><td>left</td><td>xor</td></tr>
<tr><td>left</td><td>and</td></tr>
<tr><td>right</td><td>print</td></tr>
<tr><td>left</td><td>= += -= *= /= .= %= &= != ~= <<= >>=</td></tr>
<tr><td>left</td><td>? :</td></tr>
<tr><td>left</td><td>||</td></tr>
<tr><td>left</td><td>&&</td></tr>
<tr><td>left</td><td>|</td></tr>
<tr><td>left</td><td>^</td></tr>
<tr><td>left</td><td>&</td></tr>
<tr><td>non-associative</td><td>== !=</td></tr>
<tr><td>non-associative</td><td>< <= > >=</td></tr>
<tr><td>left</td><td><< >></td></tr>
<tr><td>left</td><td>+ - .</td></tr>
<tr><td>left</td><td>* / %</td></tr>
<tr><td>right</td><td>! ~ ++ – (int) (double) (string) (array) (object) @</td></tr>
<tr><td>right</td><td>[</td></tr>
</table>

Associativity	Operators
non-associative	new

Chapter 11. 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 truth value. If *`expr`* evaluates to `TRUE`, PHP will execute statement, and if it evaluates to `FALSE` - it'll ignore it.

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 below).

`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 `elseif`s 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 `if` structures: `if(): ... endif;`

PHP offers a different way to group statements within an `if` statement. This is most commonly used when you nest HTML blocks inside `if` statements, but can be used anywhere. Instead of using curly braces, `if (`*`expr`*`)` should be followed by a colon, the list of one or more statements, and end with `endif;`. Consider the following example:

```
<?php if ($a==5): ?>
A = 5
<?php endif; ?>
```

In the above example, the HTML block "A = 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;
```

`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 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 guarenteed 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.

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 uses the while statement and the `list` and `each` functions for this. See the documentation for these functions for an example.

break

`break` breaks out of the current looping control-structures.

```
$i = 0;
while ($i < 10) {
    if ($arr[$i] == "stop") {
        break;
    }
    $i++;
}
```

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.

```
while (list($key,$value) = each($arr)) {
    if ($key % 2) { // skip even members
        continue;
    }
    do_something_odd ($value);
}
```

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 equals to 0, PHP would execute all of the print statements! If $i equals to 1, PHP would execute the last two print statements, and only if $i equals to 2, you'd get the 'expected' behavior and only 'i equals 2' would be displayed. So, it's important not to forget `break` statements (even though you may want to avoid supplying them on purpose under certain circumstances).

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. 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 scalar type, that is, integer or floating-point numbers and strings. Arrays or objects are meaningless in that context.

require

The `require` statement replaces itself with the specified file, much like the C preprocessor's #include works.

This means that you can't put a `require` statement inside of a loop structure and expect it to include the contents of a different file on each iteration. To do that, use an `include` statement.

```
require 'header.inc';
```

include

The `include` statement includes and evaluates the specified file.

This happens each time the `include` statement is encountered, so you can use an `include` statement within a looping structure to include a number of different file.

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($i = 0; $i < count($files); $i++) {
    include $files[$i];
}
```

`include` differs from `require` in that the include statement is re-evaluated each time it is encountered (and only when it is being executed), whereas the `require` statement is replaced by the required file when it is first encountered, whether the contents of the file will be evaluated or not (for example, if it is inside an if statement whose condition evaluated to false).

Because `include` is a special language construct, you must enclose it within a statement block if it is inside a conditional block.

```
/* 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);
}
```

When the file is evaluated, the parser begins in "HTML-mode" which will output the contents of the file until the first PHP start tag (<?) is encountered.

See also `readfile`, `require`, `virtual`.

Chapter 12. 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.

Functions must be defined before they are referenced.

Returning values

Values are returned by using the optional return statement. Any type may be returned, including lists and objects.

```
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();
```

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 not supported, but a similar effect may be obtained by passing arrays.

```
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.’
```

If you wish to pass a variable by reference to a function which does not do this by default, you may prepend an ampersand to the argument name in the function call:

```
function foo ($bar) {
    $bar .= ’ and something extra.’;
}
$str = ’This is a string, ’;
foo ($str);
echo $str;    // outputs ’This is a string, ’
foo (&$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 = "cappucino") {
    return "Making a cup of $type.\n";
}
echo makecoffee ();
echo makecoffee ("espresso");
```

The output from the above snippet is:

```
Making a cup of cappucino.
Making a cup of espresso.
```

The default value must be a constant expression, not (for example) a variable or class member.

In PHP 4.0 it's also possible to specify `unset` for default argument. This means that the argument will not be set at all, if a value is not supplied.

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.
```

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->PHP3 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 PHP3 form) to call the `old_function`.

Chapter 13. 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

    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.

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.

```
$cart = new Cart;
$cart->add_item("10", 1);
```

This creates an object $cart of the class Cart. The function add_item() of that object is being called to add 1 item of article number 10 to the cart.

Classes can be extensions of other classes. The extended or derived class has all variables and functions of the base class and what you add in the extended definition. This is done using the extends keyword.

```
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)
```

Within functions of a class the variable $this means this object. You have to use $this->something to access any variable or function named something within your current object.

Constructors are functions in a class that are automatically called when you create a new instance of a class. A function becomes a constructor when it has the same name as the class.

```
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 also take arguments and these arguments can be optional, which makes them much more useful.

```
class Constructor_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);
```

Caution

For derived classes, the constructor of the parent class is not automatically called when the derived class's constructor is called.

III. Features

Chapter 14. Error handling

There are 4 types of errors and warnings in PHP. They are:

- 1 - Normal Function Errors
- 2 - Normal Warnings
- 4 - Parser Errors
- 8 - Notices (warnings you can ignore but which may imply a bug in your code)

The above 4 numbers are added up to define an error reporting level. The default error reporting level is 7 which is 1 + 2 + 4, or everything except notices. This level can be changed in the php3.ini file with the error_reporting directive. It can also be set in your Apache httpd.conf file with the php3_error_reporting directive or lastly it may be set at runtime within a script using the `error_reporting` function.

All PHP expressions can also be called with the "@" prefix, which turns off error reporting for that particular expression. If an error occurred during such an expression and the track_errors feature is enabled, you can find the error message in the global variable $php_errormsg.

Chapter 15. Creating GIF images

PHP is not limited to creating just HTML output. It can also be used to create GIF image files, or even more convenient GIF image streams. You will need to compile PHP with the GD library of image functions for this to work.

Example 15-1. GIF creation with PHP

```
<?php
    Header("Content-type: image/gif");
    $string=implode($argv," ");
    $im = imagecreatefromgif("images/button1.gif");
    $orange = ImageColorAllocate($im, 220, 210, 60);
    $px = (imagesx($im)-7.5*strlen($string))/2;
    ImageString($im,3,$px,9,$string,$orange);
    ImageGif($im);
    ImageDestroy($im);
?>
```

This example would be called from a page with a tag like: <img src="button.php3?text"> The above button.php3 script then takes this "text" string an overlays it on top of a base image which in this case is "images/button1.gif" and outputs the resulting image. This is a very convenient way to avoid having to draw new button images every time you want to change the text of a button. With this method they are dynamically generated.

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 variables, $PHP_AUTH_USER, $PHP_AUTH_PW and $PHP_AUTH_TYPE set to the user name, password and authentication type respectively. Only "Basic" authentication is supported at this point. See the `Header` function for more information.

An example script fragment which would force client authentication on a page would be the following:

Example 16-1. HTTP Authentication example

```
<?php
  if(!isset($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\n";
    exit;
  } else {
    echo "Hello $PHP_AUTH_USER.<P>";
    echo "You entered $PHP_AUTH_PW as your password.<P>";
  }
?>
```

Instead of simply printing out the $PHP_AUTH_USER and $PHP_AUTH_PW, you would probably 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.

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. In this case, the $REMOTE_USER variable can be used to identify the externally-authenticated user.

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 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  ac-
cess  this  resource\n";
    exit;
  }

  if(!isset($PHP_AUTH_USER)  ||  ($SeenBefore ==  1  &&  !str-
cmp($OldAuth,  $PHP_AUTH_USER))  )  {
    authenticate();
  }
  else  {
    echo  "Welcome:  $PHP_AUTH_USER<BR>";
    echo  "Old:  $OldAuth";
    echo  "<FORM  ACTION=\"$PHP_SELF\"  METHOD=POST>\n";
    echo  "<INPUT  TYPE=HIDDEN  NAME=\"SeenBefore\"  VALUE=\"1\">\n";
    echo  "<INPUT  TYPE=HIDDEN  NAME=\"OldAuth\"  VALUE=\"$PHP_AUTH_USER\">\n";
    echo  "<INPUT  TYPE=Submit  VALUE=\"Re  Authenticate\">\n";
    echo  "</FORM>\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).

Also note that this does not work using Microsoft's IIS server and the CGI version of PHP due to a limitation of IIS.

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.

Any cookies sent to you from the client will automatically be turned into a PHP variable just like GET and POST method data. If you wish to assign multiple values to a single cookie, just add *[]* to the cookie name. For more details 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.

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:

Example 18-1. File Upload Form

```
<FORM ENCTYPE="multipart/form-data" ACTION="_URL_" METHOD=POST>
<INPUT TYPE="hidden" name="MAX_FILE_SIZE" value="1000">
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. In this destination file, the following variables will be defined upon a successful upload:

- $userfile - The temporary filename in which the uploaded file was stored on the server machine.
- $userfile_name - The original name of the file on the sender's system.
- $userfile_size - The size of the uploaded file in bytes.
- $userfile_type - The mime type of the file if the browser provided this information. An example would be "image/gif".

Note that the "$userfile" part of the above variables is whatever the name of the INPUT field of TYPE=file is in the upload form. In the above upload form example, we chose to call it "userfile".

Files will by default be stored in the server's default temporary directory. This 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.

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 $file_size variable to throw away any files that are either too small or too big. You could use the $file_type variable to throw away any files that didn't match a certain type criteria. 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.

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 in the PHP3.ini file or the corresponding php3_upload_max_filesize Apache .conf directive. The default is 2 Megabytes.

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.

Uploading multiple files

It is 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 18-2. Uploading multiple forms

```
<form action="file-upload.html" 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 `$userfile`, `$userfile_name`, and `$userfile_size` will be formed in the global scope (as well as in $HTTP_POST_VARS). 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, `$userfile_name[0]` would contain the value `review.html`, and `$userfile_name[1]` would contain the value `xwp.out`. Similarly, `$userfile_size[0]` would contain `review.html`'s filesize, and so forth.

PUT method support

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.php3
```

This tells Apache to send all PUT requests for URIs that match the context in which you put this line to the put.php3 script. This assumes, of course, that you have PHP enabled for the .php3 extension and PHP is active.

Inside your put.php3 file you would then do something like this:

```
<? 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 bu 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 support for the "URL fopen wrapper" is enabled when you configure PHP (which it is unless you explicitly pass the `-disable-url-fopen-wrapper` flag to configure), you can use HTTP and FTP URLs with most functions that take a filename as a parameter, including the `require` and `include` statements.

Note: You can't use remote files in `include` and `require` statements on Windows.

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
  $file = fopen("http://www.php.net/", "r");
  if (!$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 as long you connect as a user with the correct access rights, and the file doesn't exist already. 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
  $file = fopen("ftp://ftp.php.net/incoming/outputfile", "w");
  if (!$file) {
    echo "<p>Unable to open remote file for writing.\n";
    exit;
  }
  /* Write the data here. */
  fputs($file, "$HTTP_USER_AGENT\n");
  fclose($file);
?>
```

> **Note:** You might get the idea from the example above to use this technique to write to a remote log, but as mentioned above, you can only write to a new file using the URL fopen() wrappers. 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 php3.ini directive as well as through the corresponding php3_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 diconnect 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 php3.ini directive or the corresponding php3_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).

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 efficently, 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 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 us 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 this is only theoretical – PHP does not yet work as a plug-in for any multithreaded web servers. Work is progressing on support for ISAPI, WSAPI, and NSAPI (on Windows), which will all allow PHP to be used as a plug-in on multithreaded servers like Netscape FastTrack, Microsoft's Internet Information Server (IIS), and O'Reilly's WebSite Pro. When this happens, the behavior will be essentially the same as for the multiprocess model described before.

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.

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!

IV. Function Reference

I. Mail functions

The `mail` function allows you to send mail.

mail

Name

`mail` — send mail

Description

```
bool mail(string to, string subject, string message, string
[additional_headers]);
```

`Mail` automatically mails the message specified in *`message`* to the receiver specified in *`to`*. Multiple recipients can be specified by putting a comma between each address in *`to`*.

Example 1. Sending mail.

```
mail("rasmus@lerdorf.on.ca", "My Subject", "Line 1\nLine 2\nLine 3");
```

If a fourth string argument is passed, this string is inserted at the end of the header. This is typically used to add extra headers. Multiple extra headers are separated with a newline.

Example 2. Sending mail with extra headers.

```
mail("nobody@aol.com", "the subject", $message,
     "From: webmaster@$SERVER_NAME\nReply-To: webmaster@$SERVER_NAME\nX-
Mailer: PHP/" . phpversion());
```

II. Mathematical functions

Introduction

These math functions will only handle values within the range of the long and double types on your computer. If you need to handle bigger numbers, take a look at the arbitrary precision math functions.

Math constants

The following values are defined as constants in PHP by the math extension:

Table 1. Math constants

Constant	Value	Description
M_PI	3.14159265358979323846	The value of ¶ (pi)

Abs

Name

`Abs` — absolute value

Description

```
mixed abs(mixed number);
```

Returns the absolute value of number. If the argument number is float, return type is also float, otherwise it is int.

Acos

Name

`Acos` — arc cosine

Description

```
float acos(float arg);
```

Returns the arc cosine of arg in radians.

See also `asin` and `atan`.

Asin

Name

`Asin` — arc sine

Description

```
float asin(float arg);
```

Returns the arc sine of arg in radians.

See also `acos` and `atan`.

Atan

Name

`Atan` — arc tangent

Description

```
float atan(float arg);
```

Returns the arc tangent of arg in radians.

See also `acos` and `atan`.

Atan2

Name

`Atan2` — arc tangent of two variables

Description

```
float atan2(float y, float x);
```

This function calculates the arc tangent of the two variables x and y. It is similar to calculating the arc tangent of y / x, except that the signs of both arguments are used to determine the quadrant of the result.

The function returns the result in radians, which is between -PI and PI (inclusive).

See also `acos` and `atan`.

base_convert

Name

`base_convert` — convert a number between arbitrary bases

Description

```
strin base_convert(string number, int frombase, int tobase);
```

Returns a string containing *`number`* represented in base *`tobase`*. The base in which *`number`* is given is specified in *`frombase`*. Both *`frombase`* and *`tobase`* have to be between 2 and 36, inclusive. Digits in numbers with a base higher than 10 will be represented with the letters a-z, with a meaning 10, b meaning 11 and z meaning 36.

Example 1. base_convert()

```
$binary = base_convert($hexadecimal, 16, 2);
```

BinDec

Name

`BinDec` — binary to decimal

Description

```
int bindec(string binary_string);
```

Returns the decimal equivalent of the binary number represented by the binary_string argument.

OctDec converts a binary number to a decimal number. The largest number that can be converted is 31 bits of 1's or 2147483647 in decimal.

See also the `decbin` function.

Ceil

Name

`Ceil` — round fractions up

Description

```
int ceil(float number);
```

Returns the next highest integer value from *`number`*. Using `ceil` on integers is absolutely a waste of time.

NOTE: PHP/FI 2's `ceil` returned a float. Use: `$new = (double)ceil($number);` to get the old behaviour.

See also `floor` and `round`.

Cos

Name

`Cos` — cosine

Description

```
float cos(float arg);
```

Returns the cosine of arg in radians.

See also `sin` and `tan`.

DecBin

Name

`DecBin` — decimal to binary

Description

```
string decbin(int number);
```

Returns a string containing a binary representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to a string of 31 1's.

See also the `bindec` function.

DecHex

Name

`DecHex` — decimal to hexadecimal

Description

`string` **`dechex`**`(int` *`number`*`);`

Returns a string containing a hexadecimal representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to "7fffffff".

See also the `hexdec` function.

DecOct

Name

`DecOct` — decimal to octal

Description

`string` **`decoct`**`(int` *`number`*`);`

Returns a string containing an octal representation of the given number argument. The largest number that can be converted is 2147483647 in decimal resulting to "17777777777". See also `octdec`.

Exp

Name

`Exp` — e to the power of...

Description

```
float exp(float arg);
```

Returns e raised to the power of *`arg`*.

See also `pow`.

Floor

Name

`Floor` — round fractions down

Description

```
int floor(float number);
```

Returns the next lowest integer value from *`number`*. Using `floor` on integers is absolutely a waste of time.

NOTE: PHP/FI 2's `floor` returned a float. Use: `$new = (double)floor($number);` to get the old behaviour.

See also `ceil` and `round`.

getrandmax

Name

`getrandmax` — show largest possible random value

Description

```
int getrandmax(void );
```

Returns the maximum value that can be returned by a call to `rand`.

See also `rand`, `srand` `mt_rand`, `mt_srand` and `mt_getrandmax`.

HexDec

Name

`HexDec` — hexadecimal to decimal

Description

```
int hexdec(string hex_string);
```

Returns the decimal equivalent of the hexadecimal number represented by the hex_string argument. HexDec converts a hexadecimal string to a decimal number. The largest number that can be converted is 7fffffff or 2147483647 in decimal.

See also the `dechex` function.

Log

Name

`Log` — natural logarithm

Description

```
float log(float arg);
```

Returns the natural logarithm of arg.

Log10

Name

`Log10` — base-10 logarithm

Description

```
float log10(float arg);
```

Returns the base-10 logarithm of arg.

max

Name

`max` — find highest value

Description

```
mixed max(mixed arg1, mixed arg2, mixed argn);
```

`max` returns the numerically highest of the parameter values.

If the first parameter is an array, `max` returns the highest value in that array. If the first parameter is an integer, string or double, you need at least two parameters and `max` returns the biggest of these values. You can compare an unlimited number of values.

If one or more of the values is a double, all the values will be treated as doubles, and a double is returned. If none of the values is a double, all of them will be treated as integers, and an integer is returned.

min

Name

`min` — find lowest value

Description

```
mixed min(mixed arg1, mixed arg2, mixed argn);
```

`min` returns the numerically lowest of the parameter values.

If the first parameter is an array, `min` returns the lowest value in that array. If the first parameter is an integer, string or double, you need at least two parameters and `min` returns the lowest of these values. You can compare an unlimited number of values.

If one or more of the values is a double, all the values will be treated as doubles, and a double is returned. If none of the values is a double, all of them will be treated as integers, and an integer is returned.

mt_rand

Name

`mt_rand` — generate a better random value

Description

```
int mt_rand([int min], [int max]);
```

Many random number generators of older libcs have dubious or unknown characteristics and are slow. By default, PHP uses the libc random number generator with the `rand` function. `mt_rand` function is a drop-in replacement for this. It uses a random number generator with known characteristics, the Mersenne Twister, which will produce random numbers that should be suitable for cryptographic purposes and is four times faster than what the average libc provides. The Homepage of the Mersenne Twister can be found at http://www.math.keio.ac.jp/~matumoto/emt.html, and an optimized version of the MT source is available from http://www.scp.syr.edu/~marc/hawk/twister.html.

If called without the optional min,max arguments `mt_rand` returns a pseudo-random value between 0 and RAND_MAX. If you want a random number between 5 and 15 (inclusive), for example, use mt_rand(5,15).

Remember to seed the random number generator before use with `mt_srand`.

See also `mt_srand`, `mt_getrandmax`, `srand`, `rand` and `getrandmax`.

mt_srand

Name

`mt_srand` — seed the better random number generator

Description

```
void mt_srand(int seed);
```

Seeds the random number generator with *`seed`*.

```
// seed with microseconds since last "whole" second
mt_srand((double)microtime()*1000000);
$randval = mt_rand();
```

See also `mt_rand`, `mt_getrandmax`, `srand`, `rand` and `getrandmax`.

mt_getrandmax

Name

`mt_getrandmax` — show largest possible random value

Description

```
int mt_getrandmax(void );
```

Returns the maximum value that can be returned by a call to `mt_rand`.

See also `mt_rand`, `mt_srand` `rand`, `srand` and `getrandmax`.

number_format

Name

`number_format` — format a number with grouped thousands

Description

```
string number_format(float number, int decimals, string dec_point, string
thousands_sep);
```

`number_format` returns a formatted version of *`number`*. This function accepts either one, two or four parameters (not three):

If only one parameter is given, *`number`* will be formatted without decimals, but with a comma (",") between every group of thousands.

If two parameters are given, *`number`* will be formatted with *`decimals`* decimals with a dot (".") in front, and a comma (",") between every group of thousands.

If all four parameters are given, *`number`* will be formatted with *`decimals`* decimals, *`dec_point`* instead of a dot (".") before the decimals and *`thousands_sep`* instead of a comma (",") between every group of thousands.

OctDec

Name

`OctDec` — octal to decimal

Description

```
int octdec(string octal_string);
```

Returns the decimal equivalent of the octal number represented by the octal_string argument. OctDec converts an octal string to a decimal number. The largest number that can be converted is 17777777777 or 2147483647 in decimal.

See also `decoct`.

pi

Name

`pi` — get value of pi

Description

```
double pi(void );
```

Returns an approximation of pi.

pow

Name

`pow` — exponential expression

Description

```
float pow(float base, float exp);
```

Returns base raised to the power of exp.

See also `exp`.

rand

Name

`rand` — generate a random value

Description

`int` **`rand`**`(`*`[int min]`*`, `*`[int max]`*`);`

If called without the optional min,max arguments rand() returns a pseudo-random value between 0 and RAND_MAX. If you want a random number between 5 and 15 (inclusive), for example, use rand(5,15).

Remember to seed the random number generator before use with `srand`.

See also `srand`, `getrandmax`, `mt_rand`, `mt_srand` and `mt_getrandmax`.

round

Name

`round` — Rounds a float.

Description

`double` **`round`**`(double `*`val`*`);`

Returns the rounded value of *`val`*.

```
$foo = round( 3.4 );   // $foo == 3.0
$foo = round( 3.5 );   // $foo == 4.0
$foo = round( 3.6 );   // $foo == 4.0
```

See also `ceil` and `floor`.

Sin

Name

`Sin` — sine

Description

```
float sin(float arg);
```

Returns the sine of arg in radians.

See also `cos` and `tan`.

Sqrt

Name

`Sqrt` — square root

Description

```
float sqrt(float arg);
```

Returns the square root of arg.

srand

Name

`srand` — seed the random number generator

Description

void **srand**(int *seed*);

Seeds the random number generator with *`seed`*.

```
// seed with microseconds since last "whole" second
srand((double)microtime()*1000000);
$randval = rand();
```

See also `rand`, `getrandmax`, `mt_rand`, `mt_srand` and `mt_getrandmax`.

Tan

Name

`Tan` — tangent

Description

float **tan**(float *arg*);

Returns the tangent of arg in radians.

See also `sin` and `cos`.

III. MCAL functions

MCAL stands for Modular Calendar Access Library.

Libmcal is a C library for accessing calendars. It’s written to be very modular, with plugable drivers. MCAL is the calendar equivalent of the IMAP module for mailboxes.

With mcal support, a calendar stream can be opened much like the mailbox stream with the IMAP support. Calendars can be local file stores, remote ICAP servers, or other formats that are supported by the mcal library.

Calendar events can be pulled up, queried, and stored. There is also support for calendar triggers (alarms) and reoccuring events.

With libmcal, central calendar servers can be accessed and used, removing the need for any specific database or local file programming.

To get these functions to work, you have to compile PHP with `-with-mcal`. That requires the mcal library to be installed. Grab the latest version from http://mcal.chek.com/ and compile and install it.

The following constants are defined when using the MCAL module: MCAL_SUNDAY, MCAL_MONDAY, MCAL_TUESDAY, MCAL_WEDNESDAY, MCAL_THURSDAY, MCAL_FRIDAY, MCAL_SATURDAY, MCAL_JANUARY, MCAL_FEBRUARY, MCAL_MARCH, MCAL_APRIL, MCAL_MAY, MCAL_JUNE, MCAL_JULY, MCAL_AUGUGT, MCAL_SEPTEMBER, MCAL_OCTOBER, MCAL_NOVEMBER, and MCAL_DECEMBER. Most of the functions use an internal event structure that is unique for each stream. This alleviates the need to pass around large objects between functions. There are convenience functions for setting, initializing, and retrieving the event structure values.

mcal_open

Name

`mcal_open` — Opens up an MCAL connection

Description

```
int mcal_open(string calendar, string username, string password, string
options);
```

Returns an MCAL stream on success, false on error.

`mcal_open` opens up an MCAL connection to the specified *`calendar`* store. If the optional *`options`* is specified, passes the *`options`* to that mailbox also. The streams internal event structure is also initialized upon connection.

mcal_close

Name

`mcal_close` — Close an MCAL stream

Description

```
int mcal_close(int mcal_stream, int flags);
```

Closes the given mcal stream.

mcal_fetch_event

Name

`mcal_fetch_event` — Fetches an event from the calendar stream

Description

`object` **`mcal_fetch_event`** `(int` *`mcal_stream`*`, int` *`event_id`*`, int` *`options`*`);`

`mcal_fetch_event` fetches an event from the calendar stream specified by *`id`*.

Returns an event object consisting of:

- int id - ID of that event.
- int public - TRUE if the event if public, FALSE if it is private.
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - number of minutes before the event to send an alarm/reminder.
- object start - Object containing a datetime entry.
- object end - Object containing a datetime entry.
- int recur_type - recurrence type
- int recur_interval - recurrence interval
- datetime recur_endate - recurrence end date
- int recur_data - recurrence data

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour

- int min - minutes
- int sec - seconds
- int alarm - minutes before event to send an alarm

mcal_list_events

Name

`mcal_list_events` — Return a list of events between two given datetimes

Description

```
array mcal_list_events(int mcal_stream, int [begin_year] , int [begin_month]
, int [begin_day] , int [end_year] , int [end_month] , int [end_day] );
```

Returns an array of event ID's that are between the start and end dates, or if just a stream is given, uses the start and end dates in the global event structure.

`mcal_list_events` function takes in an optional beginning date and an end date for a calendar stream. An array of event id's that are between the given dates or the internal event dates are returned.

mcal_store_event

Name

`mcal_store_event` — Store an event into an MCAL calendar

Description

`int` **`mcal_store_event`**`(int` *`mcal_stream`*`);`

`mcal_store_event` Stores the global event into an MCAL calendar for the given stream.

Returns true on success and false on error.

mcal_delete_event

Name

`mcal_delete_event` — Delete an event from an MCAL calendar

Description

`int` **`mcal_delete_event`**`(int` *`uid`*`);`

`mcal_delete_event` deletes the calendar event specified by the uid.

Returns true.

mcal_snooze

Name

`mcal_snooze` — Turn off an alarm for an event

Description

`int mcal_snooze(int uid);`

`mcal_snooze` turns off an alarm for a calendar event specified by the uid.

Returns true.

mcal_list_alarms

Name

`mcal_list_alarms` — Return a list of events that has an alarm triggered at the given datetime

Description

```
array mcal_list_events(int mcal_stream, int [begin_year] , int [begin_month]
, int [begin_day] , int [end_year] , int [end_month] , int [end_day] );
```

Returns an array of event ID's that has an alarm going off between the start and end dates, or if just a stream is given, uses the start and end dates in the global event structure.

`mcal_list_events` function takes in an optional beginning date and an end date for a calendar stream. An array of event id's that are between the given dates or the internal event dates are returned.

mcal_event_init

Name

`mcal_event_init` — Initializes a streams global event structure

Description

`int` **`mcal_event_init`** `(int` *`stream`*`);`

`mcal_event_init` initializes a streams global event structure. this effectively sets all elements of the structure to 0, or the default settings.

Returns true.

mcal_event_set_category

Name

`mcal_event_set_category` — Sets the category of the streams global event structure

Description

`int` **`mcal_event_set_category`** `(int` *`stream`*`, string` *`category`*`);`

`mcal_event_set_category` sets the streams global event structure's category to the given string.

Returns true.

mcal_event_set_title

Name

`mcal_event_set_title` — Sets the title of the streams global event structure

Description

`int` **`mcal_event_set_title`** `(int` *`stream`*`, string` *`title`*`);`

`mcal_event_set_title` sets the streams global event structure's title to the given string.

Returns true.

mcal_event_set_description

Name

`mcal_event_set_description` — Sets the description of the streams global event structure

Description

`int` **`mcal_event_set_description`** `(int` *`stream`*`, string` *`description`*`);`

`mcal_event_set_description` sets the streams global event structure's description to the given string.

Returns true.

mcal_event_set_start

Name

`mcal_event_set_start` — Sets the start date and time of the streams global event structure

Description

`int` **`mcal_event_set_start`**`(int` *`stream`*`, int` *`year`*`, int` *`month`*`, int` *`[day]`* `, int` *`[hour]`* `, int` *`[min]`* `, int` *`[sec]`* `);`

`mcal_event_set_start` sets the streams global event structure's start date and time to the given values.

Returns true.

mcal_event_set_end

Name

`mcal_event_set_end` — Sets the end date and time of the streams global event structure

Description

`int` **`mcal_event_set_end`**`(int` *`stream`*`, int` *`year`*`, int` *`month`*`, int` *`[day]`* `, int` *`[hour]`* `, int` *`[min]`* `, int` *`[sec]`* `);`

`mcal_event_set_end` sets the streams global event structure's end date and time to the given values.

Returns true.

mcal_event_set_alarm

Name

`mcal_event_set_alarm` — Sets the alarm of the streams global event structure

Description

`int **mcal_event_set_alarm**(int *stream*, int *alarm*);`

`mcal_event_set_alarm` sets the streams global event structure's alarm to the given minutes before the event.

Returns true.

mcal_event_set_class

Name

`mcal_event_set_class` — Sets the class of the streams global event structure

Description

`int **mcal_event_set_class**(int *stream*, int *class*);`

`mcal_event_set_class` sets the streams global event structure's class to the given value. The class is either 0 for public, or 1 for private.

Returns true.

mcal_is_leap_year

Name

`mcal_is_leap_year` — Returns if the given year is a leap year or not

Description

`int` **`mcal_is_leap_year`**`(int` *`year`*`);`

`mcal_is_leap_year` returns 1 if the given year is a leap year, 1 if not.

mcal_days_in_month

Name

`mcal_days_in_month` — Returns the number of days in the given month

Description

`int` **`mcal_days_in_month`**`(int` *`month`*`, int` *`leap year`*`);`

`mcal_days_in_month` Returns the number of days in the given month, taking into account if the given year is a leap year or not.

mcal_date_valid

Name

`mcal_date_valid` — Returns true if the given year, month, day is a valid date

Description

`int` **`mcal_date_valid`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`);`

`mcal_date_valid` Returns true if the given year, month and day is a valid date, false if not.

mcal_time_valid

Name

`mcal_time_valid` — Returns true if the given year, month, day is a valid time

Description

`int` **`mcal_time_valid`**`(int` *`hour`*`, int` *`minutes`*`, int` *`seconds`*`);`

`mcal_time_valid` Returns true if the given hour, minutes and seconds is a valid time, false if not.

mcal_day_of_week

Name

`mcal_day_of_week` — Returns the day of the week of the given date

Description

`int` **`mcal_`**`(int` *`year`*`, int` *`month`*`, int` *`day`*`);`

`mcal_day_of_week` returns the day of the week of the given date

Returns true.

mcal_day_of_year

Name

`mcal_day_of_year` — Returns the day of the year of the given date

Description

int **mcal_**(int *year*, int *month*, int *day*);

`mcal_day_of_year` returns the day of the year of the given date

Returns true.

mcal_date_compare

Name

`mcal_date_compare` — Compares two dates

Description

int **mcal_date_compare**(int *a_year*, int *a_month*, int *a_day*, int *b_year*, int *b_month*, int *b_day*);

`mcal_date_compare` Compares the two given dates, returns <0, 0, >0 if a<b, a==b, a>b respectively

mcal_next_recurrence

Name

`mcal_next_recurrence` — Returns the next recurrence of the event

Description

`int` **`mcal_next_recurrence`** `(int` *`stream`*`, int` *`weekstart`*`, array` *`next`*`);`

`mcal_next_recurrence` returns an object filled with the next date the event occurs, on or after the supplied date. Returns empty date field if event does not occur or something is invalid. Uses weekstart to determine what day is considered the beginning of the week.

Returns true.

mcal_event_set_recur_daily

Name

`mcal_event_set_recur_daily` — Sets the recurrence of the streams global event structure

Description

`int` **`mcal_event_set_recur_daily`** `(int` *`stream`*`, int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`min`*`, int` *`sec`*`, int` *`interval`*`);`

`mcal_event_set_recur_daily` sets the streams global event structure’s recurrence to the given value to be reoccuring on a daily basis, ending at the given date.

mcal_event_set_recur_weekly

Name

`mcal_event_set_recur_weekly` — Sets the recurrence of the streams global event structure

Description

`int` **`mcal_event_set_recur_weekly`** `(int` *`stream`*`, int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`min`*`, int` *`sec`*`, int` *`interval`*`, int` *`weekdays`*`);`

`mcal_event_set_recur_weekly` sets the streams global event structure's recurrence to the given value to be reoccuring on a weekly basis, ending at the given date.

mcal_event_set_recur_monthly_mday

Name

`mcal_event_set_recur_monthly_mday` — Sets the recurrence of the streams global event structure

Description

`int` **`mcal_event_set_recur_monthly_mday`** `(int` *`stream`*`, int` *`year`*`, int` *`month`*`, int` *`day`*`, int` *`hour`*`, int` *`min`*`, int` *`sec`*`, int` *`interval`*`);`

`mcal_event_set_recur_monthly_mday` sets the streams global event structure's recurrence to the given value to be reoccuring on a monthly by month day basis, ending at the given date.

mcal_event_set_recur_monthly_wday

Name

`mcal_event_set_recur_monthly_wday` — Sets the recurrence of the streams global event structure

Description

int **mcal_event_set_recur_monthly_wday** (int *stream*, int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, int *interval*);

`mcal_event_set_recur_monthly_wday` sets the streams global event structure's recurrence to the given value to be reoccuring on a monthly by week basis, ending at the given date.

mcal_event_set_recur_yearly

Name

`mcal_event_set_recur_yearly` — Sets the recurrence of the streams global event structure

Description

int **mcal_event_set_recur_yearly** (int *stream*, int *year*, int *month*, int *day*, int *hour*, int *min*, int *sec*, int *interval*);

`mcal_event_set_recur_yearly` sets the streams global event structure's recurrence to the given value to be reoccuring on a yearly basis,ending at the given date .

mcal_fetch_current_stream_event

Name

`mcal_fetch_current_stream_event` — Returns an object containing the current streams event structure

Description

`int` **`mcal_fetch_current_stream_event`** `(int` *`stream`*`);`

`mcal_event_fetch_current_stream_event` returns the current stream's event structure as an object containing:

- int id - ID of that event.
- int public - TRUE if the event if public, FALSE if it is private.
- string category - Category string of the event.
- string title - Title string of the event.
- string description - Description string of the event.
- int alarm - number of minutes before the event to send an alarm/reminder.
- object start - Object containing a datetime entry.
- object end - Object containing a datetime entry.
- int recur_type - recurrence type
- int recur_interval - recurrence interval
- datetime recur_endate - recurrence end date
- int recur_data - recurrence data

All datetime entries consist of an object that contains:

- int year - year
- int month - month
- int mday - day of month
- int hour - hour

- int min - minutes
- int sec - seconds
- int alarm - minutes before event to send an alarm

IV. Encryption functions

These functions work using mcrypt (ftp://argeas.cs-net.gr/pub/unix/mcrypt/).

This is an interface to the mcrypt library, which supports a wide variety of block algorithms such as DES, TripleDES, Blowfish (default), 3-WAY, SAFER-SK64, SAFER-SK128, TWOFISH, TEA, RC2 and GOST in CBC, OFB, CFB and ECB cipher modes. Additionally, it supports RC6 and IDEA which are considered "non-free".

To use it, download libmcrypt-x.x.tar.gz from here (ftp://argeas.cs-net.gr/pub/unix/mcrypt/) and follow the included installation instructions. You need to compile PHP with the `-with-mcrypt` parameter to enable this extension.

mcrypt can be used to encrypt and decrypt using the above mentioned ciphers. The four important mcrypt commands (`mcrypt_cfb`, `mcrypt_cbc`, `mcrypt_ecb`, and `mcrypt_ofb`) can operate in both modes which are named MCRYPT_ENCRYPT and MCRYPT_DECRYPT, respectively.

Example 1. Encrypt an input value with TripleDES in ECB mode

```
<?php
$key = "this is a very secret key";
$input = "Let us meet at 9 o’clock at the secret place.";

$encrypted_data = mcrypt_ecb(MCRYPT_TripleDES, $key, $input, MCRYPT_ENCRYPT);
?>
```

This example will give you the encrypted data as a string in $encrypted_data.

mcrypt can operate in four cipher modes (CBC, OFB, CFB, and ECB). We will outline the normal use for each of these modes. For a more complete reference and discussion see Applied Cryptography by Schneier (ISBN 0-471-11709-9).

- ECB (electronic codebook) is suitable for random data, such as encrypting other keys. Since data there is short and random, the disadvantages of ECB have a favorable negative effect.
- CBC (cipher block chaining) is especially suitable for encrypting files where the security is increased over ECB significantly.
- CFB (cipher feedback) is the best mode for encrypting byte streams where single bytes must be encrypted.
- OFB (output feedback) is comparable to CFB, but can be used in applications where error propagation cannot be tolerated.

PHP does not support encrypting/decrypting bit streams currently. As of now, PHP only supports handling of strings.

For a complete list of supported ciphers, see the defines at the end of mcrypt.h. The general rule is that you can access the cipher from PHP with MCRYPT_ciphername.

Here is a short list of ciphers which are currently supported by the mcrypt extension. If a cipher is not listed here, but is listed by mcrypt as supported, you can safely assume that this documentation is outdated.

- MCRYPT_BLOWFISH
- MCRYPT_DES
- MCRYPT_TripleDES
- MCRYPT_ThreeWAY
- MCRYPT_GOST
- MCRYPT_CRYPT
- MCRYPT_DES_COMPAT
- MCRYPT_SAFER64
- MCRYPT_SAFER128
- MCRYPT_CAST128
- MCRYPT_TEAN
- MCRYPT_RC2
- MCRYPT_TWOFISH (for older mcrypt 2.x versions)
- MCRYPT_TWOFISH128 (TWOFISHxxx are available in newer 2.x versions)
- MCRYPT_TWOFISH192
- MCRYPT_TWOFISH256
- MCRYPT_RC6
- MCRYPT_IDEA

You must (in CFB and OFB mode) or can (in CBC mode) supply an initialization vector (IV) to the respective cipher function. The IV must be unique and must be the same when decrypting/encrypting. With data which is stored encrypted, you can take the output of a function of the index under which the data is stored (e.g. the MD5 key of the filename). Alternatively, you can transmit the IV together with the encrypted data (see chapter 9.3 of Applied Cryptography by Schneier (ISBN 0-471-11709-9) for a discussion of this topic).

mcrypt_get_cipher_name

Name

`mcrypt_get_cipher_name` — Get the name of the specified cipher

Description

`string` **`mcrypt_get_cipher_name`** `(int` *`cipher`*`);`

`mcrypt_get_cipher_name` is used to get the name of the specified cipher.

`mcrypt_get_cipher_name` takes the cipher number as an argument and returns the name of the cipher or false, if the cipher does not exist.

Example 1. mcrypt_get_cipher_name example

```
<?php
$cipher = MCRYPT_TripleDES;

print mcrypt_get_cipher_name($cipher);
?>
```

The above example will produce:

```
TripleDES
```

mcrypt_get_block_size

Name

`mcrypt_get_block_size` — Get the block size of the specified cipher

Description

`int` **`mcrypt_get_block_size`** `(int` *`cipher`*`);`

`mcrypt_get_block_size` is used to get the size of a block of the specified *`cipher`*.

`mcrypt_get_block_size` takes one argument, the *`cipher`* and returns the size in bytes.

See also: `mcrypt_get_key_size`

mcrypt_get_key_size

Name

`mcrypt_get_key_size` — Get the key size of the specified cipher

Description

`int` **`mcrypt_get_key_size`** `(int` *`cipher`*`);`

`mcrypt_get_key_size` is used to get the size of a key of the specified *`cipher`*.

`mcrypt_get_key_size` takes one argument, the *`cipher`* and returns the size in bytes.

See also: `mcrypt_get_block_size`

mcrypt_create_iv

Name

`mcrypt_create_iv` — Create an initialization vector (IV) from a random source

Description

`string` **`mcrypt_create_iv`**`(int` *`size`*`, int` *`source`*`);`

`mcrypt_create_iv` is used to create an IV.

`mcrypt_create_iv` takes two arguments, *`size`* determines the size of the IV, *`source`* specifies the source of the IV.

The source can be MCRYPT_RAND (system random number generator), MCRYPT_DEV_RANDOM (read data from /dev/random) and MCRYPT_DEV_URANDOM (read data from /dev/urandom). If you use MCRYPT_RAND, make sure to call srand() before to initialize the random number generator.

Example 1. mcrypt_create_iv example

```
<?php
$cipher = MCRYPT_TripleDES;
$block_size = mcrypt_get_block_size($cipher);
$iv = mcrypt_create_iv($block_size, MCRYPT_DEV_RANDOM);
?>
```

mcrypt_cbc

Name

`mcrypt_cbc` — Encrypt/decrypt data in CBC mode

Description

`int` **`mcrypt_cbc`**`(int` *`cipher`*`, string` *`key`*`, string` *`data`*`, int` *`mode`*`, string` *`[iv]`*`);`

`mcrypt_cbc` encrypts or decrypts (depending on *`mode`*) the *`data`* with *`cipher`* and *`key`* in CBC cipher mode and returns the resulting string.

`cipher` is one of the MCRYPT_ciphername constants.

`key` is the key supplied to the algorithm. It must be kept secret.

`data` is the data which shall be encrypted/decrypted.

`mode` is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

`iv` is the optional initialization vector.

See also: `mcrypt_cfb`, `mcrypt_ecb`, `mcrypt_ofb`

mcrypt_cfb

Name

`mcrypt_cfb` — Encrypt/decrypt data in CFB mode

Description

```
int mcrypt_cfb(int cipher, string key, string data, int mode, string iv);
```

`mcrypt_cfb` encrypts or decrypts (depending on `mode`) the `data` with `cipher` and `key` in CFB cipher mode and returns the resulting string.

`cipher` is one of the MCRYPT_ciphername constants.

`key` is the key supplied to the algorithm. It must be kept secret.

`data` is the data which shall be encrypted/decrypted.

`mode` is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

`iv` is the initialization vector.

See also: `mcrypt_cbc`, `mcrypt_ecb`, `mcrypt_ofb`

mcrypt_ecb

Name

`mcrypt_ecb` — Encrypt/decrypt data in ECB mode

Description

`int` **`mcrypt_ecb`**`(int` *`cipher`*`, string` *`key`*`, string` *`data`*`, int` *`mode`*`);`

`mcrypt_ecb` encrypts or decrypts (depending on *`mode`*) the *`data`* with *`cipher`* and *`key`* in ECB cipher mode and returns the resulting string.

`cipher` is one of the MCRYPT_ciphername constants.

`key` is the key supplied to the algorithm. It must be kept secret.

`data` is the data which shall be encrypted/decrypted.

`mode` is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

See also: `mcrypt_cbc`, `mcrypt_cfb`, `mcrypt_ofb`

mcrypt_ofb

Name

`mcrypt_ofb` — Encrypt/decrypt data in OFB mode

Description

`int` **`mcrypt_ofb`**`(int` *`cipher`*`, string` *`key`*`, string` *`data`*`, int` *`mode`*`, string` *`iv`*`);`

`mcrypt_ofb` encrypts or decrypts (depending on *`mode`*) the *`data`* with *`cipher`* and *`key`* in OFB cipher mode and returns the resulting string.

`cipher` is one of the MCRYPT_ciphername constants.

`key` is the key supplied to the algorithm. It must be kept secret.

`data` is the data which shall be encrypted/decrypted.

`mode` is MCRYPT_ENCRYPT or MCRYPT_DECRYPT.

`iv` is the initialization vector.

See also: `mcrypt_cbc`, `mcrypt_cfb`, `mcrypt_ecb`

V. Hash functions

These functions are intended to work with mhash (http://sasweb.de/mhash/).

This is an interface to the mhash library. mhash supports a wide variety of hash algorithms such as MD5, SHA1, GOST, and many others.

To use it, download the mhash distribution from its web site (http://sasweb.de/mhash/) and follow the included installation instructions. You need to compile PHP with the `-with-mhash` parameter to enable this extension.

mhash can be used to create checksums, message digests, and more.

Example 1. Compute the SHA1 key and print it out as hex

```
<?php
$input = "Let us meet at 9 o’ clock at the secret place.";
$hash = mhash(MHASH_SHA1, $input);

print "The hash is ".bin2hex($hash)."\n";

?>
```

This will produce:

```
The hash is d3b85d710d8f6e4e5efd4d5e67d041f9cecedafe
```

For a complete list of supported hashes, refer to the documentation of mhash. The general rule is that you can access the hash algorithm from PHP with MHASH_HASHNAME. For example, to access HAVAL you use the PHP constant MHASH_HAVAL.

Here is a list of hashes which are currently supported by mhash. If a hash is not listed here, but is listed by mhash as supported, you can safely assume that this documentation is outdated.

- MHASH_MD5
- MHASH_SHA1
- MHASH_HAVAL
- MHASH_RIPEMD160
- MHASH_RIPEMD128
- MHASH_SNEFRU

- MHASH_TIGER
- MHASH_GOST
- MHASH_CRC32
- MHASH_CRC32B

mhash_get_hash_name

Name

`mhash_get_hash_name` — Get the name of the specified hash

Description

```
string mhash_get_hash_name (int hash);
```

`mhash_get_hash_name` is used to get the name of the specified hash.

`mhash_get_hash_name` takes the hash id as an argument and returns the name of the hash or false, if the hash does not exist.

Example 1. mhash_get_hash_name example

```
<?php
$hash = MHASH_MD5;

print mhash_get_hash_name($hash);
?>
```

The above example will print out:

```
MD5
```

mhash_get_block_size

Name

`mhash_get_block_size` — Get the block size of the specified hash

Description

`int` **`mhash_get_block_size`**`(int` *`hash`*`);`

`mhash_get_block_size` is used to get the size of a block of the specified *`hash`*.

`mhash_get_block_size` takes one argument, the *`hash`* and returns the size in bytes or false, if the *`hash`* does not exist.

mhash_count

Name

`mhash_count` — Get the highest available hash id

Description

`int` **`mhash_count`**`(void );`

`mhash_count` returns the highest available hash id. Hashes are numbered from 0 to this hash id.

Example 1. Traversing all hashes

```
<?php

$nr = mhash_count();

for($i = 0; $i <= $nr; $i++) {
    echo sprintf("The blocksize of %s is %d\n",
            mhash_get_hash_name($i),
            mhash_get_block_size($i));
}
?>
```

mhash

Name

`mhash` — Compute hash

Description

```
string mhash(int hash, string data);
```

`mhash` applies a hash function specified by *`hash`* to the *`data`* and returns the resulting hash (also called digest).

VI. Miscellaneous functions

These functions were placed here because none of the other categories seemed to fit.

connection_aborted

Name

`connection_aborted` — Returns true if client disconnected

Description

`int` **`connection_aborted`**`(void );`

Returns true if client disconnected. See the Connection Handling description in the Features chapter for a complete explanation.

connection_status

Name

`connection_status` — Returns connection status bitfield

Description

`int` **`connection_status`**`(void );`

Returns the connection status bitfield. See the Connection Handling description in the Features chapter for a complete explanation.

connection_timeout

Name

`connection_timeout` — Return true if script timed out

Description

```
int connection_timeout(void );
```

Returns true if script timed out. See the Connection Handling description in the Features chapter for a complete explanation.

define

Name

`define` — Defines a named constant.

Description

```
int define(string name, mixed value, int [case_insensitive]);
```

Defines a named constant, which is similar to a variable except:

- Constants do not have a dollar sign '$' before them;
- Constants may be 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.

The name of the constant is given by *`name`*; the value is given by *`value`*.

The optional third parameter *`case_insensitive`* is also available. If the value *1* is given, then the constant will be defined case-insensitive. The default behaviour is case-sensitive; i.e. CONSTANT and Constant represent different values.

Example 1. Defining Constants

```
<?php
define("CONSTANT", "Hello world.");
echo CONSTANT; // outputs "Hello world."
?>
```

`define` returns TRUE on success and FALSE if an error occurs.

See also `defined` and the section on Constants.

defined

Name

`defined` — Checks whether a given named constant exists.

Description

```
int defined(string name);
```

Returns TRUE if the named constant given by *`name`* has been defined, false otherwise.

See also `define` and the section on Constants.

die

Name

`die` — Output a message and terminate the current script

Description

```
void die(string message);
```

This language construct outputs a message and terminates parsing of the script. It does not return.

Example 1. die example

```
<?php
$filename = '/path/to/data-file';
$file = fopen($filename, 'r')
  or die "unable to open file ($filename)";
?>
```

eval

Name

`eval` — Evaluate a string as PHP code

Description

```
void eval(string code_str);
```

`eval` evaluates the string given in *`code_str`* as PHP code. Among other things, this can be useful for storing code in a database text field for later execution.

There are some factors to keep in mind when using `eval`. Remember that the string passed must be valid PHP code, including things like terminating statements with a semicolon so the parser doesn't die on the line after the `eval`, and properly escaping things in *`code_str`*.

Also remember that variables given values under `eval` will retain these values in the main script afterwards.

Example 1. eval() example - simple text merge

```
<?php
$string = ’cup’;
$name = ’coffee’;
$str = ’This is a $string with my $name in it.<br>’;
echo $str;
eval( "\$str = \"$str\";" );
echo $str;
?>
```

The above example will show:

```
This is a $string with my $name in it.
This is a cup with my coffee in it.
```

exit

Name

`exit` — Terminate current script

Description

```
void exit(void);
```

This language construct terminates parsing of the script. It does not return.

function_exists

Name

`function_exists` — Return true if the given function has been defined

Description

`int` **`function_exists`**`(string` *`function_name`*`);`

Checks the list of defined functions for *`function_name`*. Returns true if the given function name was found, false otherwise.

get_browser

Name

`get_browser` — Tells what the user's browser is capable of.

Description

`object` **`get_browser`**`(string` *`[user_agent]`*`);`

`get_browser` attempts to determine the capabilities of the user's browser. This is done by looking up the browser's information in the `browscap.ini` file. By default, the value of $HTTP_USER_AGENT is used; however, you can alter this (i.e., look up another browser's info) by passing the optional *`user_agent`* parameter to `get_browser`.

The information is returned in an object, which will contain various data elements representing, for instance, the browser's major and minor version numbers and ID string; true/false values for features such as frames, JavaScript, and cookies; and so forth.

While `browscap.ini` contains information on many browsers, it relies on user updates to keep the database current. The format of the file is fairly self-explanatory.

The following example shows how one might list all available information retrieved about the user's browser.

Example 1. `get_browser` example

```
<?php
function list_array( $array ) {
   while ( list( $key, $value ) = each( $array ) ) {
  $str .= "<b>$key:</b> $value<br>\n";
   }
   return $str;
}
echo "$HTTP_USER_AGENT<hr>\n";
$browser = get_browser();
echo list_array( (array) $browser );
?>
```

The output of the above script would look something like this:

```
Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)<hr>
<b>browser_name_pattern:</b> Mozilla/4\.5.*<br>
<b>parent:</b> Netscape 4.0<br>
<b>platform:</b> Unknown<br>
<b>majorver:</b> 4<br>
<b>minorver:</b> 5<br>
<b>browser:</b> Netscape<br>
<b>version:</b> 4<br>
<b>frames:</b> 1<br>
<b>tables:</b> 1<br>
<b>cookies:</b> 1<br>
<b>backgroundsounds:</b> <br>
<b>vbscript:</b> <br>
<b>javascript:</b> 1<br>
<b>javaapplets:</b> 1<br>
<b>activexcontrols:</b> <br>
<b>beta:</b> <br>
<b>crawler:</b> <br>
```

```
<b>authenticodeupdate:</b> <br>
<b>msn:</b> <br>
```

In order for this to work, your browscap configuration file setting must point to the correct location of the `browscap.ini` file.

For more information (including locations from which you may obtain a `browscap.ini` file), check the PHP FAQ at http://www.php.net/FAQ.html (http://www.php.net/FAQ.php3).

> **Note:** browscap support was added to PHP in version 3.0b2.

ignore_user_abort

Name

`ignore_user_abort` — Set whether a client disconnect should abort script execution

Description

```
int ignore_user_abort(int [setting]);
```

This function sets whether a client disconnect should cause a script to be aborted. It will return the previous setting and can be called without an argument to not change the current setting and only return the current setting. See the Connection Handling section in the Features chapter for a complete description of connection handling in PHP.

iptcparse

Name

`iptcparse` — Parse a binary IPTC http://www.xe.net/iptc/ (http://www.xe.net/iptc/) block into single tags.

Description

`array` **`iptcparse`**`(string` *`iptcblock`*`);`

This function parses a binary IPTC block into its single tags. It returns an array using the tagmarker as an index and the value as the value. It returns false on error or if no IPTC data was found. See `GetImageSize` for a sample.

leak

Name

`leak` — Leak memory

Description

`void` **`leak`**`(int` *`bytes`*`);`

`Leak` leaks the specified amount of memory.

This is useful when debugging the memory manager, which automatically cleans up "leaked" memory when each request is completed.

pack

Name

`pack` — pack data into binary string

Description

`string pack(string format, mixed [args]...);`

Pack given arguments into binary string according to `format`. Returns binary string containing data.

The idea to this function was taken from Perl and all formatting codes work the same as there. The format string consists of format codes followed by an optional repeater argument. The repeater argument can be either an integer value or * for repeating to the end of the input data. For a, A, h, H the repeat count specifies how many characters of one data argument are taken, for @ it is the absolute position where to put the next data, for everything else the repeat count specifies how many data arguments are consumed and packed into the resulting binary string. Currently implemented are

- a NUL-padded string
- A SPACE-padded string
- h Hex string, low nibble first
- H Hex string, high nibble first
- c signed char
- C unsigned char
- s signed short (always 16 bit, machine byte order)
- S unsigned short (always 16 bit, machine byte order)
- n unsigned short (always 16 bit, big endian byte order)
- v unsigned short (always 16 bit, little endian byte order)
- i signed integer (machine dependant size and byte order)
- I unsigned integer (machine dependant size and byte order)
- l signed long (always 32 bit, machine byte order)
- L unsigned long (always 32 bit, machine byte order)
- N unsigned long (always 32 bit, big endian byte order)
- V unsigned long (always 32 bit, little endian byte order)
- f float (machine dependent size and representation)
- d double (machine dependent size and representation)
- x NUL byte
- X Back up one byte

- @ NUL-fill to absolute position

Example 1. pack format string

```
$binarydata = pack("nvc*", 0x1234, 0x5678, 65, 66);
```

The resulting binary string will be 6 bytes long and contain the byte sequence 0x12, 0x34, 0x78, 0x56, 0x41, 0x42.

Note that the distinction between signed and unsigned values only affects the function `unpack`, where as function `pack` gives the same result for signed and unsigned format codes.

Also note that PHP internally stores integral values as signed values of a machine dependant size. If you give it an unsigned integral value too large to be stored that way it is converted to a double which often yields an undesired result.

register_shutdown_function

Name

`register_shutdown_function` — Register a function for execution on shutdown.

Description

`int` **`register_shutdown_function`** `(string` *`func`*`);`

Registers the function named by *`func`* to be executed when script processing is complete.

Common Pitfalls:

Since no output is allowed to the browser in this function, you will be unable to debug it using statements such as print or echo.

serialize

Name

`serialize` — generates a storable representation of a value

Description

`string **serialize**(mixed *value*);`

`serialize` returns a string containing a byte-stream representation of *`value`* that can be stored anywhere.

This is useful for storing or passing PHP values around without losing their type and structure.

To make the serialized string into a PHP value again, use `unserialize`. `serialize` handles the types integer, double, string, array (multidimensional) and object (object properties will be serialized, but methods are lost).

Example 1. serialize example

```
// $session_data contains a multi-dimensional array with session
// information for the current user.  We use serialize() to store
// it in a database at the end of the request.

$conn = odbc_connect("webdb", "php", "chicken");
$stmt = odbc_prepare($conn,
                     "UPDATE sessions SET data = ? WHERE id = ?");
$sqldata = array(serialize($session_data), $PHP_AUTH_USER);
if (!odbc_execute($stmt, &$sqldata)) {
    $stmt = odbc_prepare($conn,
                         "INSERT INTO sessions (id, data) VALUES(?, ?)");
    if (!odbc_execute($stmt, &$sqldata)) {
        /* Something went wrong.  Bitch, whine and moan. */
    }
}
```

sleep

Name

`sleep` — Delay execution

Description

```
void sleep(int seconds);
```

The sleep function delays program execution for the given number of *`seconds`*.

See also `usleep`.

uniqid

Name

`uniqid` — Generate a unique id.

Description

```
int uniqid(string prefix, boolean [lcg]);
```

`uniqid` returns a prefixed unique identifier based on the current time in microseconds. The prefix can be useful for instance if you generate identifiers simultaneously on several hosts that might happen to generate the identifier at the same microsecond. *`prefix`* can be up to 114 characters long.

If the optional *`lcg`* parameter is true, `uniqid` will add additional "combined LCG" entropy at the end of the return value, which should make the results more unique.

With an empty *`prefix`*, the returned string will be 13 characters long. If *`lcg`* is true, it will be 23 characters.

> **Note:** The *`lcg`* parameter is only available in PHP 4 and PHP 3.0.13 and later.

If you need a unique identifier or token and you intend to give out that token to the user via the network (i.e. session cookies), it is recommended that you use something along the lines of

```
$token = md5(uniqid("")); // no random portion
$better_token = md5(uniqid(rand())); // better, difficult to guess
```

This will create a 32 character identifier (a 128 bit hex number) that is extremely difficult to predict.

unpack

Name

`unpack` — unpack data from binary string

Description

```
array unpack(string format, string data);
```

Unpack from binary string into array according to *`format`*. Returns array containing unpacked elements of binary string.

Unpack works slightly different from Perl as the unpacked data is stored in an associative array. To accomplish this you have to name the different format codes and separate them by a slash /.

Example 1. unpack format string

```
$array = unpack("c2chars/nint", $binarydata);
```

The resulting array will contain the entries "chars1", "chars2" and "int".

For an explanation of the format codes see also: `pack`

Note that PHP internally stores integral values as signed. If you unpack a large unsigned long and it is of the same size as PHP internally stored values the result will be a negative number even though unsigned unpacking was specified.

unserialize

Name

`unserialize` — creates a PHP value from a stored representation

Description

```
mixed unserialize(string str);
```

`unserialize` takes a single serialized variable (see `serialize`) and converts it back into a PHP value. The converted value is returned, and can be an integer, double, string, array or object. If an object was serialized, its methods are not preserved in the returned value.

Example 1. unserialize example

```
// Here, we use unserialize() to load session data from a database
// into $session_data.  This example complements the one described
// with serialize.

$conn = odbc_connect("webdb", "php", "chicken");
$stmt = odbc_prepare($conn, "SELECT data FROM sessions WHERE id = ?");
$sqldata = array($PHP_AUTH_USER);
if (!odbc_execute($stmt, &$sqldata) || !odbc_fetch_into($stmt, &$tmp)) {
    // if the execute or fetch fails, initialize to empty array
    $session_data = array();
} else {
    // we should now have the serialized data in $tmp[0].
    $session_data = unserialize($tmp[0]);
    if (!is_array($session_data)) {
        // something went wrong, initialize to empty array
        $session_data = array();
    }
}
```

usleep

Name

`usleep` — Delay execution in microseconds

Description

`void **usleep**(int *micro_seconds*);`

The sleep function delays program execution for the given number of *`micro_seconds`*.

See also `sleep`.

VII. mSQL functions

msql

Name

`msql` — send mSQL query

Description

```
int msql(string database, string query, int link_identifier);
```

Returns a positive mSQL query identifier to the query result, or false on error.

`msql` selects a database and executes a query on it. If the optional link identifier isn't specified, the function will try to find an open link to the mSQL server and if no such link is found it'll try to create one as if `msql_connect` was called with no arguments (see `msql_connect`).

msql_affected_rows

Name

`msql_affected_rows` — returns number of affected rows

Description

```
int msql_affected_rows(int query_identifier);
```

Returns number of affected ("touched") rows by a specific query (i.e. the number of rows returned by a SELECT, the number of rows modified by an update, or the number of rows removed by a delete).

See also: `msql_query`

msql_close

Name

`msql_close` — close mSQL connection

Description

`int` **`msql_close`**`(int` *`link_identifier`*`);`

Returns true on success, false on error.

msql_close() closes the link to a mSQL database that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note that this isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

msql_close() will not close persistent links generated by `msql_pconnect`.

See also: `msql_connect` and `msql_pconnect`.

msql_connect

Name

`msql_connect` — open mSQL connection

Description

`int` **`msql_connect`**`(string` *`hostname`*`);`

Returns a positive mSQL link identifier on success, or false on error.

msql_connect() establishes a connection to a mSQL server. The hostname argument is optional, and if it's missing, localhost is assumed.

In case a second call is made to msql_connect() with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned.

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `msql_close`.

See also `msql_pconnect`, `msql_close`.

msql_create_db

Name

`msql_create_db` — create mSQL database

Description

`int` **`msql_create_db`**`(string` *`database name`*`, int` *`[link_identifier]`* `);`

msql_create_db() attempts to create a new database on the server associated with the specified link identifier.

See also: `msql_drop_db`.

msql_createdb

Name

`msql_createdb` — create mSQL database

Description

`int` **`msql_createdb`**`(string` *`database name`*`, int` *`[link_identifier]`* `);`

Identical to `msql_create_db`.

msql_data_seek

Name

`msql_data_seek` — move internal row pointer

Description

`int` **`msql_data_seek`**`(int` *`query_identifier`*`, int` *`row_number`*`);`

Returns true on success, false on failure.

msql_data_seek() moves the internal row pointer of the mSQL result associated with the specified query identifier to pointer to the specifyed row number. The next call to `msql_fetch_row` would return that row.

See also: `msql_fetch_row`.

msql_dbname

Name

`msql_dbname` — get current mSQL database name

Description

`string` **`msql_dbname`**`(int` *`query_identifier`*`, int` *`i`*`);`

`msql_dbname` returns the database name stored in position *`i`* of the result pointer returned from the `msql_listdbs` function. The `msql_numrows` function can be used to determine how many database names are available.

msql_drop_db

Name

`msql_drop_db` — drop (delete) mSQL database

Description

`int` **`msql_drop_db`**`(string` *`database_name`*`, int` *`link_identifier`*`);`

Returns true on success, false on failure.

msql_drop_db() attempts to drop (remove) an entire database from the server associated with the specified link identifier.

See also: `msql_create_db`.

msql_dropdb

Name

`msql_dropdb` — drop (delete) mSQL database

Description

See `msql_drop_db`.

msql_error

Name

`msql_error` — returns error message of last msql call

Description

`string` **`msql_error`**`( );`

Errors coming back from the mSQL database backend no longer issue warnings. Instead, use these functions to retrieve the error string.

msql_fetch_array

Name

`msql_fetch_array` — fetch row as array

Description

`int` **`msql_fetch_array`**`(int` *`query_identifier`*`, int` *`[result_type]`* `);`

Returns an array that corresponds to the fetched row, or false if there are no more rows.

`msql_fetch_array` is an extended version of `msql_fetch_row`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

The second optional argument *`result_type`* in `msql_fetch_array` is a constant and can take the following values: MSQL_ASSOC, MSQL_NUM, and MYSQL_BOTH.

Be careful if you are retrieving results from a query that may return a record that contains only one field that has a value of 0 (or an empty string, or NULL).

An important thing to note is that using `msql_fetch_array` is NOT significantly slower than using `msql_fetch_row`, while it provides a significant added value.

For further details, also see `msql_fetch_row`

msql_fetch_field

Name

`msql_fetch_field` — get field information

Description

```
object msql_fetch_field(int query_identifier, int field_offset);
```

Returns an object containing field information

msql_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retreived by msql_fetch_field() is retreived.

The properties of the object are:

- name - column name
- table - name of the table the column belongs to
- not_null - 1 if the column cannot be null
- primary_key - 1 if the column is a primary key
- unique - 1 if the column is a unique key

- type - the type of the column

See also `msql_field_seek`.

msql_fetch_object

Name

`msql_fetch_object` — fetch row as object

Description

int **msql_fetch_object** (int *query_identifier*, int *[result_type]*);

Returns an object with properties that correspond to the fetched row, or false if there are no more rows.

`msql_fetch_object` is similar to `msql_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The optional second argument *`result_type`* in `msql_fetch_array` is a constant and can take the following values: MSQL_ASSOC, MSQL_NUM, and MSQL_BOTH.

Speed-wise, the function is identical to `msql_fetch_array`, and almost as quick as `msql_fetch_row` (the difference is insignificant).

See also: `msql_fetch_array` and `msql_fetch_row`.

msql_fetch_row

Name

`msql_fetch_row` — get row as enumerated array

Description

array **`msql_fetch_row`**`(int` *`query_identifier`*`);`

Returns an array that corresponds to the fetched row, or false if there are no more rows.

msql_fetch_row() fetches one row of data from the result associated with the specified query identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to msql_fetch_row() would return the next row in the result set, or false if there are no more rows.

See also: `msql_fetch_array`, `msql_fetch_object`, `msql_data_seek`, and `msql_result`.

msql_fieldname

Name

`msql_fieldname` — get field name

Description

string **`msql_fieldname`**`(int` *`query_identifier`*`, int` *`field`*`);`

msql_fieldname() returns the name of the specified field. *`query_identifier`* is the query identifier, and *`field`* is the field index. `msql_fieldname($result, 2);` will return the name of the second field in the result associated with the result identifier.

msql_field_seek

Name

`msql_field_seek` — set field offset

Description

`int` **`msql_field_seek`**`(int` *`query_identifier`*`, int` *`field_offset`*`);`

Seeks to the specified field offset. If the next call to `msql_fetch_field` won't include a field offset, this field would be returned.

See also: `msql_fetch_field`.

msql_fieldtable

Name

`msql_fieldtable` — get table name for field

Description

`int` **`msql_fieldtable`**`(int` *`query_identifier`*`, int` *`field`*`);`

Returns the name of the table *`field`* was fetched from.

msql_fieldtype

Name

`msql_fieldtype` — get field type

Description

`string` **`msql_fieldtype`**`(int` *`query_identifier`*`, int` *`i`*`);`

msql_fieldtype() is similar to the `msql_fieldname` function. The arguments are identical, but the field type is returned. This will be one of "int", "string" or "real".

msql_fieldflags

Name

`msql_fieldflags` — get field flags

Description

`string` **`msql_fieldflags`**`(int` *`query_identifier`*`, int` *`i`*`);`

msql_fieldflags() returns the field flags of the specified field. Currently this is either, "not null", "primary key", a combination of the two or "" (an empty string).

msql_fieldlen

Name

`msql_fieldlen` — get field length

Description

`int` **`msql_fieldlen`**`(int` *`query_identifier`*`, int` *`i`*`);`

msql_fieldlen() returns the length of the specified field.

msql_free_result

Name

`msql_free_result` — free result memory

Description

`int` **`msql_free_result`**`(int` *`query_identifier`*`);`

`msql_free_result` frees the memory associated with *`query_identifier`*. When PHP completes a request, this memory is freed automatically, so you only need to call this function when you want to make sure you don't use too much memory while the script is running.

msql_freeresult

Name

`msql_freeresult` — free result memory

Description

See `msql_free_result`

msql_list_fields

Name

`msql_list_fields` — list result fields

Description

int **`msql_list_fields`**`(string` *`database`*`, string` *`tablename`*`);`

msql_list_fields() retrieves information about the given tablename. Arguments are the database name and the table name. A result pointer is returned which can be used with `msql_fieldflags`, `msql_fieldlen`, `msql_fieldname`, and `msql_fieldtype`. A query identifier is a positive integer. The function returns `-1` if a error occurs. A string describing the error will be placed in `$phperrmsg`, and unless the function was called as `@msql_list_fields()` then this error string will also be printed out.

See also `msql_error`.

msql_listfields

Name

`msql_listfields` — list result fields

Description

See `msql_list_fields`.

msql_list_dbs

Name

`msql_list_dbs` — list mSQL databases on server

Description

`int` **`msql_list_dbs`**`(void);`

`msql_list_dbs` will return a result pointer containing the databases available from the current msql daemon. Use the `msql_dbname` function to traverse this result pointer.

msql_listdbs

Name

`msql_listdbs` — list mSQL databases on server

Description

See `msql_list_dbs`.

msql_list_tables

Name

`msql_list_tables` — list tables in an mSQL database

Description

`int` **`msql_list_tables`**`(string` *`database`*`);`

`msql_list_tables` takes a database name and result pointer much like the `msql` function. The `msql_tablename` function should be used to extract the actual table names from the result pointer.

msql_listtables

Name

`msql_listtables` — list tables in an mSQL database

Description

See `msql_list_tables`.

msql_num_fields

Name

`msql_num_fields` — get number of fields in result

Description

```
int msql_num_fields(int query_identifier);
```

msql_num_fields() returns the number of fields in a result set.

See also: `msql`, `msql_query`, `msql_fetch_field`, and `msql_num_rows`.

msql_num_rows

Name

`msql_num_rows` — get number of rows in result

Description

`int` **`msql_num_rows`**`(int` *`query_identifier`*`);`

msql_num_rows() returns the number of rows in a result set.

See also: `msql`, `msql_query`, and `msql_fetch_row`.

msql_numfields

Name

`msql_numfields` — get number of fields in result

Description

`int` **`msql_numfields`**`(int` *`query_identifier`*`);`

Identical to `msql_num_fields`.

msql_numrows

Name

`msql_numrows` — get number of rows in result

Description

`int` **`msql_numrows`**`(void);`

Identical to `msql_num_rows`.

msql_pconnect

Name

`msql_pconnect` — open persistent mSQL connection

Description

`int msql_pconnect(string hostname);`

Returns a positive mSQL persistent link identifier on success, or false on error.

msql_pconnect() acts very much like `msql_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`msql_close` will not close links established by msql_pconnect()).

This type of links is therefore called 'persistent'.

msql_query

Name

`msql_query` — send mSQL query

Description

`int **msql_query**(string *query*, int *link_identifier*);`

msql_query() sends a query to the currently active database on the server that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed. If no link is open, the function tries to establish a link as if `msql_connect` was called, and use it.

Returns a positive mSQL query identifier on success, or false on error.

See also: `msql`, `msql_select_db`, and `msql_connect`.

msql_regcase

Name

`msql_regcase` — make regular expression for case insensitive match

Description

See `sql_regcase`.

msql_result

Name

`msql_result` — get result data

Description

`int **msql_result**(int *query_identifier*, int *i*, mixed *field*);`

Returns the contents of the cell at the row and offset in the specified mSQL result set.

msql_result() returns the contents of one cell from a mSQL result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than msql_result(). Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: `msql_fetch_row`, `msql_fetch_array`, and `msql_fetch_object`.

msql_select_db

Name

`msql_select_db` — select mSQL database

Description

`int` **`msql_select_db`**`(string` *`database_name`*`, int` *`link_identifier`*`);`

Returns true on success, false on error.

msql_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if msql_connect() was called, and use it.

Every subsequent call to `msql_query` will be made on the active database.

See also: `msql_connect`, `msql_pconnect`, and `msql_query`.

msql_selectdb

Name

`msql_selectdb` — select mSQL database

Description

See `msql_select_db`.

msql_tablename

Name

`msql_tablename` — get table name of field

Description

`string` **`msql_tablename`**`(int` *`query_identifier`*`, int` *`field`*`);`

msql_tablename() takes a result pointer returned by the `msql_list_tables` function as well as an integer index and returns the name of a table. The `msql_numrows` function may be used to determine the number of tables in the result pointer.

Example 1. msql_tablename() example

```
<?php
msql_connect ("localhost");
$result = msql_list_tables("wisconsin");
$i = 0;
while ($i < msql_numrows($result)) {
    $tb_names[$i] = msql_tablename($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

VIII. Microsoft SQL Server functions

mssql_close

Name

`mssql_close` — close MS SQL Server connection

Description

`int **mssql_close**(int *link_identifier*);`

Returns: true on success, false on error

mssql_close() closes the link to a MS SQL Server database that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note that this isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

mssql_close() will not close persistent links generated by mssql_pconnect().

See also: `mssql_connect`, `mssql_pconnect`.

mssql_connect

Name

`mssql_connect` — open MS SQL server connection

Description

`int **mssql_connect**(string *servername*, string *username*, string *password*);`

Returns: A positive MS SQL link identifier on success, or false on error.

mssql_connect() establishes a connection to a MS SQL server. The servername argument has to be a valid servername that is defined in the 'interfaces' file.

In case a second call is made to mssql_connect() with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned.

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `mssql_close`.

See also `mssql_pconnect`, `mssql_close`.

mssql_data_seek

Name

`mssql_data_seek` — move internal row pointer

Description

```
int mssql_data_seek(int result_identifier, int row_number);
```

Returns: true on success, false on failure

mssql_data_seek() moves the internal row pointer of the MS SQL result associated with the specified result identifier to pointer to the specifyed row number. The next call to `mssql_fetch_row` would return that row.

See also: `mssql_data_seek`.

mssql_fetch_array

Name

`mssql_fetch_array` — fetch row as array

Description

int **mssql_fetch_array**(int *result*);

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

mssql_fetch_array() is an extended version of `mssql_fetch_row`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

An important thing to note is that using mssql_fetch_array() is NOT significantly slower than using mssql_fetch_row(), while it provides a significant added value.

For further details, also see `mssql_fetch_row`

mssql_fetch_field

Name

`mssql_fetch_field` — get field information

Description

object **mssql_fetch_field**(int *result*, int *field_offset*);

Returns an object containing field information.

mssql_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retreived by mssql_fetch_field() is retreived.

The properties of the object are:

- name - column name. if the column is a result of a function, this property is set to computed#N, where #N is a serial number.
- column_source - the table from which the column was taken
- max_length - maximum length of the column

- numeric - 1 if the column is numeric

See also `mssql_field_seek`

mssql_fetch_object

Name

`mssql_fetch_object` — fetch row as object

Description

`int **mssql_fetch_object** (int *result*);`

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

mssql_fetch_object() is similar to `mssql_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

Speed-wise, the function is identical to `mssql_fetch_array`, and almost as quick as `mssql_fetch_row` (the difference is insignificant).

See also: `mssql_fetch-array` and `mssql_fetch-row`.

mssql_fetch_row

Name

`mssql_fetch_row` — get row as enumerated array

Description

`array` **`mssql_fetch_row`**`(int` *`result`*`);`

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

mssql_fetch_row() fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to mssql_fetch_rows() would return the next row in the result set, or false if there are no more rows.

See also: `mssql_fetch_array`, `mssql_fetch_object`, `mssql_data_seek`, `mssql_fetch_lengths`, and `mssql_result`.

mssql_field_seek

Name

`mssql_field_seek` — set field offset

Description

`int` **`mssql_field_seek`**`(int` *`result`*`, int` *`field_offset`*`);`

Seeks to the specified field offset. If the next call to `mssql_fetch_field` won't include a field offset, this field would be returned.

See also: `mssql_fetch_field`.

mssql_free_result

Name

`mssql_free_result` — free result memory

Description

`int` **`mssql_free_result`** `(int` *`result`*`);`

`mssql_free_result` only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script, you may call `mssql_free_result` with the result identifier as an argument and the associated result memory will be freed.

mssql_num_fields

Name

`mssql_num_fields` — get number of fields in result

Description

`int` **`mssql_num_fields`** `(int` *`result`*`);`

mssql_num_fields() returns the number of fields in a result set.

See also: `mssql_db_query`, `mssql_query`, `mssql_fetch_field`, `mssql_num_rows`.

mssql_num_rows

Name

`mssql_num_rows` — get number of rows in result

Description

`int` **`mssql_num_rows`**`(string` *`result`*`);`

mssql_num_rows() returns the number of rows in a result set.

See also: `mssql_db_query`, `mssql_query` and, `mssql_fetch_row`.

mssql_pconnect

Name

`mssql_pconnect` — open persistent MS SQL connection

Description

`int` **`mssql_pconnect`**`(string` *`servername`*`, string` *`username`*`, string` *`password`*`);`

Returns: A positive MS SQL persistent link identifier on success, or false on error

mssql_pconnect() acts very much like `mssql_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`mssql_close` will not close links established by `mssql_pconnect()`).

This type of links is therefore called ’persistent’.

mssql_query

Name

`mssql_query` — send MS SQL query

Description

`int` **`mssql_query`**`(string` *`query`*`, int` *`link_identifier`*`);`

Returns: A positive MS SQL result identifier on success, or false on error.

mssql_query() sends a query to the currently active database on the server that’s associated with the specified link identifier. If the link identifier isn’t specified, the last opened link is assumed. If no link is open, the function tries to establish a link as if `mssql_connect` was called, and use it.

See also: `mssql_db_query`, `mssql_select_db`, and `mssql_connect`.

mssql_result

Name

`mssql_result` — get result data

Description

`int` **`mssql_result`**`(int` *`result`*`, int` *`i`*`, mixed` *`field`*`);`

Returns: The contents of the cell at the row and offset in the specified MS SQL result set.

mssql_result() returns the contents of one cell from a MS SQL result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than mssql_result(). Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: `mssql_fetch_row`, `mssql_fetch_array`, and `mssql_fetch_object`.

mssql_select_db

Name

`mssql_select_db` — select MS SQL database

Description

`int` **`mssql_select_db`**`(string` *`database_name`*`, int` *`link_identifier`*`);`

Returns: true on success, false on error

mssql_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if `mssql_connect` was called, and use it.

Every subsequent call to `mssql_query` will be made on the active database.

See also: `mssql_connect`, `mssql_pconnect`, and `mssql_query`

IX. MySQL functions

These functions allow you to access MySQL database servers.

More information about MySQL can be found at http://www.mysql.com/.

mysql_affected_rows

Name

`mysql_affected_rows` — Get number of affected rows in previous MySQL operation

Description

```
int mysql_affected_rows(int [link_identifier] );
```

`mysql_affected_rows` returns the number of rows affected by the last INSERT, UPDATE or DELETE query on the server associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

If the last query was a DELETE query with no WHERE clause, all of the records will have been deleted from the table but this function will return zero.

This command is not effective for SELECT statements, only on statements which modify records. To retrieve the number of rows returned from a SELECT, use `mysql_num_rows`.

mysql_close

Name

`mysql_close` — close MySQL connection

Description

```
int mysql_close(int [link_identifier] );
```

Returns: true on success, false on error

`mysql_close` closes the link to a MySQL database that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note that this isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

`mysql_close` will not close persistent links generated by `mysql_pconnect`.

See also: `mysql_connect`, and `mysql_pconnect`.

mysql_connect

Name

`mysql_connect` — Open a connection to a MySQL Server

Description

```
int mysql_connect(string  [hostname [:port] [:/path/to/socket] ]  , string
[username] , string [password] );
```

Returns: A positive MySQL link identifier on success, or false on error.

`mysql_connect` establishes a connection to a MySQL server. All of the arguments are optional, and if they're missing, defaults are assumed ('localhost', user name of the user that owns the server process, empty password).

> The hostname string can also include a port number. eg. "hostname:port" or a path to a socket eg. ":/path/to/socket" for the localhost.
>
> **Note:** Support for ":port" wass added in 3.0B4.
>
> Support for the ":/path/to/socket" was added in 3.0.10.

In case a second call is made to `mysql_connect` with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned.

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `mysql_close`.

See also `mysql_pconnect`, and `mysql_close`.

mysql_create_db

Name

`mysql_create_db` — Create a MySQL database

Description

`int` **`mysql_create_db`**`(string` *`database name`*`, int` *`[link_identifier]`* `);`

`mysql_create_db` attempts to create a new database on the server associated with the specified link identifier.

See also: `mysql_drop_db`. For downwards compatibility `mysql_createdb` can also be used.

mysql_data_seek

Name

`mysql_data_seek` — Move internal result pointer

Description

`int` **`mysql_data_seek`**`(int` *`result_identifier`*`, int` *`row_number`*`);`

Returns: true on success, false on failure

`mysql_data_seek` moves the internal row pointer of the MySQL result associated with the specified result identifier to point to the specified row number. The next call to `mysql_fetch_row` would return that row.

`Row_number` starts at 0.

mysql_db_query

Name

`mysql_db_query` — Send an MySQL query to MySQL

Description

`int **mysql_db_query**(string *database*, string *query*, int *[link_identifier]* );`

Returns: A positive MySQL result identifier to the query result, or false on error.

`mysql_db_query` selects a database and executes a query on it. If the optional link identifier isn't specified, the function will try to find an open link to the MySQL server and if no such link is found it'll try to create one as if `mysql_connect` was called with no arguments

See also `mysql_connect`. For downwards compatibility `mysql` can also be used.

mysql_drop_db

Name

`mysql_drop_db` — Drop (delete) a MySQL database

Description

`int **mysql_drop_db**(string *database_name*, int *[link_identifier]* );`

Returns: true on success, false on failure.

`mysql_drop_db` attempts to drop (remove) an entire database from the server associated with the specified link identifier.

See also: `mysql_create_db`. For downward compatibility `mysql_dropdb` can also be used.

mysql_errno

Name

`mysql_errno` — Returns the number of the error message from previous MySQL operation

Description

```
int mysql_errno(int [link_identifier] );
```

Errors coming back from the mySQL database backend no longer issue warnings. Instead, use these functions to retrieve the error number.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

See also: `mysql_error`

mysql_error

Name

`mysql_error` — Returns the text of the error message from previous MySQL operation

Description

```
string mysql_error(int [link_identifier] );
```

Errors coming back from the mySQL database backend no longer issue warnings. Instead, use these functions to retrieve the error string.

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

See also: `mysql_errno`

mysql_fetch_array

Name

`mysql_fetch_array` — Fetch a result row as an associative array

Description

```
array mysql_fetch_array(int result, int [result_type] );
```

Returns an array that corresponds to the fetched row, or false if there are no more rows.

`mysql_fetch_array` is an extended version of `mysql_fetch_row`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

If two or more columns of the result have the same field names, the last column will take precedence. To access the other column(s) of the same name, you must the numeric index of the column or make an alias for the column.

```
select t1.f1 as foo t2.f1 as bar from t1, t2
```

An important thing to note is that using `mysql_fetch_array` is NOT significantly slower than using `mysql_fetch_row`, while it provides a significant added value.

The optional second argument *`result_type`* in `mysql_fetch_array` is a constant and can take the following values: MYSQL_ASSOC, MYSQL_NUM, and MYSQL_BOTH.

For further details, also see `mysql_fetch_row`

Example 1. mysql fetch array

```
<?php
mysql_connect($host,$user,$password);
$result = mysql_db_query("database","select * from table");
while($row = mysql_fetch_array($result)) {
    echo $row["user_id"];
    echo $row["fullname"];
}
mysql_free_result($result);
?>
```

mysql_fetch_field

Name

`mysql_fetch_field` — Get column information from a result and return as an object

Description

```
object mysql_fetch_field(int result, int [field_offset] );
```

Returns an object containing field information.

`mysql_fetch_field` can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retrieved by `mysql_fetch_field` is retrieved.

The properties of the object are:

- name - column name
- table - name of the table the column belongs to
- max_length - maximum length of the column
- not_null - 1 if the column cannot be null
- primary_key - 1 if the column is a primary key
- unique_key - 1 if the column is a unique key
- multiple_key - 1 if the column is a non-unique key
- numeric - 1 if the column is numeric
- blob - 1 if the column is a BLOB
- type - the type of the column
- unsigned - 1 if the column is unsigned
- zerofill - 1 if the column is zero-filled

See also `mysql_field_seek`

mysql_fetch_lengths

Name

`mysql_fetch_lengths` — Get the length of each output in a result

Description

```
array mysql_fetch_lengths(int result);
```

Returns: An array that corresponds to the lengths of each field in the last row fetched by `mysql_fetch_row`, or false on error.

`mysql_fetch_lengths` stores the lengths of each result column in the last row returned by `mysql_fetch_row`, `mysql_fetch_array`, and `mysql_fetch_object` in an array, starting at offset 0.

See also: `mysql_fetch_row`.

mysql_fetch_object

Name

`mysql_fetch_object` — Fetch a result row as an object

Description

`object` **`mysql_fetch_object`** `(int` *`result`*`, int` *`[result_typ]`*`);`

Returns an object with properties that correspond to the fetched row, or false if there are no more rows.

`mysql_fetch_object` is similar to `mysql_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The optional argument *`result_typ`* is a constant and can take the following values: MYSQL_ASSOC, MYSQL_NUM, and MYSQL_BOTH.

Speed-wise, the function is identical to `mysql_fetch_array`, and almost as quick as `mysql_fetch_row` (the difference is insignificant).

Example 1. mysql fetch object

```
<?php
mysql_connect($host,$user,$password);
$result = mysql_db_query("database","select * from table");
while($row = mysql_fetch_object($result)) {
    echo $row->user_id;
    echo $row->fullname;
}
mysql_free_result($result);
?>
```

See also: `mysql_fetch_array` and `mysql_fetch_row`.

mysql_fetch_row

Name

`mysql_fetch_row` — Get a result row as an enumerated array

Description

```
array mysql_fetch_row(int result);
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

`mysql_fetch_row` fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to `mysql_fetch_row` would return the next row in the result set, or false if there are no more rows.

See also: `mysql_fetch_array`, `mysql_fetch_object`, `mysql_data_seek`, `mysql_fetch_lengths`, and `mysql_result`.

mysql_field_name

Name

`mysql_field_name` — Get the name of the specified field in a result

Description

```
string mysql_field_name(int result, int field_index);
```

`mysql_field_name` returns the name of the specified field. Arguments to the function is the result identifier and the field index, ie. `mysql_field_name($result,2);`

Will return the name of the second field in the result associated with the result identifier.

For downwards compatibility `mysql_fieldname` can also be used.

mysql_field_seek

Name

`mysql_field_seek` — Set result pointer to a specified field offset

Description

`int` **`mysql_field_seek`**`(int` *`result`*`, int` *`field_offset`*`);`

Seeks to the specified field offset. If the next call to `mysql_fetch_field` won't include a field offset, this field would be returned.

See also: `mysql_fetch_field`.

mysql_field_table

Name

`mysql_field_table` — Get name of the table the specified field is in

Description

`string` **`mysql_field_table`**`(int` *`result`*`, int` *`field_offset`*`);`

Get the table name for field. For downward compatibility `mysql_fieldtable` can also be used.

mysql_field_type

Name

`mysql_field_type` — Get the type of the specified field in a result

Description

`string **mysql_field_type**(int *result*, int *field_offset*);`

`mysql_field_type` is similar to the `mysql_field_name` function. The arguments are identical, but the field type is returned. This will be one of "int", "real", "string", "blob", or others as detailed in the MySQL documentation.

Example 1. mysql field types

```
<?php
mysql_connect("localhost:3306");
mysql_select_db("wisconsin");
$result = mysql_query("SELECT * FROM onek");
$fields = mysql_num_fields($result);
$rows   = mysql_num_rows($result);
$i = 0;
$table = mysql_field_table($result, $i);
echo "Your ’".$table."’ ta-
ble has ".$fields." fields and ".$rows." records <BR>";
echo "The table has the following fields <BR>";
while ($i < $fields) {
    $type  = mysql_field_type  ($result, $i);
    $name  = mysql_field_name  ($result, $i);
    $len   = mysql_field_len   ($result, $i);
    $flags = mysql_field_flags ($result, $i);
    echo $type." ".$name." ".$len." ".$flags."<BR>";
    $i++;
}
mysql_close();
?>
```

For downward compatibility `mysql_fieldtype` can also be used.

mysql_field_flags

Name

`mysql_field_flags` — Get the flags associated with the specified field in a result

Description

```
string mysql_field_flags(int result, int field_offset);
```

`mysql_field_flags` returns the field flags of the specified field. The flags are reported as a single word per flag separated by a single space, so that you can split the returned value using `explode`.

The following flags are reported, if your version of MySQL is current enough to support them: "not_null", "primary_key", "unique_key", "multiple_key", "blob", "unsigned", "zerofill", "binary", "enum", "auto_increment", "timestamp".

For downward compatibility `mysql_fieldflags` can also be used.

mysql_field_len

Name

`mysql_field_len` — Returns the length of the specified field

Description

```
int mysql_field_len(int result, int field_offset);
```

`mysql_field_len` returns the length of the specified field. For downward compatibility `mysql_fieldlen` can also be used.

mysql_free_result

Name

`mysql_free_result` — Free result memory

Description

`int` **`mysql_free_result`**`(int` *`result`*`);`

`mysql_free_result` only needs to be called if you are worried about using too much memory while your script is running. All associated result memory for the specified result identifier will automatically be freed.

For downward compatibility `mysql_freeresult` can also be used.

mysql_insert_id

Name

`mysql_insert_id` — Get the id generated from the previous INSERT operation

Description

`int` **`mysql_insert_id`**`(int` *`[link_identifier]`* `);`

`mysql_insert_id` returns the ID generated for an AUTO_INCREMENTED field. It will return the auto-generated ID returned by the last INSERT query performed using the given *`link_identifier`*. If *`link_identifier`* isn't specified, the last opened link is assumed.

mysql_list_fields

Name

`mysql_list_fields` — List MySQL result fields

Description

`int` **`mysql_list_fields`**`(string` *`database_name`*`, string` *`table_name`*`, int` *`[link_identifier]`* `);`

`mysql_list_fields` retrieves information about the given tablename. Arguments are the database name and the table name. A result pointer is returned which can be used with `mysql_field_flags`, `mysql_field_len`, `mysql_field_name`, and `mysql_field_type`.

A result identifier is a positive integer. The function returns -1 if a error occurs. A string describing the error will be placed in `$phperrmsg`, and unless the function was called as `@mysql()` then this error string will also be printed out.

For downward compatibility `mysql_listfields` can also be used.

mysql_list_dbs

Name

`mysql_list_dbs` — List databases available on on MySQL server

Description

`int` **`mysql_list_dbs`**`(int` *`[link_identifier]`* `);`

`mysql_list_dbs` will return a result pointer containing the databases available from the current mysql daemon. Use the `mysql_tablename` function to traverse this result pointer.

For downward compatibility `mysql_listdbs` can also be used.

mysql_list_tables

Name

`mysql_list_tables` — List tables in a MySQL database

Description

```
int mysql_list_tables(string database, int [link_identifier] );
```

`mysql_list_tables` takes a database name and returns a result pointer much like the `mysql_db_query` function. The `mysql_tablename` function should be used to extract the actual table names from the result pointer.

For downward compatibility `mysql_listtables` can also be used.

mysql_num_fields

Name

`mysql_num_fields` — Get number of fields in result

Description

```
int mysql_num_fields(int result);
```

`mysql_num_fields` returns the number of fields in a result set.

See also: `mysql_db_query`, `mysql_query`, `mysql_fetch_field`, `mysql_num_rows`.

For downward compatibility `mysql_numfields` can also be used.

mysql_num_rows

Name

`mysql_num_rows` — Get number of rows in result

Description

`int` **`mysql_num_rows`**`(int` *`result`*`);`

`mysql_num_rows` returns the number of rows in a result set.

See also: `mysql_db_query`, `mysql_query` and, `mysql_fetch_row`.

For downward compatibility `mysql_numrows` can also be used.

mysql_pconnect

Name

`mysql_pconnect` — Open a persistent connection to a MySQL Server

Description

`int` **`mysql_pconnect`**`(string` *`[hostname [:port] [:/path/to/socket] ]`* `, string` *`[username]`* `, string` *`[password]`* `);`

Returns: A positive MySQL persistent link identifier on success, or false on error `mysql_pconnect` establishes a connection to a MySQL server. All of the arguments are optional, and if they’re missing, defaults are assumed (’localhost’, user name of the user that owns the server process, empty password).

> The hostname string can also include a port number. eg. "hostname:port" or a path to a socket eg. ":/path/to/socket" for the localhost.
>
> **Note:** Support for ":port" wass added in 3.0B4.
>
> Support for the ":/path/to/socket" was added in 3.0.10.

`mysql_pconnect` acts very much like `mysql_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that’s already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`mysql_close` will not close links established by `mysql_pconnect`).

This type of links is therefore called ’persistent’.

mysql_query

Name

`mysql_query` — Send an SQL query to MySQL

Description

```
int mysql_query(string query, int [link_identifier] );
```

`mysql_query` sends a query to the currently active database on the server that’s associated with the specified link identifier. If *`link_identifier`* isn’t specified, the last opened link is assumed. If no link is open, the function tries to establish a link as if `mysql_connect` was called with no arguments, and use it.

The query string should not end with a semicolon.

`mysql_query` returns TRUE (non-zero) or FALSE to indicate whether or not the query succeeded. A return value of TRUE means that the query was legal and could be executed by the server. It does not indicate anything about the number of rows affected or returned. It is perfectly possible for a query to succeed but affect no rows or return no rows.

The following query is syntactically invalid, so `mysql_query` fails and returns FALSE:

Example 1. `mysql_query`

```
<?php
$result = mysql_query ("SELECT * WHERE 1=1")
    or die ("Invalid query");
?>
```

The following query is semantically invalid if `my_col` is not a column in the table `my_tbl`, so `mysql_query` fails and returns FALSE:

Example 2. `mysql_query`

```
<?php
$result = mysql_query ("SELECT my_col FROM my_tbl")
    or die ("Invalid query");
?>
```

`mysql_query` will also fail and return FALSE if you don’t have permission to access the table(s) referenced by the query.

Assuming the query succeeds, you can call `mysql_affected_rows` to find out how many rows were affected (for DELETE, INSERT, REPLACE, or UPDATE statements). For SELECT statements, `mysql_query` returns a new result identifier that you can pass to `mysql_result`. When you are done with the result set, you can free the resources associated with it by calling `mysql_free_result`.

See also: `mysql_affected_rows`, `mysql_db_query`, `mysql_free_result`, `mysql_result`, `mysql_select_db`, and `mysql_connect`.

mysql_result

Name

`mysql_result` — Get result data

Description

`int` **`mysql_result`**`(int` *`result`*`, int` *`row`*`, mixed` *`[field]`* `);`

`mysql_result` returns the contents of one cell from a MySQL result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than `mysql_result`. Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Calls `mysql_result` should not be mixed with calls to other functions that deal with the result set.

Recommended high-performance alternatives: `mysql_fetch_row`, `mysql_fetch_array`, and `mysql_fetch_object`.

mysql_select_db

Name

`mysql_select_db` — Select a MySQL database

Description

`int` **`mysql_select_db`**`(string` *`database_name`*`, int` *`[link_identifier]`* `);`

Returns: true on success, false on error

`mysql_select_db` sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if `mysql_connect` was called, and use it.

Every subsequent call to `mysql_query` will be made on the active database.

See also: `mysql_connect`, `mysql_pconnect`, and `mysql_query`

For downward compatibility `mysql_selectdb` can also be used.

mysql_tablename

Name

`mysql_tablename` — Get table name of field

Description

```
string mysql_tablename(int result, int i);
```

`mysql_tablename` takes a result pointer returned by the `mysql_list_tables` function as well as an integer index and returns the name of a table. The `mysql_num_rows` function may be used to determine the number of tables in the result pointer.

Example 1. `mysql_tablename` example

```
<?php
mysql_connect ("localhost:3306");
$result = mysql_list_tables ("wisconsin");
$i = 0;
while ($i < mysql_num_rows ($result)) {
    $tb_names[$i] = mysql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

X. Sybase functions

sybase_affected_rows

Name

`sybase_affected_rows` — get number of affected rows in last query

Description

`int` **`sybase_affected_rows`** `(int` *`[link_identifier]`* `);`

Returns: The number of affected rows by the last query.

`sybase_affected_rows` returns the number of rows affected by the last INSERT, UPDATE or DELETE query on the server associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

> This command is not effective for SELECT statements, only on statements which modify records. To retrieve the number of rows returned from a SELECT, use `sybase_num_rows`.
>
> **Note:** This function is only available using the CT library interface to Sybase, and not the DB library.

sybase_close

Name

`sybase_close` — close Sybase connection

Description

`int` **`sybase_close`** `(int` *`link_identifier`*`);`

Returns: true on success, false on error

sybase_close() closes the link to a Sybase database that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed.

Note that this isn't usually necessary, as non-persistent open links are automatically closed at the end of the script's execution.

sybase_close() will not close persistent links generated by sybase_pconnect().

See also: `sybase_connect`, `sybase_pconnect`.

sybase_connect

Name

`sybase_connect` — open Sybase server connection

Description

```
int sybase_connect(string servername, string username, string password);
```

Returns: A positive Sybase link identifier on success, or false on error.

sybase_connect() establishes a connection to a Sybase server. The servername argument has to be a valid servername that is defined in the 'interfaces' file.

In case a second call is made to sybase_connect() with the same arguments, no new link will be established, but instead, the link identifier of the already opened link will be returned.

The link to the server will be closed as soon as the execution of the script ends, unless it's closed earlier by explicitly calling `sybase_close`.

See also `sybase_pconnect`, `sybase_close`.

sybase_data_seek

Name

`sybase_data_seek` — move internal row pointer

Description

`int` **`sybase_data_seek`**`(int` *`result_identifier`*`, int` *`row_number`*`);`

Returns: true on success, false on failure

sybase_data_seek() moves the internal row pointer of the Sybase result associated with the specified result identifier to pointer to the specifyed row number. The next call to `sybase_fetch_row` would return that row.

See also: `sybase_data_seek`.

sybase_fetch_array

Name

`sybase_fetch_array` — fetch row as array

Description

`int` **`sybase_fetch_array`**`(int` *`result`*`);`

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

sybase_fetch_array() is an extended version of `sybase_fetch_row`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

An important thing to note is that using sybase_fetch_array() is NOT significantly slower than using sybase_fetch_row(), while it provides a significant added value.

For further details, also see `sybase_fetch_row`

sybase_fetch_field

Name

`sybase_fetch_field` — get field information

Description

```
object sybase_fetch_field(int result, int field_offset);
```

Returns an object containing field information.

sybase_fetch_field() can be used in order to obtain information about fields in a certain query result. If the field offset isn't specified, the next field that wasn't yet retreived by sybase_fetch_field() is retreived.

The properties of the object are:

- name - column name. if the column is a result of a function, this property is set to computed#N, where #N is a serial number.
- column_source - the table from which the column was taken
- max_length - maximum length of the column
- numeric - 1 if the column is numeric

See also `sybase_field_seek`

sybase_fetch_object

Name

`sybase_fetch_object` — fetch row as object

Description

`int` **`sybase_fetch_object`**`(int` *`result`*`);`

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

sybase_fetch_object() is similar to `sybase_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

Speed-wise, the function is identical to `sybase_fetch_array`, and almost as quick as `sybase_fetch_row` (the difference is insignificant).

See also: `sybase_fetch-array` and `sybase_fetch-row`.

sybase_fetch_row

Name

`sybase_fetch_row` — get row as enumerated array

Description

`array` **`sybase_fetch_row`**`(int` *`result`*`);`

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

sybase_fetch_row() fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to sybase_fetch_rows() would return the next row in the result set, or false if there are no more rows.

See also: `sybase_fetch_array`, `sybase_fetch_object`, `sybase_data_seek`, `sybase_fetch_lengths`, and `sybase_result`.

sybase_field_seek

Name

`sybase_field_seek` — set field offset

Description

`int` **`sybase_field_seek`**`(int` *`result`*`, int` *`field_offset`*`);`

Seeks to the specified field offset. If the next call to `sybase_fetch_field` won't include a field offset, this field would be returned.

See also: `sybase_fetch_field`.

sybase_free_result

Name

`sybase_free_result` — free result memory

Description

`int` **`sybase_free_result`**`(int` *`result`*`);`

`sybase_free_result` only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script, you may call `sybase_free_result` with the result identifier as an argument and the associated result memory will be freed.

sybase_num_fields

Name

`sybase_num_fields` — get number of fields in result

Description

`int` **`sybase_num_fields`** `(int` *`result`*`);`

sybase_num_fields() returns the number of fields in a result set.

See also: `sybase_db_query`, `sybase_query`, `sybase_fetch_field`, `sybase_num_rows`.

sybase_num_rows

Name

`sybase_num_rows` — get number of rows in result

Description

`int` **`sybase_num_rows`** `(string` *`result`*`);`

sybase_num_rows() returns the number of rows in a result set.

See also: `sybase_db_query`, `sybase_query` and, `sybase_fetch_row`.

sybase_pconnect

Name

`sybase_pconnect` — open persistent Sybase connection

Description

```
int sybase_pconnect(string servername, string username, string password);
```

Returns: A positive Sybase persistent link identifier on success, or false on error

sybase_pconnect() acts very much like `sybase_connect` with two major differences.

First, when connecting, the function would first try to find a (persistent) link that's already open with the same host, username and password. If one is found, an identifier for it will be returned instead of opening a new connection.

Second, the connection to the SQL server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (`sybase_close` will not close links established by `sybase_pconnect()`).

This type of links is therefore called ’persistent’.

sybase_query

Name

`sybase_query` — send Sybase query

Description

`int` **`sybase_query`**`(string` *`query`*`, int` *`link_identifier`*`);`

Returns: A positive Sybase result identifier on success, or false on error.

sybase_query() sends a query to the currently active database on the server that's associated with the specified link identifier. If the link identifier isn't specified, the last opened link is assumed. If no link is open, the function tries to establish a link as if `sybase_connect` was called, and use it.

See also: `sybase_db_query`, `sybase_select_db`, and `sybase_connect`.

sybase_result

Name

`sybase_result` — get result data

Description

`int` **`sybase_result`**`(int` *`result`*`, int` *`i`*`, mixed` *`field`*`);`

Returns: The contents of the cell at the row and offset in the specified Sybase result set.

sybase_result() returns the contents of one cell from a Sybase result set. The field argument can be the field's offset, or the field's name, or the field's table dot field's name (fieldname.tablename). If the column name has been aliased ('select foo as bar from...'), use the alias instead of the column name.

When working on large result sets, you should consider using one of the functions that fetch an entire row (specified below). As these functions return the contents of multiple cells in one function call, they're MUCH quicker than sybase_result(). Also, note that specifying a numeric offset for the field argument is much quicker than specifying a fieldname or tablename.fieldname argument.

Recommended high-performance alternatives: `sybase_fetch_row`, `sybase_fetch_array`, and `sybase_fetch_object`.

sybase_select_db

Name

`sybase_select_db` — select Sybase database

Description

`int` **`sybase_select_db`**`(string` *`database_name`*`, int` *`link_identifier`*`);`

Returns: true on success, false on error

sybase_select_db() sets the current active database on the server that's associated with the specified link identifier. If no link identifier is specified, the last opened link is assumed. If no link is open, the function will try to establish a link as if `sybase_connect` was called, and use it.

Every subsequent call to `sybase_query` will be made on the active database.

See also: `sybase_connect`, `sybase_pconnect`, and `sybase_query`

XI. Network functions

checkdnsrr

Name

`checkdnsrr` — Check DNS records corresponding to a given Internet host name or IP address.

Description

`int **checkdnsrr**(string *host*, string *[type]*);`

Searches DNS for records of type *`type`* corresponding to *`host`*. Returns true if any records are found; returns false if no records were found or if an error occurred.

`type` may be any one of: A, MX, NS, SOA, PTR, CNAME, or ANY. The default is MX.

`host` may either be the IP address in dotted-quad notation or the host name.

See also `getmxrr`, `gethostbyaddr`, `gethostbyname`, `gethostbynamel`, and the named(8) manual page.

closelog

Name

`closelog` — close connection to system logger

Description

`int **closelog**(void);`

`closelog` closes the descriptor being used to write to the system logger. The use of `closelog` is optional.

debugger_off

Name

`debugger_off` — disable internal PHP debugger

Description

```
int debugger_off(void);
```

Disables the internal PHP debugger. The debugger is still under development.

debugger_on

Name

`debugger_on` — enable internal PHP debugger

Description

```
int debugger_on(string address);
```

Enables the internal PHP debugger, connecting it to *`address`*. The debugger is still under development.

fsockopen

Name

`fsockopen` — Open Internet or Unix domain socket connection.

Description

```
int fsockopen(string hostname, int port, int [errno], string [errstr], double
[timeout]);
```

Initiates a stream connection in the Internet (AF_INET) or Unix (AF_UNIX) domain. For the Internet domain, it will open a TCP socket connection to *`hostname`* on port *`port`*. For the Unix domain, *`hostname`* will be used as the path to the socket, *`port`* must be set to 0 in this case. The optional *`timeout`* can be used to set a timeout in seconds for the connect system call.

`fsockopen` returns a file pointer which may be used together with the other file functions (such as `fgets`, `fgetss`, `fputs`, `fclose`, `feof`).

If the call fails, it will return false and if the optional *`errno`* and *`errstr`* arguments are present they will be set to indicate the actual system level error that occurred on the system-level connect() call. If the returned errno is 0 and the function returned false, it is an indication that the error occurred before the connect() call. This is most likely due to a problem initializing the socket. Note that the errno and errstr arguments must be passed by reference.

Depending on the environment, the Unix domain or the optional connect timeout may not be available.

The socket will by default be opened in blocking mode. You can switch it to non-blocking mode by using `set_socket_blocking`.

Example 1. fsockopen example

```
$fp = fsockopen("www.php.net", 80, &$errno, &$errstr, 30);
if(!$fp) {
echo "$errstr ($errno)<br>\n";
} else {
fputs($fp,"GET / HTTP/1.0\n\n");
while(!feof($fp)) {
echo fgets($fp,128);
}
fclose($fp);
}
```

See also: `pfsockopen`

gethostbyaddr

Name

`gethostbyaddr` — Get the Internet host name corresponding to a given IP address.

Description

```
string gethostbyaddr(string ip_address);
```

Returns the host name of the Internet host specified by *`ip_address`*. If an error occurs, returns *`ip_address`*.

See also `gethostbyname`.

gethostbyname

Name

`gethostbyname` — Get the IP address corresponding to a given Internet host name.

Description

```
string gethostbyname(string hostname);
```

Returns the IP address of the Internet host specified by *`hostname`*.

See also `gethostbyaddr`.

gethostbynamel

Name

`gethostbynamel` — Get a list of IP addresses corresponding to a given Internet host name.

Description

```
array gethostbynamel(string hostname);
```

Returns a list of IP addresses to which the Internet host specified by *`hostname`* resolves.

See also `gethostbyname`, `gethostbyaddr`, `checkdnsrr`, `getmxrr`, and the named(8) manual page.

getmxrr

Name

`getmxrr` — Get MX records corresponding to a given Internet host name.

Description

```
int getmxrr(string hostname, array mxhosts, array [weight]);
```

Searches DNS for MX records corresponding to *`hostname`*. Returns true if any records are found; returns false if no records were found or if an error occurred.

A list of the MX records found is placed into the array *`mxhosts`*. If the *`weight`* array is given, it will be filled with the weight information gathered.

See also `checkdnsrr`, `gethostbyname`, `gethostbynamel`, `gethostbyaddr`, and the named(8) manual page.

openlog

Name

`openlog` — open connection to system logger

Description

```
int openlog(string ident, int option, int facility);
```

`openlog` opens a connection to the system logger for a program. The string *`ident`* is added to each message. Values for *`option`* and *`facility`* are given in the next section. The use of openlog() is optional; It will automatically be called by `syslog` if necessary, in which case ident will default to `false`. See also `syslog` and `closelog`.

pfsockopen

Name

`pfsockopen` — Open persistent Internet or Unix domain socket connection.

Description

```
int pfsockopen(string hostname, int port, int [errno], string [errstr], int
[timeout]);
```

This function behaves exactly as `fsockopen` with the difference that the connection is not closed after the script finishes. It is the persistent version of `fsockopen`.

set_socket_blocking

Name

`set_socket_blocking` — Set blocking/non-blocking mode on a socket

Description

`int` **`set_socket_blocking`**`(int` *`socket descriptor`*`, int` *`mode`*`);`

If *`mode`* is false, the given socket descriptor will be switched to non-blocking mode, and if true, it will be switched to blocking mode. This affects calls like `fgets` that read from the socket. In non-blocking mode an fgets() call will always return right away while in blocking mode it will wait for data to become available on the socket.

syslog

Name

`syslog` — generate a system log message

Description

`int` **`syslog`**`(int` *`priority`*`, string` *`message`*`);`

`syslog` generates a log message that will be distributed by the system logger. *`priority`* is a combination of the facility and the level, values for which are given in the next section. The remaining argument is the message to send, except that the two characters `%m` will be replaced by the error message string (strerror) corresponding to the present value of errno.

More information on the syslog facilities can be found in the man pages for syslog on Unix machines.

On Windows NT, the syslog service is emulated using the Event Log.

XII. NIS functions

NIS (formerly called Yellow Pages) allows network management of important administrative files (e.g. the password file). For more information refer to the NIS manpage and Introduction to YP/NIS (http://www.desy.de/~sieversm/ypdoku/ypdoku/ypdoku.html). There is also a book called Managing NFS and NIS (http://www.oreilly.com/catalog/nfs/noframes.html) by Hal Stern.

To get these functions to work, you have to configure PHP with `-with-yp`.

yp_get_default_domain

Name

`yp_get_default_domain` — Fetches the machine’s default NIS domain.

Description

int **yp_get_default_domain** (void);

`yp_get_default_domain` returns the default domain of the node or FALSE. Can be used as the domain parameter for successive NIS calls.

A NIS domain can be described a group of NIS maps. Every host that needs to look up information binds itself to a certain domain. Refer to the documents mentioned at the beginning for more detailed information.

Example 1. Example for the default domain

```
<?php
    $domain = yp_get_default_domain();

    if(!$domain) {
        echo yp_errno() . ": " . yp_err_string();
    }

    echo "Default NIS domain is: " . $domain;
?>
```

See also: yp_errno and yp_err_string

yp_order

Name

`yp_order` — Returns the order number for a map.

Description

`int` **`yp_order`**`(string` *`domain`*`, string` *`map`*`);`

`yp_order` returns the order number for a map or FALSE.

Example 1. Example for the NIS order

```
<?php
    $number = yp_order($domain,$mapname);

    if(!$number) {
        echo yp_errno() . ": " . yp_err_string();
    }

    echo "Order number for this map is: " . $order;
?>
```

See also: yp_get_default_domain yp_errno and yp_err_string

yp_master

Name

`yp_master` — Returns the machine name of the master NIS server for a map.

Description

`string` **`yp_master`**`(string` *`domain`*`, string` *`map`*`);`

`yp_master` returns the machine name of the master NIS server for a map.

Example 1. Example for the NIS master

```
<?php
    $number = yp_master($domain, $mapname);

    if(!$number) {
        echo yp_errno() . ": " . yp_err_string();
    }

    echo "Master for this map is: " . $master;
?>
```

See also: yp_get_default_domain yp_errno and yp_err_string

yp_match

Name

`yp_match` — Returns the matched line.

Description

`string` **`yp_match`**`(string` *`domain`*`, string` *`map`*`, string` *`key`*`);`

`yp_match` returns the value associated with the passed key out of the specified map or FALSE. This key must be exact.

Example 1. Example for NIS match

```
<?php
    $entry = yp_match($domain, "passwd.byname", "joe");

    if(!$entry) {
        echo yp_errno() . ": " . yp_err_string();
    }

    echo "Matched entry is: " . $entry;
?>
```

In this case this could be: joe:##joe:11111:100:Joe User:/home/j/joe:/usr/local/bin/bash

See also: yp_get_default_domain yp_errno and yp_err_string

yp_first

Name

`yp_first` — Returns the first key-value pair from the named map.

Description

`string[] yp_first(string domain, string map);`

`yp_first` returns the first key-value pair from the named map in the named domain, otherwise FALSE.

Example 1. Example for the NIS first

```
<?php
    $entry = yp_first($domain, "passwd.byname");

    if(!$entry) {
        echo yp_errno() . ": " . yp_err_string();
    }
    $key = key($entry);
    echo "First entry in this map has key " . $key
          . " and value " . $entry[$key];
?>
```

See also: yp_get_default_domain yp_errno and yp_err_string

yp_next

Name

`yp_next` — Returns the next key-value pair in the named map.

Description

`string[] **yp_next**(string *domain*, string *map*, string *key*);`

`yp_next` returns the next key-value pair in the named map after the specified key or FALSE.

Example 1. Example for NIS next

```
<?php
    $entry = yp_next($domain, "passwd.byname", "joe");

    if(!$entry) {
        echo yp_errno() . ": " . yp_err_string();
    }

    $key = key($entry);

    echo "The next entry after joe has key " . $key
          . " and value " . $entry[$key];
?>
```

See also: yp_get_default_domain, yp_errno and yp_err_string

yp_errno

Name

`yp_errno` — Returns the error code of the previous operation.

Description

```
int yp_errno();
```

`yp_errno` returns the error code of the previous operation.

Possible errors are:

1 args to function are bad
2 RPC failure - domain has been unbound
3 can’t bind to server on this domain
4 no such map in server’s domain
5 no such key in map
6 internal yp server or client error
7 resource allocation failure
8 no more records in map database
9 can’t communicate with portmapper
10 can’t communicate with ypbind
11 can’t communicate with ypserv
12 local domain name not set
13 yp database is bad
14 yp version mismatch
15 access violation
16 database busy

See also: yp_err_string

yp_err_string

Name

`yp_err_string` — Returns the error string associated with the previous operation.

Description

string **`yp_err_string`**(void);

`yp_err_string` returns the error message associated with the previous operation. Useful to indicate what exactly went wrong.

Example 1. Example for NIS errors

```
<?php
    echo "Error: " . yp_err_string();
?>
```

See also: yp_errno

XIII. ODBC functions

odbc_autocommit

Name

`odbc_autocommit` — Toggle autocommit behaviour

Description

`int` **`odbc_autocommit`**`(int` *`connection_id`*`, int` *`[OnOff]`*`);`

Without the *`OnOff`* parameter, this function returns auto-commit status for *`connection_id`*. True is returned if auto-commit is on, false if it is off or an error occurs.

If *`OnOff`* is true, auto-commit is enabled, if it is false auto-commit is disabled. Returns `true` on success, `false` on failure.

By default, auto-commit is on for a connection. Disabling auto-commit is equivalent with starting a transaction.

See also `odbc_commit` and `odbc_rollback`.

odbc_binmode

Name

`odbc_binmode` — handling of binary column data

Description

`int` **`odbc_binmode`**`(int` *`result_id`*`, int` *`mode`*`);`

(ODBC SQL types affected: BINARY, VARBINARY, LONGVARBINARY)

- ODBC_BINMODE_PASSTHRU: Passthru BINARY data

- ODBC_BINMODE_RETURN: Return as is
- ODBC_BINMODE_CONVERT: Convert to char and return

When binary SQL data is converted to character C data, each byte (8 bits) of source data is represented as two ASCII characters. These characters are the ASCII character representation of the number in its hexadecimal form. For example, a binary 00000001 is converted to `"01"` and a binary 11111111 is converted to `"FF"`.

Table 1. LONGVARBINARY handling

binmode	longreadlen	result
ODBC_BINMODE_PASSTHRU	0	passthru
ODBC_BINMODE_RETURN	0	passthru
ODBC_BINMODE_CONVERT	0	passthru
ODBC_BINMODE_PASSTHRU	0	passthru
ODBC_BINMODE_PASSTHRU	>0	passthru
ODBC_BINMODE_RETURN	>0	return as is
ODBC_BINMODE_CONVERT	>0	return as char

If `odbc_fetch_into` is used, passthru means that an empty string is returned for these columns.

If *`result_id`* is `0`, the settings apply as default for new results.

Note: Default for longreadlen is `4096` and binmode defaults to `ODBC_BINMODE_RETURN`. Handling of binary long columns is also affected by `odbc_longreadlen`

odbc_close

Name

`odbc_close` — Close an ODBC connection

Description

`void **odbc_close**(int *connection_id*);`

`odbc_close` will close down the connection to the database server associated with the given connection identifier.

Note: This function will fail if there are open transactions on this connection. The connection will remain open in this case.

odbc_close_all

Name

`odbc_close_all` — Close all ODBC connections

Description

`void **odbc_close_all**(void);`

`odbc_close_all` will close down all connections to database server(s).

Note: This function will fail if there are open transactions on a connection. This connection will remain open in this case.

odbc_commit

Name

`odbc_commit` — Commit an ODBC transaction

Description

`int` **`odbc_commit`**`(int` *`connection_id`*`);`

Returns: `true` on success, `false` on failure. All pending transactions on *`connection_id`* are committed.

odbc_connect

Name

`odbc_connect` — Connect to a datasource

Description

`int` **`odbc_connect`**`(string` *`dsn`*`, string` *`user`*`, string` *`password`*`, int` *`[cursor_type]`*`);`

Returns an ODBC connection id or 0 (`false`) on error.

The connection id returned by this functions is needed by other ODBC functions. You can have multiple connections open at once. The optional fourth parameter sets the type of cursor to be used for this connection. This parameter is not normally needed, but can be useful for working around problems with some ODBC drivers.

With some ODBC drivers, executing a complex stored procedure may fail with an error similar to: "Cannot open a cursor on a stored procedure that has anything other than a single select statement in it".

Using SQL_CUR_USE_ODBC may avoid that error. Also, some drivers don't support the optional row_number parameter in `odbc_fetch_row`. SQL_CUR_USE_ODBC might help in that case, too.

The following constants are defined for cursortype:

- SQL_CUR_USE_IF_NEEDED
- SQL_CUR_USE_ODBC
- SQL_CUR_USE_DRIVER
- SQL_CUR_DEFAULT

For persistent connections see `odbc_pconnect`.

odbc_cursor

Name

`odbc_cursor` — Get cursorname

Description

```
string odbc_cursor(int result_id);
```

odbc_cursor will return a cursorname for the given result_id.

odbc_do

Name

`odbc_do` — synonym for `odbc_exec`

Description

string **`odbc_do`**(int *`conn_id`*, string *`query`*);

odbc_do will execute a query on the given connection

odbc_exec

Name

`odbc_exec` — Prepare and execute a SQL statement

Description

int **`odbc_exec`**(int *`connection_id`*, string *`query_string`*);

Returns `false` on error. Returns an ODBC result identifier if the SQL command was executed successfully.

`odbc_exec` will send an SQL statement to the database server specified by *`connection_id`*. This parameter must be a valid identifier returned by `odbc_connect` or `odbc_pconnect`.

See also: `odbc_prepare` and `odbc_execute` for multiple execution of SQL statements.

odbc_execute

Name

`odbc_execute` — execute a prepared statement

Description

`int` **`odbc_execute`**`(int` *`result_id`*`, array` *`[parameters_array]`*`);`

Executes a statement prepared with `odbc_prepare`. Returns `true` on successful execution, `false` otherwise. The array *`arameters_array`* only needs to be given if you really have parameters in your statement.

odbc_fetch_into

Name

`odbc_fetch_into` — Fetch one result row into array

Description

`int` **`odbc_fetch_into`**`(int` *`result_id`*`, int` *`[rownumber]`*`, array` *`result_array`*`);`

Returns the number of columns in the result; `false` on error. *`result_array`* must be passed by reference, but it can be of any type since it will be converted to type array. The array will contain the column values starting at array index 0.

odbc_fetch_row

Name

`odbc_fetch_row` — Fetch a row

Description

`int` **`odbc_fetch_row`**`(int` *`result_id`*`, int` *`[row_number]`*`);`

If `odbc_fetch_row` was succesful (there was a row), `true` is returned. If there are no more rows, `false` is returned.

`odbc_fetch_row` fetches a row of the data that was returned by `odbc_do` / `odbc_exec`. After `odbc_fetch_row` is called, the fields of that row can be accessed with `odbc_result`.

If *`row_number`* is not specified, `odbc_fetch_row` will try to fetch the next row in the result set. Calls to `odbc_fetch_row` with and without *`row_number`* can be mixed.

To step through the result more than once, you can call `odbc_fetch_row` with *`row_number`* 1, and then continue doing `odbc_fetch_row` without *`row_number`* to review the result. If a driver doesn't support fetching rows by number, the *`row_number`* parameter is ignored.

odbc_field_name

Name

`odbc_field_name` — Get the columnname

Description

`string` **`odbc_fieldname`**`(int` *`result_id`*`, int` *`field_number`*`);`

`odbc_field_name` will return the name of the field occupying the given column number in the given ODBC result identifier. Field numbering starts at 1. `false` is returned on error.

odbc_field_type

Name

`odbc_field_type` — datatype of a field

Description

```
string odbc_field_type(int result_id, int field_number);
```

`odbc_field_type` will return the SQL type of the field referecend by number in the given ODBC result identifier. Field numbering starts at 1.

odbc_field_len

Name

`odbc_field_len` — get the Length of a field

Description

```
int odbc_field_len(int result_id, int field_number);
```

`odbc_field_len` will return the length of the field referecend by number in the given ODBC result identifier. Field numbering starts at 1.

odbc_free_result

Name

`odbc_free_result` — free resources associated with a result

Description

```
int odbc_free_result(int result_id);
```

Always returns `true`.

`odbc_free_result` only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call `odbc_free_result`, and the memory associated with *`result_id`* will be freed.

> **Note:** If auto-commit is disabled (see `odbc_autocommit`) and you call `odbc_free_result` before commiting, all pending transactions are rolled back.

odbc_longreadlen

Name

`odbc_longreadlen` — handling of LONG columns

Description

```
int odbc_longreadlen(int result_id, int length);
```

(ODBC SQL types affected: LONG, LONGVARBINARY) The number of bytes returned to PHP is controled by the parameter length. If it is set to 0, Long column data is passed thru to the client.

> **Note:** Handling of LONGVARBINARY columns is also affected by `odbc_binmode`

odbc_num_fields

Name

`odbc_num_fields` — number of columns in a result

Description

`int` **`odbc_num_fields`**`(int` *`result_id`*`);`

`odbc_num_fields` will return the number of fields (columns) in an ODBC result. This function will return -1 on error. The argument is a valid result identifier returned by `odbc_exec`.

odbc_pconnect

Name

`odbc_pconnect` — Open a persistent database connection

Description

`int` **`odbc_pconnect`**`(string` *`dsn`*`, string` *`user`*`, string` *`password`*`, int` *`[cursor_type]`*`);`

Returns an ODBC connection id or 0 (`false`) on error. This function is much like `odbc_connect`, except that the connection is not really closed when the script has finished. Future requests for a connection with the same *`dsn`*, *`user`*, *`password`* combination (via `odbc_connect` and `odbc_pconnect`) can reuse the persistent connection.

> **Note:** Persistent connections have no effect if PHP is used as a CGI program.

For information about the optional cursor_type parameter see the `odbc_connect` function. For more information on persistent connections, refer to the PHP FAQ.

odbc_prepare

Name

`odbc_prepare` — Prepares a statement for execution

Description

`int` **`odbc_prepare`**`(int` *`connection_id`*`, string` *`query_string`*`);`

Returns `false` on error.

Returns an ODBC result identifier if the SQL command was prepared successfully. The result identifier can be used later to execute the statement with `odbc_execute`.

odbc_num_rows

Name

`odbc_num_rows` — Number of rows in a result

Description

`int` **`odbc_num_rows`**`(int` *`result_id`*`);`

`odbc_num_rows` will return the number of rows in an ODBC result. This function will return -1 on error. For INSERT, UPDATE and DELETE statements `odbc_num_rows` returns the number of rows affected. For a SELECT clause this `can` be the number of rows available.

Note: Using `odbc_num_rows` to determine the number of rows available after a SELECT will return -1 with many drivers.

odbc_result

Name

`odbc_result` — get result data

Description

`string` **`odbc_result`**`(int` *`result_id`*`, mixed` *`field`*`);`

Returns the contents of the field.

`field` can either be an integer containing the column number of the field you want; or it can be a string containing the name of the field. For example:

```
$item_3 = odbc_result($Query_ID, 3 );
$item_val = odbc_result($Query_ID, "val");
```

The first call to `odbc_result` returns the value of the third field in the current record of the query result. The second function call to `odbc_result` returns the value of the field whose field name is "val" in the current record of the query result. An error occurs if a column number parameter for a field is less than one or exceeds the number of columns (or fields) in the current record. Similarly, an error occurs if a field with a name that is not one of the fieldnames of the table(s) that is(are) being queried.

Field indices start from 1. Regarding the way binary or long column data is returned refer to `odbc_binmode` and `odbc_longreadlen`.

odbc_result_all

Name

`odbc_result_all` — Print result as HTML table

Description

`int` **`odbc_result_all`**`(int` *`result_id`*`, string` *`[format]`*`);`

Returns the number of rows in the result or `false` on error.

`odbc_result_all` will print all rows from a result identifier produced by `odbc_exec`. The result is printed in HTML table format. With the optional string argument *`format`*, additional overall table formatting can be done.

odbc_rollback

Name

`odbc_rollback` — Rollback a transaction

Description

`int` **`odbc_rollback`**`(int` *`connection_id`*`);`

Rolls back all pending statements on *`connection_id`*. Returns `true` on success, `false` on failure.

odbc_setoption

Name

`odbc_setoption` — Adjust ODBC settings. Returns false if an error occurs, otherwise true.

Description

```
int odbc_setoption(int id, int function, int option, int param);
```

This function allows fiddling with the ODBC options for a particular connection or query result. It was written to help find work arounds to problems in quirky ODBC drivers. You should probably only use this function if you are an ODBC programmer and understand the effects the various options will have. You will certainly need a good ODBC reference to explain all the different options and values that can be used. Different driver versions support different options.

Because the effects may vary depending on the ODBC driver, use of this function in scripts to be made publicly available is strongly discouraged. Also, some ODBC options are not available to this function because they must be set before the connection is established or the query is prepared. However, if on a particular job it can make PHP work so your boss doesn't tell you to use a commercial product, that's all that really matters.

`Id` is a connection id or result id on which to change the settings.For SQLSetConnectOption(), this is a connection id. For SQLSetStmtOption(), this is a result id.

`function` is the ODBC function to use. The value should be 1 for SQLSetConnectOption() and 2 for SQLSetStmtOption().

Parmeter *`option`* is the option to set.

Parameter *`param`* is the value for the given *`option`*.

Example 1. ODBC Setoption Examples

```
// 1. Option 102 of SQLSetConnectOption() is SQL_AUTOCOMMIT.
//    Value 1 of SQL_AUTOCOMMIT is SQL_AUTOCOMMIT_ON.
//    This example has the same effect as
//    odbc_autocommit($conn, true);

odbc_setoption ($conn, 1, 102, 1);
```

```
// 2. Option 0 of SQLSetStmtOption() is SQL_QUERY_TIMEOUT.
//    This example sets the query to timeout after 30 seconds.

$result = odbc_prepare ($conn, $sql);
odbc_setoption ($result, 2, 0, 30);
odbc_execute ($result);
```

XIV. Oracle 8 functions

These functions allow you to access Oracle8 and Oracle7 databases. It uses the Oracle8 Call-Interface (OCI8). You will need the Oracle8 client libraries to use this extension.

This extension is more flexible than the standard Oracle extension. It supports binding of global and local PHP variables to Oracle placeholders, has full LOB, FILE and ROWID support and allows you to use user-supplied define variables.

OCIDefineByName

Name

`OCIDefineByName` — Use a PHP variable for the define-step during a SELECT

Description

`int` **`OCIDefineByName`** `(int` *`stmt`*`, string` *`Column-Name`*`, mixed &`*`variable`*`, int` *`[type]`*`);`

`OCIDefineByName` uses fetches SQL-Columns into user-defined PHP-Variables. Be careful that Oracle user ALL-UPPERCASE column-names, whereby in your select you can also write lower-case. `OCIDefineByName` expects the *`Column-Name`* to be in uppercase. If you define a variable that doesn't exists in you select statement, no error will be given!

If you need to define an abstract Datatype (LOB/ROWID/BFILE) you need to allocate it first using `OCINewDescriptor` function. See also the `OCIBindByName` function.

Example 1. OCIDefineByName

```
<?php
/* OCIDefineByPos example thies@digicol.de (980219) */

$conn = OCILogon("scott","tiger");

$stmt = OCIParse($conn,"select empno, ename from emp");

/* the define MUST be done BEFORE ociexecute! */

OCIDefineByName($stmt,"EMPNO",&$empno);
OCIDefineByName($stmt,"ENAME",&$ename);

OCIExecute($stmt);

while (OCIFetch($stmt)) {
    echo "empno:".$empno."\n";
    echo "ename:".$ename."\n";
}
```

```
OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIBindByName

Name

`OCIBindByName` — Bind a PHP variable to an Oracle Placeholder

Description

`int` **`OCIBindByName`**`(int` *`stmt`*`, string` *`ph_name`*`, mixed &`*`variable`*`, int`*`length`*`, int` *`[type]`*`);`

`OCIBindByName` binds the PHP variable *`variable`* to the Oracle placeholder *`ph_name`*. Whether it will be used for input or output will be determined run-time, and the necessary storage space will be allocated. The *`length`* paramter sets the maximum length for the bind. If you set *`length`* to -1 `OCIBindByName` will use the current length of *`variable`* to set the maximum length.

If you need to bind an abstract Datatype (LOB/ROWID/BFILE) you need to allocate it first using `OCINewDescriptor` function. The *`length`* is not used for abstract Datatypes and should be set to -1. The *`type`* variable tells oracle, what kind of descriptor we want to use. Possible values are: OCI_B_FILE (Binary-File), OCI_B_CFILE (Character-File), OCI_B_CLOB (Character-LOB), OCI_B_BLOB (Binary-LOB) and OCI_B_ROWID (ROWID).

Example 1. OCIDefineByName

```
<?php
/* OCIBindByPos example thies@digicol.de (980221)

  inserts 3 resords into emp, and uses the ROWID for updating the
  records just after the insert.
*/

$conn = OCILogon("scott","tiger");
```

```
$stmt = OCIParse($conn,"insert into emp (empno, ename) ".
   "values (:empno,:ename) ".
   "returning ROWID into :rid");

$data = array(1111 => "Larry", 2222 => "Bill", 3333 => "Jim");

$rowid = OCINewDescriptor($conn,OCI_D_ROWID);

OCIBindByName($stmt,":empno",&$empno,32);
OCIBindByName($stmt,":ename",&$ename,32);
OCIBindByName($stmt,":rid",&$rowid,-1,OCI_B_ROWID);

$update = OCIParse($conn,"update emp set sal = :sal where ROWID = :rid");
OCIBindByName($update,":rid",&$rowid,-1,OCI_B_ROWID);
OCIBindByName($update,":sal",&$sal,32);

$sal = 10000;

while (list($empno,$ename) = each($data)) {
OCIExecute($stmt);
OCIExecute($update);
}

$rowid->free();

OCIFreeStatement($update);
OCIFreeStatement($stmt);

$stmt = OCIParse($conn,"select * from emp where empno in (1111,2222,3333)");
OCIExecute($stmt);
while (OCIFetchInto($stmt,&$arr,OCI_ASSOC)) {
var_dump($arr);
}
OCIFreeStatement($stmt);

/* delete our "junk" from the emp table.... */
$stmt = OCIParse($conn,"delete from emp where empno in (1111,2222,3333)");
OCIExecute($stmt);
OCIFreeStatement($stmt);

OCILogoff($conn);
?>
```

OCILogon

Name

`OCILogon` — Establishes a connection to Oracle

Description

`int OCILogon(string username, string password, string [db]);`

`OCILogon` returns an connection identifier needed for most other OCI calls. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

Connections are shared at the page level when using `OCILogon`. This means that commits and rollbacks apply to all open transactions in the page, even if you have created multiple connections.

This example demonstrates how the connections are shared.

Example 1. OCILogon

```
<?php
print "<HTML><PRE>";
$db = "";

$c1 = ocilogon("scott","tiger",$db);
$c2 = ocilogon("scott","tiger",$db);

function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test
varchar2(64))");
  ociexecute($stmt);
  echo $conn." created table\n\n";
}

function drop_table($conn)
{ $stmt = ociparse($conn,"drop table scott.hallo");
  ociexecute($stmt);
  echo $conn." dropped table\n\n";
}

function insert_data($conn)
{ $stmt = ociparse($conn,"insert into scott.hallo val-
ues('$conn' || ' ' || to_char(sysdate,'DD-MON-YY HH24:MI:SS'))");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." inserted hallo\n\n";
}

function delete_data($conn)
{ $stmt = ociparse($conn,"delete from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." deleted hallo\n\n";
}

function commit($conn)
{ ocicommit($conn);
  echo $conn." commited\n\n";
}

function rollback($conn)
{ ocirollback($conn);
  echo $conn." rollback\n\n";
}

function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."---selecting\n\n";
  while (ocifetch($stmt))
    echo $conn." <".ociresult($stmt,"TEST").">\n\n";
  echo $conn."---done\n\n";
}

create_table($c1);
insert_data($c1);   // Insert a row using c1
insert_data($c2);   // Insert a row using c2

select_data($c1);   // Results of both inserts are returned
select_data($c2);

rollback($c1);      // Rollback using c1

select_data($c1);   // Both inserts have been rolled back
select_data($c2);
```

```
insert_data($c2);   // Insert a row using c2
commit($c2);        // commit using c2

select_data($c1);   // result of c2 insert is returned

delete_data($c1);   // delete all rows in table using c1
select_data($c1);   // no rows returned
select_data($c2);   // no rows returned
commit($c1);        // commit using c1

select_data($c1);   // no rows returned
select_data($c2);   // no rows returned


drop_table($c1);
print "</PRE></HTML>";
?>
```

See also `OCIPLogon` and `OCINLogon`.

OCIPLogon

Name

`OCIPLogon` — Connect to an Oracle database and log on using a persistant connection. Returns a new session.

Description

int **OCIPLogon**(string *username*, string *password*, string *[db]*);

`OCIPLogon` creates a persistent connection to an Oracle 8 database and logs on. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

See also `OCILogon` and `OCINLogon`.

OCINLogon

Name

`OCINLogon` — Connect to an Oracle database and log on using a new connection. Returns a new session.

Description

```
int OCINLogon(string username, string password, string [db]);
```

`OCINLogon` creates a new connection to an Oracle 8 database and logs on. The optional third parameter can either contain the name of the local Oracle instance or the name of the entry in tnsnames.ora to which you want to connect. If the optional third parameter is not specified, PHP uses the environment variables ORACLE_SID (Oracle instance) or TWO_TASK (tnsnames.ora) to determine which database to connect to.

`OCINLogon` forces a new connection. This should be used if you need to isolate a set of transactions. By default, connections are shared at the page level if using `OCILogon` or at the web server process level if using `OCIPLogon`. If you have multiple connections open using `OCINLogon`, all commits and rollbacks apply to the specified connection only.

This example demonstrates how the connections are separated.

Example 1. OCINLogon

```
<?php
print "<HTML><PRE>";
$db = "";

$c1 = ocilogon("scott","tiger",$db);
$c2 = ocinlogon("scott","tiger",$db);

function create_table($conn)
{ $stmt = ociparse($conn,"create table scott.hallo (test
varchar2(64))");
  ociexecute($stmt);
  echo $conn." created table\n\n";
}

function drop_table($conn)
{ $stmt = ociparse($conn,"drop table scott.hallo");
  ociexecute($stmt);
  echo $conn." dropped table\n\n";
}

function insert_data($conn)
{ $stmt = ociparse($conn,"insert into scott.hallo val-
ues('$conn' || ' ' || to_char(sysdate,'DD-MON-YY HH24:MI:SS'))");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." inserted hallo\n\n";
}

function delete_data($conn)
{ $stmt = ociparse($conn,"delete from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn." deleted hallo\n\n";
}

function commit($conn)
{ ocicommit($conn);
  echo $conn." commited\n\n";
}

function rollback($conn)
{ ocirollback($conn);
  echo $conn." rollback\n\n";
}

function select_data($conn)
{ $stmt = ociparse($conn,"select * from scott.hallo");
  ociexecute($stmt,OCI_DEFAULT);
  echo $conn."---selecting\n\n";
  while (ocifetch($stmt))
    echo $conn." <".ociresult($stmt,"TEST").">\n\n";
  echo $conn."---done\n\n";
}

create_table($c1);
insert_data($c1);
```

```
select_data($c1);
select_data($c2);

rollback($c1);

select_data($c1);
select_data($c2);

insert_data($c2);
commit($c2);

select_data($c1);

delete_data($c1);
select_data($c1);
select_data($c2);
commit($c1);

select_data($c1);
select_data($c2);


drop_table($c1);
print "</PRE></HTML>";
?>
```

See also `OCILogon` and `OCIPLogon`.

OCILogOff

Name

`OCILogOff` — Disconnects from Oracle

Description

```
int OCILogOff(int connection);
```

`OCILogOff` closes an Oracle connection.

OCIExecute

Name

`OCIExecute` — Execute a statement

Description

`int` **`OCIExecute`**`(int` *`statement`*`, int` *`[mode]`*`);`

`OCIExecute` executes a previously parsed statement. (see `OCIParse`). The optional *`mode`* allows you to specify the execution-mode (default is OCI_COMMIT_ON_SUCCESS). If you don't want statements to be commited automaticly specify OCI_DEFAULT as your mode.

OCICommit

Name

`OCICommit` — Commits outstanding transactions

Description

`int` **`OCICommit`**`(int` *`connection`*`);`

`OCICommit` commits all outstanding statements for Oracle connection *`connection`*.

OCIRollback

Name

`OCIRollback` — Rolls back outstanding transactions

Description

`int` **`OCIRollback`**`(int` *`connection`*`);`

`OCIRollback` rolls back all outstanding statements for Oracle connection *`connection`*.

OCINewDescriptor

Name

`OCINewDescriptor` — Initialize a new empty descriptor LOB/FILE (LOB is default)

Description

`string` **`OCINewDescriptor`**`(int` *`connection`*`, int` *`[type]`*`);`

`OCINewDescriptor` Allocates storage to hold descriptors or LOB locators. Valid values for the valid *`type`* are OCI_D_FILE, OCI_D_LOB, OCI_D_ROWID. For LOB desriptors, the methods load, save, and savefile are associated with the descriptor, for BFILE only the load method exists. See the second example usage hints.

Example 1. OCINewDescriptor

```
<?php
    /* This script is designed to be called from a HTML form.
     * It expects $user, $password, $table, $where, and $commitsize
     * to be passed in from the form.  The script then deletes
     * the selected rows using the ROWID and commits after each
     * set of $commitsize rows. (Use with care, there is no rollback)
     */
    $conn = OCILogon($user, $password);
    $stmt = OCIParse($conn,"select rowid from $table $where");
    $rowid = OCINewDescriptor($conn,OCI_D_ROWID);
    OCIDefineByName($stmt,"ROWID",&$rowid);
    OCIExecute($stmt);
    while ( OCIFetch($stmt) ) {
       $nrows = OCIRowCount($stmt);
       $delete = OCIParse($conn,"delete from $table where ROWID = :rid");
       OCIBindByName($delete,":rid",&$rowid,-1,OCI_B_ROWID);
       OCIExecute($delete);
       print "$nrows\n";
       if ( ($nrows % $commitsize) == 0 ) {
           OCICommit($conn);
       }
    }
    $nrows = OCIRowCount($stmt);
    print "$nrows deleted...\n";
    OCIFreeStatement($stmt);
    OCILogoff($conn);
?>

<?php
    /* This script demonstrates file upload to LOB columns
     * The formfield used for this example looks like this
     * <form action="upload.php3" method="post" enctype="multipart/form-
data">
     * <input type="file" name="lob_upload">
     * ...
     */
  if(!isset($lob_upload) || $lob_upload == ’none’){
?>
<form action="upload.php3" method="post" enctype="multipart/form-data">
Upload file: <input type="file" name="lob_upload"><br>
<input type="submit" value="Upload"> - <input type="reset">
</form>
<?php
  } else {
     // $lob_upload contains the temporary filename of the uploaded file
     $conn = OCILogon($user, $password);
     $lob = OCINewDescriptor($conn, OCI_D_LOB);
     $stmt = OCIParse($conn,"insert into $table (id, the_blob) val-
ues(my_seq.NEXTVAL, EMPTY_BLOB()) returning the_blob into :the_blob");
```

```
     OCIBindByName($stmt, ':the_blob', &$lob, -1, OCI_B_BLOB);
     OCIExecute($stmt);
     if($lob->savefile($lob_upload)){
        OCICommit($conn);
        echo "Blob successfully uploaded\n";
     }else{
        echo "Couldn't upload Blob\n";
     }
     OCIFreeDescriptor($lob);
     OCIFreeStatement($stmt);
     OCILogoff($conn);
  }
?>
```

OCIRowCount

Name

`OCIRowCount` — Gets the number of affected rows

Description

int **OCIRowCount** (int *statement*);

`OCIRowCounts` returns the number of rows affected for eg update-statements. This funtions will not tell you the number of rows that a select will return!

Example 1. OCIRowCount

```
<?php
    print "<HTML><PRE>";
    $conn = OCILogon("scott","tiger");
    $stmt = OCIParse($conn,"create table emp2 as select * from emp");
    OCIExecute($stmt);
    print OCIRowCount($stmt) . " rows inserted.<BR>";
    OCIFreeStatement($stmt);
    $stmt = OCIParse($conn,"delete from emp2");
    OCIExecute($stmt);
    print OCIRowCount($stmt) . " rows deleted.<BR>";
    OCICommit($conn);
    OCIFreeStatement($stmt);
    $stmt = OCIParse($conn,"drop table emp2");
    OCIExecute($stmt);
    OCIFreeStatement($stmt);
    OCILogOff($conn);
    print "</PRE></HTML>";
?>
```

OCINumCols

Name

`OCINumCols` — Return the number of result columns in a statement

Description

`int` **`OCINumCols`**`(int` *`stmt`*`);`

`OCINumCols` returns the number of columns in a statement

Example 1. OCINumCols

```
<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    while ( OCIFetch($stmt) ) {
        print "\n";
        $ncols = OCINumCols($stmt);
        for ( $i = 1; $i <= $ncols; $i++ ) {
            $column_name  = OCIColumnName($stmt,$i);
            $column_value = OCIResult($stmt,$i);
            print $column_name . ': ' . $column_value . "\n";
        }
        print "\n";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>";
    print "</HTML>\n";
?>
```

OCIResult

Name

`OCIResult` — Returns coulumn value for fetched row

Description

`int` **`OCIResult`**`(int` *`statement`*`, mixed` *`column`*`);`

`OCIResult` returns the data for column *`column`* in the current row (see `OCIFetch`).`OCIResult` will return everything as strings except for abstract types (ROWIDs, LOBs and FILEs).

OCIFetch

Name

`OCIFetch` — Fetches the next row into result-buffer

Description

int **`OCIFetch`**(int *`statement`*);

`OCIFetch` fetches the next row (for SELECT statements) into the internal result-buffer.

OCIFetchInto

Name

`OCIFetchInto` — Fetches the next row into result-array

Description

int **`OCIFetchInto`**(int *`stmt`*, array &*`result`*, int *`[mode]`*);

`OCIFetchInto` fetches the next row (for SELECT statements) into the *`result`* array. `OCIFetchInto` will overwrite the previous content of *`result`*. By default *`result`* will contain a one-based array of all columns that are not NULL.

The *`mode`* parameter allows you to change the default behaviour. You can specify more than one flag by simply addig them up (eg OCI_ASSOC+OCI_RETURN_NULLS). The known flags are:

```
OCI_ASSOC Return an associative array.
OCI_NUM Return an numbered array starting with one. (DEFAULT)
OCI_RETURN_NULLS Return empty columns.
OCI_RETURN_LOBS Return the value of a LOB instead of the desxriptor.
```

OCIFetchStatement

Name

`OCIFetchStatement` — Fetch all rows of result data into an array.

Description

`int` **`OCIFetchStatement`** `(int` *`stmt`*`, array &`*`variable`*`);`

`OCIFetchStatement` fetches all the rows from a result into a user-defined array. `OCIFetchStatement` returns the number of rows fetched.

Example 1. OCIFetchStatement

```
<?php
/* OCIFetchStatement example mbritton@verinet.com (990624) */

$conn = OCILogon("scott","tiger");

$stmt = OCIParse($conn,"select * from emp");

OCIExecute($stmt);

$nrows = OCIFetchStatement($stmt,$results);
if ( $nrows > 0 ) {
   print "<TABLE BORDER=\"1\">\n";
   print "<TR>\n";
   while ( list( $key, $val ) = each( $results ) ) {
      print "<TH>$key</TH>\n";
   }
   print "</TR>\n";

   for ( $i = 0; $i < $nrows; $i++ ) {
      reset($results);
      print "<TR>\n";
      while ( $column = each($results) ) {
         $data = $column['value'];
         print "<TD>$data[$i]</TD>\n";
      }
      print "</TR>\n";
   }
   print "</TABLE>\n";
} else {
   echo "No data found<BR>\n";
}
print "$nrows Records Selected<BR>\n";

OCIFreeStatement($stmt);
OCILogoff($conn);

?>
```

OCIColumnIsNULL

Name

`OCIColumnIsNULL` — test whether a result column is NULL

Description

int **OCIColumnIsNULL**(int *stmt*, mixed *column*);

`OCIColumnIsNULL` returns true if the returned column *`col`* in the result from the statement *`stmt`* is NULL. You can either use the column-number (1-Based) or the column-name for the *`col`* parameter.

OCIColumnSize

Name

`OCIColumnSize` — return result column size

Description

`int` **`OCIColumnSize`**`(int` *`stmt`*`, mixed` *`column`*`);`

`OCIColumnSize` returns the size of the column as given by Oracle. You can either use the column-number (1-Based) or the column-name for the *`col`* parameter.

Example 1. OCIColumnSize

```
<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    print "<TABLE BORDER=\"1\">";
    print "<TR>";
    print "<TH>Name</TH>";
    print "<TH>Type</TH>";
    print "<TH>Length</TH>";
    print "</TR>";
    $ncols = OCINumCols($stmt);
    for ( $i = 1; $i <= $ncols; $i++ ) {
        $column_name  = OCIColumnName($stmt,$i);
        $column_type  = OCIColumnType($stmt,$i);
        $column_size  = OCIColumnSize($stmt,$i);
        print "<TR>";
        print "<TD>$column_name</TD>";
        print "<TD>$column_type</TD>";
        print "<TD>$column_size</TD>";
        print "</TR>";
    }
    print "</TABLE>";
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>";
    print "</HTML>\n";
?>
```

See also `OCINumCols`, `OCIColumnName`, and `OCIColumnSize`.

OCIServerVersion

Name

`OCIServerVersion` — Return a string containing server version information.

Description

```
string OCIServerVersion(int conn);
```

Example 1. OCIServerVersion

```
<?php
   $conn = OCILogon("scott","tiger");
   print "Server Version: " . OCIServerVersion($conn);
   OCILogOff($conn);
?>
```

OCIStatementType

Name

`OCIStatementType` — Return the type of an OCI statement.

Description

```
string OCIStatementType(int stmt);
```

`OCIStatementType` returns on of the following values:

1. "SELECT"

2. "UPDATE"
3. "DELETE"
4. "INSERT"
5. "CREATE"
6. "DROP"
7. "ALTER"
8. "BEGIN"
9. "DECLARE"
10. "UNKNOWN"

Example 1. Code examples

```
<?php
    print "<HTML><PRE>";
    $conn = OCILogon("scott","tiger");
    $sql  = "delete from emp where deptno = 10";

    $stmt = OCIParse($conn,$sql);
    if ( OCIStatementType($stmt) == "DELETE" ) {
        die "You are not allowed to delete from this table<BR>";
    }

    OCILogoff($conn);
    print "</PRE></HTML>";
?>
```

OCINewCursor

Name

`OCINewCursor` — return a new cursor (Statement-Handle) - use this to bind ref-cursors!

Description

int **`OCINewCursor`**(int *conn*);

`OCINewCursor` allocates a new statement handle on the specified connection.

Example 1. Using a REF CURSOR from a stored procedure

```
<?php
// suppose your stored procedure info.output returns a ref cursor in :data

$conn = OCILogon("scott","tiger");
$curs = OCINewCursor($conn);
$stmt = OCIParse($conn,"begin info.output(:data); end;");

ocibindbyname($stmt,"data",&$curs,-1,OCI_B_CURSOR);
ociexecute($stmt);
ociexecute($curs);

while (OCIFetchInto($curs,&$data)) {
    var_dump($data);
}

OCIFreeCursor($stmt);
OCIFreeStatement($curs);
OCILogoff($conn);
?>
```

Example 2. Using a REF CURSOR in a select statement

```
<?php
print "<HTML><BODY>";
$conn = OCILogon("scott","tiger");
$count_cursor = "CURSOR(select count(empno) num_emps from emp " .
                "where emp.deptno = dept.deptno) as EMPCNT from dept";
$stmt = OCIParse($conn,"select deptno,dname,$count_cursor");

ociexecute($stmt);
print "<TABLE BORDER=\"1\">";
print "<TR>";
print "<TH>DEPT NAME</TH>";
print "<TH>DEPT #</TH>";
print "<TH># EMPLOYEES</TH>";
print "</TR>";

while (OCIFetchInto($stmt,&$data,OCI_ASSOC)) {
    print "<TR>";
    $dname  = $data["DNAME"];
    $deptno = $data["DEPTNO"];
    print "<TD>$dname</TD>";
    print "<TD>$deptno</TD>";
    ociexecute($data[ "EMPCNT" ]);
    while (OCIFetchInto($data[ "EMPCNT" ],&$subdata,OCI_ASSOC)) {
        $num_emps = $subdata["NUM_EMPS"];
        print  "<TD>$num_emps</TD>";
    }
    print "</TR>";
}
print "</TABLE>";
print "</BODY></HTML>";
OCIFreeStatement($stmt);
OCILogoff($conn);
?>
```

OCIFreeStatement

Name

`OCIFreeStatement` — Free all resources associated with a statement.

Description

`int` **`OCIFreeStatement`** `(int` *`stmt`*`);`

`OCIFreeStatement` returns true if successful, or false if unsuccessful.

OCIFreeCursor

Name

`OCIFreeCursor` — Free all resources associated with a cursor.

Description

`int` **`OCIFreeCursor`**`(int` *`stmt`*`);`

`OCIFreeCursor` returns true if successful, or false if unsuccessful.

OCIColumnName

Name

`OCIColumnName` — Returns the name of a column.

Description

`string` **`OCIColumnName`**`(int` *`stmt`*`, int` *`col`*`);`

`OCIColumnName` returns the name of the column corresponding to the column number (1-based) that is passed in.

Example 1. OCIColumnName

```
<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    print "<TABLE BORDER=\"1\">";
    print "<TR>";
    print "<TH>Name</TH>";
    print "<TH>Type</TH>";
    print "<TH>Length</TH>";
    print "</TR>";
    $ncols = OCINumCols($stmt);
    for ( $i = 1; $i <= $ncols; $i++ ) {
        $column_name  = OCIColumnName($stmt,$i);
        $column_type  = OCIColumnType($stmt,$i);
        $column_size  = OCIColumnSize($stmt,$i);
        print "<TR>";
        print "<TD>$column_name</TD>";
        print "<TD>$column_type</TD>";
        print "<TD>$column_size</TD>";
        print "</TR>";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>";
    print "</HTML>\n";
?>
```

See also `OCINumCols`, `OCIColumnType`, and `OCIColumnSize`.

OCIColumnType

Name

`OCIColumnType` — Returns the data type of a column.

Description

`mixed` **`OCIColumnName`** `(int` *`stmt`*`, int` *`col`*`);`

`OCIColumnType` returns the data type of the column corresponding to the column number (1-based) that is passed in.

Example 1. OCIColumnType

```
<?php
    print "<HTML><PRE>\n";
    $conn = OCILogon("scott", "tiger");
    $stmt = OCIParse($conn,"select * from emp");
    OCIExecute($stmt);
    print "<TABLE BORDER=\"1\">";
    print "<TR>";
    print "<TH>Name</TH>";
    print "<TH>Type</TH>";
    print "<TH>Length</TH>";
    print "</TR>";
    $ncols = OCINumCols($stmt);
    for ( $i = 1; $i <= $ncols; $i++ ) {
        $column_name  = OCIColumnName($stmt,$i);
        $column_type  = OCIColumnType($stmt,$i);
        $column_size  = OCIColumnSize($stmt,$i);
        print "<TR>";
        print "<TD>$column_name</TD>";
        print "<TD>$column_type</TD>";
        print "<TD>$column_size</TD>";
        print "</TR>";
    }
    OCIFreeStatement($stmt);
    OCILogoff($conn);
    print "</PRE>";
    print "</HTML>\n";
?>
```

See also `OCINumCols`, `OCIColumnName`, and `OCIColumnSize`.

OCIParse

Name

`OCIParse` — Parse a query and return a statement

Description

`int OCIParse(int conn, strint query);`

`OCIParse` parses the *`query`* using *`conn`*. It returns true if the query is valid, false if not. The *`query`* can be any valid SQL statement.

OCIError

Name

`OCIError` — Return the last error of stmt|conn|global. If no error happened returns false.

Description

`int OCIError(int [stmt|conn]);`

`OCIError` returns the last error found. If the optional *`stmt|conn`* is not provided, the last error encountered is returned. If no error is found, `OCIError` returns false.

OCIInternalDebug

Name

`OCIInternalDebug` — Enables or disables internal debug output. By default it is disabled

Description

`void OCIInternalDebug(int onoff);`

`OCIInternalDebug` enables internal debug output. Set *`onoff`* to 0 to turn debug output off, 1 to turn it on.

XV. Oracle functions

Ora_Bind

Name

`Ora_Bind` — bind a PHP variable to an Oracle parameter

Description

`int` **`ora_bind`**`(int` *`cursor`*`, string` *`PHP variable name`*`, string` *`SQL parameter name`*`, int` *`length`*`, int` *`[type]`*`);`

Returns true if the bind succeeds, otherwise false. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

This function binds the named PHP variable with a SQL parameter. The SQL parameter must be in the form ":name". With the optional type parameter, you can define whether the SQL parameter is an in/out (0, default), in (1) or out (2) parameter. As of PHP 3.0.1, you can use the constants ORA_BIND_INOUT, ORA_BIND_IN and ORA_BIND_OUT instead of the numbers.

ora_bind must be called after `ora_parse` and before `ora_exec`. Input values can be given by assignment to the bound PHP variables, after calling `ora_exec` the bound PHP variables contain the output values if available.

```
<?php
ora_parse($curs, "declare tmp INTEGER; be-
gin tmp := :in; :out := tmp; :x := 7.77; end;");
ora_bind($curs, "result", ":x", $len, 2);
ora_bind($curs, "input", ":in", 5, 1);
ora_bind($curs, "output", ":out", 5, 2);
$input = 765;
ora_exec($curs);
echo "Result: $result<BR>Out: $output<BR>In: $input";
?>
```

Ora_Close

Name

`Ora_Close` — close an Oracle cursor

Description

```
int ora_close(int cursor);
```

Returns true if the close succeeds, otherwise false. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

This function closes a data cursor opened with `ora_open`.

Ora_ColumnName

Name

`Ora_ColumnName` — get name of Oracle result column

Description

```
string Ora_ColumnName(int cursor, int column);
```

Returns the name of the field/column *`column`* on the cursor *`cursor`*. The returned name is in all uppercase letters.

Ora_ColumnType

Name

`Ora_ColumnType` — get type of Oracle result column

Description

string **Ora_ColumnType**(int *cursor*, int *column*);

Returns the Oracle data type name of the field/column *`column`* on the cursor *`cursor`*. The returned type will be one of the following:

"VARCHAR2"
"VARCHAR"
"CHAR"
"NUMBER"
"LONG"
"LONG RAW"
"ROWID"
"DATE"
"CURSOR"

Ora_Commit

Name

`Ora_Commit` — commit an Oracle transaction

Description

`int` **`ora_commit`**`(int` *`conn`*`);`

Returns true on success, false on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions. This function commits an Oracle transaction. A transaction is defined as all the changes on a given connection since the last commit/rollback, autocommit was turned off or when the connection was established.

Ora_CommitOff

Name

`Ora_CommitOff` — disable automatic commit

Description

`int` **`ora_commitoff`**`(int` *`conn`*`);`

Returns true on success, false on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

This function turns off automatic commit after each `ora_exec`.

Ora_CommitOn

Name

`Ora_CommitOn` — enable automatic commit

Description

`int ora_commiton(int conn);`

This function turns on automatic commit after each `ora_exec` on the given connection.

Returns true on success, false on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

Ora_Error

Name

`Ora_Error` — get Oracle error message

Description

`string Ora_Error(int cursor_or_connection);`

> Returns an error message of the form *XXX-NNNNN* where *XXX* is where the error comes from and *NNNNN* identifies the error message.
>
> **Note:** Support for connection ids was added in 3.0.4.

On UNIX versions of Oracle, you can find details about an error message like this: `$ oerr ora 00001 00001, 00000, "unique constraint (%s.%s) violated" // *Cause: An update or insert statement attempted to insert a duplicate key // For Trusted ORACLE configured in DBMS MAC mode, you may see // this message if a duplicate entry exists at a different level. // *Action: Either remove the unique restriction or do not insert the key`

Ora_ErrorCode

Name

`Ora_ErrorCode` — get Oracle error code

Description

`int` **`Ora_ErrorCode`**`(int` *`cursor_or_connection`*`);`

Returns the numeric error code of the last executed statement on the specified cursor or connection.

* *FIXME: should possible values be listed?*

> **Note:** Support for connection ids was added in 3.0.4.

Ora_Exec

Name

`Ora_Exec` — execute parsed statement on an Oracle cursor

Description

`int` **`ora_exec`**`(int` *`cursor`*`);`

Returns true on success, false on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

Ora_Fetch

Name

`Ora_Fetch` — fetch a row of data from a cursor

Description

```
int ora_fetch(int cursor);
```

Returns true (a row was fetched) or false (no more rows, or an error occured). If an error occured, details can be retrieved using the `ora_error` and `ora_errorcode` functions. If there was no error, `ora_errorcode` will return 0. Retrieves a row of data from the specified cursor.

Ora_GetColumn

Name

`Ora_GetColumn` — get data from a fetched row

Description

```
mixed ora_getcolumn(int cursor, mixed column);
```

Returns the column data. If an error occurs, False is returned and `ora_errorcode` will return a non-zero value. Note, however, that a test for False on the results from this function may be true in cases where there is not error as well (NULL result, empty string, the number 0, the string "0"). Fetches the data for a column or function result.

Ora_Logoff

Name

`Ora_Logoff` — close an Oracle connection

Description

```
int ora_logoff(int connection);
```

Returns true on success, False on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions. Logs out the user and disconnects from the server.

Ora_Logon

Name

`Ora_Logon` — open an Oracle connection

Description

```
int ora_logon(string user, string password);
```

Establishes a connection between PHP and an Oracle database with the given username and password.

Connections can be made using SQL*Net by supplying the TNS name to *`user`* like this:

```
$conn = Ora_Logon("user@TNSNAME", "pass");
```

If you have character data with non-ASCII characters, you should make sure that NLS_LANG is set in your environment. For server modules, you should set it in the server's environment before starting the server.

Returns a connection index on success, or false on failure. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

Ora_Open

Name

`Ora_Open` — open an Oracle cursor

Description

```
int ora_open(int connection);
```

Opens an Oracle cursor associated with connection.

Returns a cursor index or False on failure. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

Ora_Parse

Name

`Ora_Parse` — parse an SQL statement

Description

```
int ora_parse(int cursor_ind, string sql_statement, int defer);
```

This function parses an SQL statement or a PL/SQL block and associates it with the given cursor. Returns 0 on success or -1 on error.

Ora_Rollback

Name

`Ora_Rollback` — roll back transaction

Description

```
int ora_rollback(int connection);
```

This function undoes an Oracle transaction. (See `ora_commit` for the definition of a transaction.)

Returns true on success, false on error. Details about the error can be retrieved using the `ora_error` and `ora_errorcode` functions.

XVI. Perl-compatible Regular Expression functions

The syntax for patterns used in these functions closely resembles Perl. The expression should be enclosed in the delimiters, a forward slash (/), for example. Any character can be used for delimiter as long as it’s not alphanumeric or backslash (\). If the delimiter character has to be used in the expression itself, it needs to be escaped by backslash.

The ending delimiter may be followed by various modifiers that affect the matching. See Pattern Modifiers.

Example 1. Examples of valid patterns

- /<\/\w+>/
- |(\d{3})-\d+|Sm
- /^(?i)php[34]/

Example 2. Examples of invalid patterns

- /href=’(.*)’ - missing ending delimiter
- /\w+\s*\w+/J - unknown modifier ’J’
- 1-\d3-\d3-\d4| - missing starting delimiter

preg_match

Name

`preg_match` — Perform a regular expression match

Description

```
int preg_match(string pattern, string subject, array [matches]);
```

Searches *`subject`* for a match to the regular expression given in *`pattern`*.

If *`matches`* is provided, then it is filled with the results of search. $matches[0] will contain the text that match the full pattern, $matches[1] will have the text that matched the first captured parenthesized subpattern, and so on.

Returns true if a match for *`pattern`* was found in the subject string, or false if not match was found or an error occurred.

Example 1. Getting the page number out of a string

```
if (preg_match("/page\s+#(\d+)/i", "Go to page #9.", $parts))
    print "Next page is $parts[1]";
else
    print "Page not found.";
```

See also `preg_match_all`, `preg_replace`, and `preg_split`.

preg_match_all

Name

`preg_match_all` — Perform a global regular expression match

Description

int **preg_match_all**(string *pattern*, string *subject*, array *matches*, int *[order]*);

Searches *subject* for all matches to the regular expression given in *pattern* and puts them in *matches* in the order specified by *order*.

After the first match is found, the subsequent searches are continued on from end of the last match.

order can be one of two things:

PREG_PATTERN_ORDER

Orders results so that $matches[0] is an array of full pattern matches, $matches[1] is an array of strings matched by the first parenthesized subpattern, and so on.

```
preg_match_all("|<[^>]+>(.*)</[^>]+>|U", "<b>example: </b><div align=left>a test</div>
print $out[0][0].", ".$out[0][1]."\n";
print $out[1][0].", ".$out[1][1]."\n"
```

This example will produce:

```
<b>example: </b>, <div align=left>this is a test</div>
example: , this is a test
```

So, $out[0] contains array of strings that matched full pattern, and $out[1] contains array of strings enclosed by tags.

PREG_SET_ORDER

Orders results so that $matches[0] is an array of first set of matches, $matches[1] is an array of second set of matches, and so on.

```
preg_match_all("|<[^>]+>(.*)</[^>]+>|U", "<b>example: </b><div align=left>a test</div>
print $out[0][0].", ".$out[0][1]."\n";
print $out[1][0].", ".$out[1][1]."\n"
```

This example will produce:

```
<b>example: </b>, example:
<div align=left>this is a test</div>, this is a test
```

In this case, $matches[0] is the first set of matches, and $matches[0][0] has text matched by full pattern, $matches[0][1] has text matched by first subpattern and so on. Similarly, $matches[1] is the second set of matches, etc.

If *`order`* is not specified, it is assumed to be PREG_PATTERN_ORDER.

Returns the number of full pattern matches, or false if no match is found or an error occurred.

Example 1. Getting all phone numbers out of some text.

```
preg_match_all("/\(?  (\d{3})?  \)?  (?(1)  [\-\s] ) \d{3}-\d{4}/x",
               "Call 555-1212 or 1-800-555-1212", $phones);
```

See also `preg_match`, `preg_replace`, and `preg_split`.

preg_replace

Name

`preg_replace` — Perform a regular expression search and replace

Description

`mixed` **`preg_replace`**`(mixed` *`pattern`*`, mixed` *`replacement`*`, mixed` *`subject`*`);`

Searches *`subject`* for matches to *`pattern`* and replaces them with *`replacement`* .

`replacement` may contain references of the form `\\`*`n`*. Every such reference will be replaced by the text captured by the *`n`*'th parenthesized pattern. *`n`* can be from 0 to 99, and `\\0` refers to the text matched by the whole pattern. Opening parentheses are counted from left to right (starting from 1) to obtain the number of the capturing subpattern.

If no matches are found in *`subject`*, then it will be returned unchanged.

Every parameter to `preg_replace` can be an array.

If *`subject`* is an array, then the search and replace is performed on every entry of *`subject`*, and the return value is an array as well.

If *`pattern`* and *`replacement`* are arrays, then `preg_replace` takes a value from each array and uses them to do search and replace on *`subject`*. If *`replacement`* has fewer values than *`pattern`*, then empty string is used for the rest of replacement values. If *`pattern`* is an array and *`replacement`* is a string; then this replacement string is used for every value of *`pattern`*. The converse would not make sense, though.

> `/e` modifier makes `preg_replace` treat the *`replacement`* parameter as PHP code after the appropriate references substitution is done. Tip: make sure that *`replacement`* constitutes a valid PHP code string, otherwise PHP will complain about a parse error at the line containing `preg_replace`.
>
> **Note:** This modifier was added in PHP 4.0.

Example 1. Replacing several values

```
$patterns = array("/(19|20\d{2})-(\d{1,2})-(\d{1,2})/", "/^\s*{(\w+)}\s*=/");
$replace = array("\\3/\\4/\\1", "$\\1 =");
print preg_replace($patterns, $replace, "{startDate} = 1999-5-27");
```

This example will produce:

```
$startDate = 5/27/1999
```

Example 2. Using /e modifier

```
preg_replace("/(<\/?)(\w+)([^>]*>)/e", "'\\1'.strtoup-
per('\\2').'\\3'", $html_body);
```

This would capitalize all HTML tags in the input text.

See also `preg_match`, `preg_match_all`, and `preg_split`.

preg_split

Name

`preg_split` — Split string by a regular expression

Description

```
array preg_split(string pattern, string subject, int [limit], int [flags]);
```

> **Note:** Parameter *`flags`* was added in PHP Beta 3.

Returns an array containing substrings of *`subject`* split along boundaries matched by *`pattern`*.

If *`limit`* is specified, then only substrings up to *`limit`* are returned.

If flags is PREG_SPLIT_NO_EMPTY then only non-empty pieces will be by `preg_split`.

Example 1. Getting parts of search string

```
$keywords = preg_split("/[\s,]+/", "hypertext language, programming");
```

See also `preg_match`, `preg_match_all`, and `preg_replace`.

preg_quote

Name

`preg_quote` — Quote regular expression characters

Description

```
string preg_quote(string str);
```

`preg_quote` takes *`str`* and puts a backslash in front of every character that is part of the regular expression syntax. This is useful if you have a run-time string that you need to match in some text and the string may contain special regex characters.

The special regular expression characters are:

```
. \\ + * ? [ ^ ] $ ( ) { } = ! < > | :
```

> **Note:** This function was added in PHP 3.0.9.

preg_grep

Name

`preg_grep` — Return array entries that match the pattern

Description

```
array preg_grep(string pattern, array input);
```

`preg_grep` returns the array consisting of the elements of the *`input`* array that match the given *`pattern`*.

Example 1. `preg_grep` example

```
preg_grep("/^(\d+)?\.\d+$/", $array); // find all floating point num-
bers in the array
```

> **Note:** This function was added in PHP 4.0.

Pattern Modifiers

Name

`Pattern Modifiers` — describes possible modifiers in regex patterns

Description

The current possible PCRE modifiers are listed below. The names in parentheses refer to internal PCRE names for these modifiers.

i (PCRE_CASELESS)

If this modifier is set, letters in the pattern match both upper and lower case letters.

m (PCRE_MULTILINE)

By default, PCRE treats the subject string as consisting of a single "line" of characters (even if it actually contains several newlines). The "start of line" metacharacter (^) matches only at the start of the string, while the "end of line" metacharacter ($) matches only at the end of the string, or before a terminating newline (unless *E* modifier is set). This is the same as Perl.

When this modifier is set, the "start of line" and "end of line" constructs match immediately following or immediately before any newline in the subject string, respectively, as well as at the very start and end. This is equivalent to Perl's /m modifier. If there are no "\n" characters in a subject string, or no occurrences of ^ or $ in a pattern, setting this modifier has no effect.

s (PCRE_DOTALL)

If this modifier is set, a dot metacharater in the pattern matches all characters, including newlines. Without it, newlines are excluded. This modifier is equivalent to Perl's /s modifier. A negative class such as [^a] always matches a newline character, independent of the setting of this modifier.

x (PCRE_EXTENDED)

If this modifier is set, whitespace data characters in the pattern are totally ignored except when escaped or inside a character class, and characters between an unescaped # outside a character class and the next newline character, inclusive, are also ignored. This is equivalent to Perl's /x modifier, and makes it possible to include comments inside complicated patterns. Note, however, that this applies only to data characters. Whitespace characters may never appear within special character sequences in a pattern, for example within the sequence (?(which introduces a conditional subpattern.

e

If this modifier is set, `preg_replace` does normal substitution of \\ references in the replacement string, evaluates it as PHP code, and uses the result for replacing the search string.

Only `preg_replace` uses this modifier; it is ignored by other PCRE functions.

> **Note:** This modifier was added in PHP 4.0.

A (PCRE_ANCHORED)

If this modifier is set, the pattern is forced to be "anchored", that is, it is constrained to match only at the start of the string which is being searched (the "subject string"). This effect can also be achieved by appropriate constructs in the pattern itself, which is the only way to do it in Perl.

E (PCRE_DOLLAR_ENDONLY)

If this modifier is set, a dollar metacharacter in the pattern matches only at the end of the subject string. Without this modifier, a dollar also matches immediately before the final character if it is a newline (but not before any other newlines). This modifier is ignored if *m* modifier is set. There is no equivalent to this modifier in Perl.

S

When a pattern is going to be used several times, it is worth spending more time analyzing it in order to speed up the time taken for matching. If this modifier is set, then this extra analysis is performed. At present, studying a pattern is useful only for non-anchored patterns that do not have a single fixed starting character.

U (PCRE_UNGREEDY)

This modifier inverts the "greediness" of the quantifiers so that they are not greedy by default, but become greedy if followed by "?". It is not compatible with Perl. It can also be set by a (?U) modifier setting within the pattern.

X (PCRE_EXTRA)

This modifier turns on additional functionality of PCRE that is incompatible with Perl. Any backslash in a pattern that is followed by a letter that has no special meaning causes an error, thus reserving these combinations for future expansion. By default, as in Perl, a backslash followed by a letter with no special meaning is treated as a literal. There are at present no other features controlled by this modifier.

Pattern Syntax

Name

`Pattern Syntax` — describes PCRE regex syntax

Description

The PCRE library is a set of functions that implement regular expression pattern matching using the same syntax and semantics as Perl 5, with just a few differences (see below). The current implementation corresponds to Perl 5.005.

Differences From Perl

The differences described here are with respect to Perl 5.005.

1. By default, a whitespace character is any character that the C library function isspace() recognizes, though it is possible to compile PCRE with alternative character type tables. Normally isspace() matches space, formfeed, newline, carriage return, horizontal tab, and vertical tab. Perl 5 no longer includes vertical tab in its set of whitespace characters. The \v escape that was in the Perl documentation for a long time was never in fact recognized. However, the character itself was treated as whitespace at least up to 5.002. In 5.004 and 5.005 it does not match \s.

2. PCRE does not allow repeat quantifiers on lookahead assertions. Perl permits them, but they do not mean what you might think. For example, (?!a){3} does not assert that the next three characters are not "a". It just asserts that the next character is not "a" three times.

3. Capturing subpatterns that occur inside negative lookahead assertions are counted, but their entries in the offsets vector are never set. Perl sets its numerical variables from any such patterns that are matched before the assertion fails to match something (thereby succeeding), but only if the negative lookahead assertion contains just one branch.

4. Though binary zero characters are supported in the subject string, they are not allowed in a pattern string because it is passed as a normal C string, terminated by zero. The escape sequence "\0" can be used in the pattern to represent a binary zero.

5. The following Perl escape sequences are not supported: \l, \u, \L, \U, \E, \Q. In fact these are implemented by Perl's general string-handling and are not part of its pattern matching engine.

6. The Perl \G assertion is not supported as it is not relevant to single pattern matches.

7. Fairly obviously, PCRE does not support the (?{code}) construction.

8. There are at the time of writing some oddities in Perl 5.005_02 concerned with the settings of captured strings when part of a pattern is repeated. For example, matching "aba" against the pattern /^(a(b)?)+$/ sets $2 to the value "b", but matching "aabbaa" against /^(aa(bb)?)+$/ leaves $2 unset. However, if the pattern is changed to /^(aa(b(b))?)+$/ then $2 (and $3) get set.

In Perl 5.004 $2 is set in both cases, and that is also true of PCRE. If in the future Perl changes to a consistent state that is different, PCRE may change to follow.

9. Another as yet unresolved discrepancy is that in Perl 5.005_02 the pattern /^(a)?(?(1)a|b)+$/ matches the string "a", whereas in PCRE it does not. However, in both Perl and

PCRE /^(a)?a/ matched against "a" leaves $1 unset.

10. PCRE provides some extensions to the Perl regular expression facilities:

(a) Although lookbehind assertions must match fixed length strings, each alternative branch of a lookbehind assertion can match a different length of string. Perl 5.005 requires them all to have the same length.

(b) If PCRE_DOLLAR_ENDONLY is set and PCRE_MULTILINE is not set, the $ meta- character matches only at the very end of the string.

(c) If PCRE_EXTRA is set, a backslash followed by a letter with no special meaning is faulted.

(d) If PCRE_UNGREEDY is set, the greediness of the repetition quantifiers is inverted, that is, by default they are not greedy, but if followed by a question mark they are.

Regular Expression Details

The syntax and semantics of the regular expressions supported by PCRE are described below. Regular expressions are also described in the Perl documentation and in a number of other books, some of which have copious examples. Jeffrey Friedl’s "Mastering Regular Expressions", published by O’Reilly (ISBN 1-56592-257-3), covers them in great detail. The description here is intended as reference documentation.

A regular expression is a pattern that is matched against a subject string from left to right. Most characters stand for themselves in a pattern, and match the corresponding characters in the subject. As a trivial example, the pattern

```
The quick brown fox
```

matches a portion of a subject string that is identical to itself. The power of regular expressions comes from the ability to include alternatives and repetitions in the pattern. These are encoded in the pattern by the use of *meta-characters*, which do not stand for themselves but instead are interpreted in some special way.

There are two different sets of meta-characters: those that are recognized anywhere in the pattern except within square brackets, and those that are recognized in square brackets. Outside square brackets, the meta-characters are as follows:

\	general escape character with several uses
^	assert start of subject (or line, in multiline mode)
$	assert end of subject (or line, in multiline mode)
.	match any character except newline (by default)
[	start character class definition
|	start of alternative branch
(	start subpattern
)	end subpattern
?	extends the meaning of (also 0 or 1 quantifier also quantifier minimizer
*	0 or more quantifier
+	1 or more quantifier
{	start min/max quantifier

Part of a pattern that is in square brackets is called a "character class". In a character class the only meta-characters are:

\	general escape character
^	negate the class, but only if the first character
-	indicates character range
]	terminates the character class

The following sections describe the use of each of the meta-characters.

BACKSLASH

The backslash character has several uses. Firstly, if it is followed by a non-alphameric character, it takes away any special meaning that character may have. This use of backslash as an escape character applies both inside and outside character classes.

For example, if you want to match a "*" character, you write "*" in the pattern. This applies whether or not the following character would otherwise be interpreted as a metacharacter, so it is always safe to precede a non-alphameric with "\" to specify that it stands for itself. In particular, if you want to match a backslash, you write "\\".

If a pattern is compiled with the PCRE_EXTENDED option, whitespace in the pattern (other than in a character class) and characters between a "#" outside a character class and the next newline character are ignored. An escaping backslash can be used to include a whitespace or "#" character as part of the pattern.

A second use of backslash provides a way of encoding nonprinting characters in patterns in a visible manner. There is no restriction on the appearance of non-printing characters, apart from the binary zero that terminates a pattern, but when a pattern is being prepared by text editing, it is usually easier to use one of the following escape sequences than the binary character it represents:

\a	alarm, that is, the BEL character (hex 07)
\cx	"control-x", where x is any character
\e	escape (hex 1B)
\f	formfeed (hex 0C)
\n	newline (hex 0A)
\r	carriage return (hex 0D)
\t	tab (hex 09)
\xhh	character with hex code hh
\ddd	character with octal code ddd, or backreference

The precise effect of "\cx" is as follows: if "x" is a lower case letter, it is converted to upper case. Then bit 6 of the character (hex 40) is inverted. Thus "\cz" becomes hex 1A, but "\c{" becomes hex 3B, while "\c;" becomes hex 7B.

After "\x", up to two hexadecimal digits are read (letters can be in upper or lower case).

After "\0" up to two further octal digits are read. In both cases, if there are fewer than two digits, just those that are present are used. Thus the sequence "\0\x\07" specifies two binary zeros followed by a BEL character. Make sure you supply two digits after the initial zero if the character that follows is itself an octal digit.

The handling of a backslash followed by a digit other than 0 is complicated. Outside a character class, PCRE reads it and any following digits as a decimal number. If the number is less than 10, or if there have been at least that many previous capturing left parentheses in the expression, the entire sequence is taken as a *back reference*. A description of how this works is given later, following the discussion of parenthesized subpatterns.

Inside a character class, or if the decimal number is greater than 9 and there have not been that many capturing subpatterns, PCRE re-reads up to three octal digits following the backslash, and generates a single byte from the least significant 8 bits of the value. Any subsequent digits stand for themselves. For example:

\040	is another way of writing a space
\40	is the same, provided there are fewer than 40 previous capturing subpatterns
\7	is always a back reference
\11	might be a back reference, or another way of writing a tab
\011	is always a tab
\0113	is a tab followed by the character "3"
\113	is the character with octal code 113 (since there can be no more than 99 back references)

\377	is a byte consisting entirely of 1 bits
\81	is either a back reference, or a binary zero followed by the two characters "8" and "1"

Note that octal values of 100 or greater must not be introduced by a leading zero, because no more than three octal digits are ever read.

All the sequences that define a single byte value can be used both inside and outside character classes. In addition, inside a character class, the sequence "\b" is interpreted as the backspace character (hex 08). Outside a character class it has a different meaning (see below).

The third use of backslash is for specifying generic character types:

\d	any decimal digit
\D	any character that is not a decimal digit
\s	any whitespace character
\S	any character that is not a whitespace character
\w	any "word" character
\W	any "non-word" character

Each pair of escape sequences partitions the complete set of characters into two disjoint sets. Any given character matches one, and only one, of each pair.

A "word" character is any letter or digit or the underscore character, that is, any character which can be part of a Perl "word". The definition of letters and digits is controlled by PCRE’s character tables, and may vary if locale-specific matching is taking place (see "Locale support" above). For example, in the "fr" (French) locale, some character codes greater than 128 are used for accented letters, and these are matched by \w.

These character type sequences can appear both inside and outside character classes. They each match one character of the appropriate type. If the current matching point is at the end of the subject string, all of them fail, since there is no character to match.

The fourth use of backslash is for certain simple assertions. An assertion specifies a condition that has to be met at a particular point in a match, without consuming any characters from the subject string. The use of subpatterns for more complicated assertions is described below. The backslashed assertions are

\b	word boundary
\B	not a word boundary
\A	start of subject (independent of multiline mode)
\Z	end of subject or newline at end (independent of multiline mode)
\z	end of subject (independent of multiline mode)

These assertions may not appear in character classes (but note that "\b" has a different meaning, namely the backspace character, inside a character class).

A word boundary is a position in the subject string where the current character and the previous character do not both match \w or \W (i.e. one matches \w and the other matches \W), or the start or end of the string if the first or last character matches \w, respectively.

The \A, \Z, and \z assertions differ from the traditional circumflex and dollar (described below) in that they only ever match at the very start and end of the subject string, whatever options are set. They are not affected by the PCRE_NOTBOL or PCRE_NOTEOL options. The difference between \Z and \z is that \Z matches before a newline that is the last character of the string as well as at the end of the string, whereas \z matches only at the end.

CIRCUMFLEX AND DOLLAR

Outside a character class, in the default matching mode, the circumflex character is an assertion which is true only if the current matching point is at the start of the subject string. Inside a character class, circumflex has an entirely different meaning (see below).

Circumflex need not be the first character of the pattern if a number of alternatives are involved, but it should be the first thing in each alternative in which it appears if the pattern is ever to match that branch. If all possible alternatives start with a circumflex, that is, if the pattern is constrained to match only at the start of the subject, it is said to be an "anchored" pattern. (There are also other constructs that can cause a pattern to be anchored.)

A dollar character is an assertion which is true only if the current matching point is at the end of the subject string, or immediately before a newline character that is the last character in the string (by default). Dollar need not be the last character of the pattern if a number of alternatives are involved, but it should be the last item in any branch in which it appears. Dollar has no special meaning in a character class.

The meaning of dollar can be changed so that it matches only at the very end of the string, by setting the PCRE_DOLLAR_ENDONLY option at compile or matching time. This does not affect the \Z assertion.

The meanings of the circumflex and dollar characters are changed if the PCRE_MULTILINE option is set. When this is the case, they match immediately after and immediately before an internal "\n" character, respectively, in addition to matching at the start and end of the subject string. For example, the pattern /^abc$/ matches the subject string "def\nabc" in multiline mode, but not otherwise. Consequently, patterns that are anchored in single line mode because all branches start with "^" are not anchored in multiline mode. The PCRE_DOLLAR_ENDONLY option is ignored if PCRE_MULTILINE is set.

Note that the sequences \A, \Z, and \z can be used to match the start and end of the subject in both modes, and if all branches of a pattern start with \A is it always anchored, whether PCRE_MULTILINE is set or not.

FULL STOP (PERIOD, DOT)

Outside a character class, a dot in the pattern matches any one character in the subject, including a non-printing character, but not (by default) newline. If the PCRE_DOTALL option is set, then dots match newlines as well. The handling of dot is entirely independent of the handling of circumflex and dollar, the only relationship being that they both involve newline characters. Dot has no special meaning in a character class.

SQUARE BRACKETS

An opening square bracket introduces a character class, terminated by a closing square bracket. A closing square bracket on its own is not special. If a closing square bracket is required as a member of the class, it should be the first data character in the class (after an initial circumflex, if present) or escaped with a backslash.

A character class matches a single character in the subject; the character must be in the set of characters defined by the class, unless the first character in the class is a circumflex, in which case the subject character must not be in the set defined by the class. If a circumflex is actually required as a member of the class, ensure it is not the first character, or escape it with a backslash.

For example, the character class [aeiou] matches any lower case vowel, while [^aeiou] matches any character that is not a lower case vowel. Note that a circumflex is just a convenient notation for specifying the characters which are in the class by enumerating those that are not. It is not an assertion: it still consumes a character from the subject string, and fails if the current pointer is at the end of the string.

When caseless matching is set, any letters in a class represent both their upper case and lower case versions, so for example, a caseless [aeiou] matches "A" as well as "a", and a caseless [^aeiou] does not match "A", whereas a caseful version would.

The newline character is never treated in any special way in character classes, whatever the setting of the PCRE_DOTALL or PCRE_MULTILINE options is. A class such as [^a] will always match a newline.

The minus (hyphen) character can be used to specify a range of characters in a character class. For example, [d-m] matches any letter between d and m, inclusive. If a minus character is required in a class, it must be escaped with a backslash or appear in a position where it cannot be interpreted as indicating a range, typically as the first or last character in the class.
It is not possible to have the literal character "]" as the end character of a range. A pattern such as [W-]46] is interpreted as a class of two characters ("W" and "-") followed by a literal string "46]", so it would match "W46]" or "-46]". However, if the "]" is escaped with a backslash it is interpreted as the end of range, so [W-\]46] is interpreted as a single class containing a range followed by two separate characters. The octal or hexadecimal representation of "]" can also be used to end a range.

Ranges operate in ASCII collating sequence. They can also be used for characters specified numerically, for example [\000-\037]. If a range that includes letters is used when caseless matching is set, it matches the letters in either case. For example, [W-c] is equivalent to [][\^_‘wxyzabc], matched caselessly, and if character tables for the "fr" locale are in use, [\xc8-\xcb] matches accented E characters in both cases.

The character types \d, \D, \s, \S, \w, and \W may also appear in a character class, and add the characters that they match to the class. For example, [\dABCDEF] matches any hexadecimal digit. A circumflex can conveniently be used with the upper case character types to specify a more restricted set of characters than the matching lower case type. For example, the class [^\W_] matches any letter or digit, but not underscore.

All non-alphameric characters other than \, -, ^ (at the start) and the terminating] are non-special in character classes, but it does no harm if they are escaped.

VERTICAL BAR

Vertical bar characters are used to separate alternative patterns. For example, the pattern

```
gilbert|sullivan
```

matches either "gilbert" or "sullivan". Any number of alternatives may appear, and an empty alternative is permitted (matching the empty string). The matching process tries each alternative in turn, from left to right, and the first one that succeeds is used. If the alternatives are within a subpattern (defined below), "succeeds" means matching the rest of the main pattern as well as the alternative in the subpattern.

INTERNAL OPTION SETTING

The settings of PCRE_CASELESS, PCRE_MULTILINE, PCRE_DOTALL, and PCRE_EXTENDED can be changed from within the pattern by a sequence of Perl option letters enclosed between "(?" and ")". The option letters are

```
i  for PCRE_CASELESS
m  for PCRE_MULTILINE
```

```
s  for PCRE_DOTALL
x  for PCRE_EXTENDED
```

For example, (?im) sets caseless, multiline matching. It is also possible to unset these options by preceding the letter with a hyphen, and a combined setting and unsetting such as (?im-sx), which sets PCRE_CASELESS and PCRE_MULTILINE while unsetting PCRE_DOTALL and PCRE_EXTENDED, is also permitted. If a letter appears both before and after the hyphen, the option is unset.

The scope of these option changes depends on where in the pattern the setting occurs. For settings that are outside any subpattern (defined below), the effect is the same as if the options were set or unset at the start of matching. The following patterns all behave in exactly the same way:

```
(?i)abc
a(?i)bc
ab(?i)c
abc(?i)
```

which in turn is the same as compiling the pattern abc with PCRE_CASELESS set. In other words, such "top level" settings apply to the whole pattern (unless there are other changes inside subpatterns). If there is more than one setting of the same option at top level, the rightmost setting is used.

If an option change occurs inside a subpattern, the effect is different. This is a change of behaviour in Perl 5.005. An option change inside a subpattern affects only that part of the subpattern that follows it, so

```
(a(?i)b)c
```

matches abc and aBc and no other strings (assuming PCRE_CASELESS is not used). By this means, options can be made to have different settings in different parts of the pattern. Any changes made in one alternative do carry on into subsequent branches within the same subpattern. For example,

```
(a(?i)b|c)
```

matches "ab", "aB", "c", and "C", even though when matching "C" the first branch is abandoned before the option setting. This is because the effects of option settings happen at compile time. There would be some very weird behaviour otherwise.

The PCRE-specific options PCRE_UNGREEDY and PCRE_EXTRA can be changed in the same way as the Perl-compatible options by using the characters U and X respectively. The (?X) flag setting is special in that it must always occur earlier in the pattern than any of the additional features it turns on, even when it is at top level. It is best put at the start.

SUBPATTERNS

Subpatterns are delimited by parentheses (round brackets), which can be nested. Marking part of a pattern as a subpattern does two things:

1. It localizes a set of alternatives. For example, the pattern

```
cat(aract|erpillar|)
```

matches one of the words "cat", "cataract", or "caterpillar". Without the parentheses, it would match "cataract", "erpillar" or the empty string.

2. It sets up the subpattern as a capturing subpattern (as defined above). When the whole pattern matches, that portion of the subject string that matched the subpattern is passed back to the caller via the *ovector* argument of `pcre_exec`. Opening parentheses are counted from left to right (starting from 1) to obtain the numbers of the captur-

ing subpatterns.

For example, if the string "the red king" is matched against the pattern

```
  the ((red|white) (king|queen))
```

the captured substrings are "red king", "red", and "king", and are numbered 1, 2, and 3.

The fact that plain parentheses fulfil two functions is not always helpful. There are often times when a grouping subpattern is required without a capturing requirement. If an opening parenthesis is followed by "?:", the subpattern does not do any capturing, and is not counted when computing the number of any subsequent capturing subpatterns. For example, if the string "the white queen" is matched against the pattern

```
  the ((?:red|white) (king|queen))
```

the captured substrings are "white queen" and "queen", and are numbered 1 and 2. The maximum number of captured substrings is 99, and the maximum number of all subpatterns, both capturing and non-capturing, is 200.

As a convenient shorthand, if any option settings are required at the start of a non-capturing subpattern, the option letters may appear between the "?" and the ":". Thus the two patterns

```
  (?i:saturday|sunday)
  (?:(?i)saturday|sunday)
```

match exactly the same set of strings. Because alternative branches are tried from left to right, and options are not reset until the end of the subpattern is reached, an option setting in one branch does affect subsequent branches, so the above patterns match "SUNDAY" as well as "Saturday".

REPETITION

Repetition is specified by quantifiers, which can follow any of the following items:

- a single character, possibly escaped
- the . metacharacter
- a character class
- a back reference (see next section)
- a parenthesized subpattern (unless it is an assertion - see below)

The general repetition quantifier specifies a minimum and maximum number of permitted matches, by giving the two numbers in curly brackets (braces), separated by a comma. The numbers must be less than 65536, and the first must be less than or equal to the second. For example:

```
z{2,4}
```

matches "zz", "zzz", or "zzzz". A closing brace on its own is not a special character. If the second number is omitted, but the comma is present, there is no upper limit; if the second number and the comma are both omitted, the quantifier specifies an exact number of required matches. Thus

```
[aeiou]{3,}
```

matches at least 3 successive vowels, but may match many more, while

```
\d{8}
```

matches exactly 8 digits. An opening curly bracket that appears in a position where a quantifier is not allowed, or one that does not match the syntax of a quantifier, is taken as a literal character. For example, {,6} is not a quantifier, but a literal string of four characters.

The quantifier {0} is permitted, causing the expression to behave as if the previous item and the quantifier were not present.

For convenience (and historical compatibility) the three most common quantifiers have single-character abbreviations:

```
  *    is equivalent to {0,}
  +    is equivalent to {1,}
  ?    is equivalent to {0,1}
```

It is possible to construct infinite loops by following a subpattern that can match no characters with a quantifier that has no upper limit, for example:

```
  (a?)*
```

Earlier versions of Perl and PCRE used to give an error at compile time for such patterns. However, because there are cases where this can be useful, such patterns are now accepted, but if any repetition of the subpattern does in fact match no characters, the loop is forcibly broken.

By default, the quantifiers are "greedy", that is, they match as much as possible (up to the maximum number of permitted times), without causing the rest of the pattern to fail. The classic example of where this gives problems is in trying to match comments in C programs. These appear between the sequences /* and */ and within the sequence, individual * and / characters may appear. An attempt to match C comments by applying the pattern

```
  /\*.*\*/
```

to the string

```
  /* first command */  not comment  /* second comment */
```

fails, because it matches the entire string due to the greediness of the .* item.

However, if a quantifier is followed by a question mark, then it ceases to be greedy, and instead matches the minimum number of times possible, so the pattern

```
  /\*.*?\*/
```

does the right thing with the C comments. The meaning of the various quantifiers is not otherwise changed, just the preferred number of matches. Do not confuse this use of question mark with its use as a quantifier in its own right. Because it has two uses, it can sometimes appear doubled, as in

```
  \d??\d
```

which matches one digit by preference, but can match two if that is the only way the rest of the pattern matches.

If the PCRE_UNGREEDY option is set (an option which is not available in Perl) then the quantifiers are not greedy by default, but individual ones can be made greedy by following them with a question mark. In other words, it inverts the default behaviour.

When a parenthesized subpattern is quantified with a minimum repeat count that is greater than 1 or with a limited maximum, more store is required for the compiled pattern, in proportion to the size of the minimum or maximum.

If a pattern starts with .* or .{0,} and the PCRE_DOTALL option (equivalent to Perl's /s) is set, thus allowing the . to match newlines, then the pattern is implicitly anchored, because whatever follows will be tried against every character position in the subject string, so there is no point in retrying the overall match at any position after the first. PCRE treats such a pattern as though it were preceded by \A. In cases where it is known that the subject string contains no newlines, it is worth setting PCRE_DOTALL when the pattern begins with .* in order to obtain this optimization, or alternatively using ^ to indicate anchoring explicitly.

When a capturing subpattern is repeated, the value captured is the substring that matched the final iteration. For example, after

```
(tweedle[dume]{3}\s*)+
```

has matched "tweedledum tweedledee" the value of the captured substring is "tweedledee". However, if there are nested capturing subpatterns, the corresponding captured values may have been set in previous iterations. For example, after

```
/(a|(b))+/
```

matches "aba" the value of the second captured substring is "b".

BACK REFERENCES

Outside a character class, a backslash followed by a digit greater than 0 (and possibly further digits) is a back reference to a capturing subpattern earlier (i.e. to its left) in the pattern, provided there have been that many previous capturing left parentheses.

However, if the decimal number following the backslash is less than 10, it is always taken as a back reference, and causes an error only if there are not that many capturing left parentheses in the entire pattern. In other words, the parentheses that are referenced need not be to the left of the reference for numbers less than 10. See the section entitled "Backslash" above for further details of the handling of digits following a backslash.

A back reference matches whatever actually matched the capturing subpattern in the current subject string, rather than anything matching the subpattern itself. So the pattern

```
(sens|respons)e and \1ibility
```

matches "sense and sensibility" and "response and responsibility", but not "sense and responsibility". If caseful matching is in force at the time of the back reference, then the case of letters is relevant. For example,

```
((?i)rah)\s+\1
```

matches "rah rah" and "RAH RAH", but not "RAH rah", even though the original capturing subpattern is matched caselessly.

There may be more than one back reference to the same subpattern. If a subpattern has not actually been used in a particular match, then any back references to it always fail. For example, the pattern

```
(a|(bc))\2
```

always fails if it starts to match "a" rather than "bc". Because there may be up to 99 back references, all digits following the backslash are taken as part of a potential back reference number. If the pattern continues with a digit character, then some delimiter must be used to terminate the back reference. If the PCRE_EXTENDED option is set, this can be whitespace. Otherwise an empty comment can be used.

A back reference that occurs inside the parentheses to which it refers fails when the subpattern is first used, so, for example, (a\1) never matches. However, such references can be useful inside repeated subpatterns. For example, the pattern

```
(a|b\1)+
```

matches any number of "a"s and also "aba", "ababaa" etc. At each iteration of the subpattern, the back reference matches the character string corresponding to the previous iteration. In order for this to work, the pattern must be such that the first iteration does not need to match the back reference. This can be done using alternation, as in the example above, or by a quantifier with a minimum of zero.

ASSERTIONS

An assertion is a test on the characters following or preceding the current matching point that does not actually consume any characters. The simple assertions coded as \b, \B, \A, \Z, \z, ^ and $ are described above. More complicated assertions are coded as subpatterns. There are two kinds: those that look ahead of the current position in the subject string, and those that look behind it.

An assertion subpattern is matched in the normal way, except that it does not cause the current matching position to be changed. Lookahead assertions start with (?= for positive assertions and (?! for negative assertions. For example,

```
\w+(?=;)
```

matches a word followed by a semicolon, but does not include the semicolon in the match, and

```
foo(?!bar)
```

matches any occurrence of "foo" that is not followed by "bar". Note that the apparently similar pattern

```
(?!foo)bar
```

does not find an occurrence of "bar" that is preceded by something other than "foo"; it finds any occurrence of "bar" whatsoever, because the assertion (?!foo) is always true when the next three characters are "bar". A lookbehind assertion is needed to achieve this effect.
Lookbehind assertions start with (?<= for positive assertions and (?<! for negative assertions. For example,

```
  (?<!foo)bar
```

does find an occurrence of "bar" that is not preceded by "foo". The contents of a lookbehind assertion are restricted such that all the strings it matches must have a fixed length. However, if there are several alternatives, they do not all have to have the same fixed length. Thus

```
  (?<=bullock|donkey)
```

is permitted, but

```
  (?<!dogs?|cats?)
```

causes an error at compile time. Branches that match different length strings are permitted only at the top level of a lookbehind assertion. This is an extension compared with Perl 5.005, which requires all branches to match the same length of string. An assertion such as

```
  (?<=ab(c|de))
```

is not permitted, because its single top-level branch can match two different lengths, but it is acceptable if rewritten to use two top-level branches:

```
  (?<=abc|abde)
```

The implementation of lookbehind assertions is, for each alternative, to temporarily move the current position back by the fixed width and then try to match. If there are insufficient characters before the current position, the match is deemed to fail. Lookbehinds in conjunction with once-only subpatterns can be particularly useful for matching at the ends of strings; an example is given at the end of the section on once-only subpatterns.

Several assertions (of any sort) may occur in succession. For example,

```
  (?<=\d{3})(?<!999)foo
```

matches "foo" preceded by three digits that are not "999". Furthermore, assertions can be nested in any combination. For example,

```
  (?<=(?<!foo)bar)baz
```

matches an occurrence of "baz" that is preceded by "bar" which in turn is not preceded by "foo".

Assertion subpatterns are not capturing subpatterns, and may not be repeated, because it makes no sense to assert the same thing several times. If an assertion contains capturing subpatterns within it, these are always counted for the purposes of numbering the capturing subpatterns in the whole pattern. Substring capturing is carried out for positive assertions, but it does not make sense for negative assertions.

Assertions count towards the maximum of 200 parenthesized subpatterns.

ONCE-ONLY SUBPATTERNS

With both maximizing and minimizing repetition, failure of what follows normally causes the repeated item to be re-evaluated to see if a different number of repeats allows the rest of the pattern to match. Sometimes it is useful to prevent this, either to change the nature of the match, or to cause it fail earlier than it otherwise might, when the author of the pattern knows there is no point in carrying on.

Consider, for example, the pattern \d+foo when applied to the subject line

```
  123456bar
```

After matching all 6 digits and then failing to match "foo", the normal action of the matcher is to try again with only 5 digits matching the \d+ item, and then with 4, and so on, before ultimately failing. Once-only subpatterns provide the means for specifying that once a portion of the pattern has matched, it is not to be re-evaluated in this way, so the matcher would give up immediately on failing to match "foo" the first time. The notation is another kind of special parenthesis, starting with (?> as in this example:

```
  (?>\d+)bar
```

This kind of parenthesis "locks up" the part of the pattern it contains once it has matched, and a failure further into the pattern is prevented from backtracking into it. Backtracking past it to previous items, however, works as normal.

An alternative description is that a subpattern of this type matches the string of characters that an identical standalone pattern would match, if anchored at the current point in the subject string.

Once-only subpatterns are not capturing subpatterns. Simple cases such as the above example can be thought of as a maximizing repeat that must swallow everything it can. So, while both \d+ and \d+? are prepared to adjust the number of digits they match in order to make the rest of the pattern match, (?>\d+) can only match an entire sequence of digits.

This construction can of course contain arbitrarily complicated subpatterns, and it can be nested.

Once-only subpatterns can be used in conjunction with lookbehind assertions to specify efficient matching at the end of the subject string. Consider a simple pattern such as

```
  abcd$
```

when applied to a long string which does not match it.

Because matching proceeds from left to right, PCRE will look for each "a" in the subject and then see if what follows matches the rest of the pattern. If the pattern is specified as

```
^.*abcd$
```

then the initial .* matches the entire string at first, but when this fails, it backtracks to match all but the last character, then all but the last two characters, and so on. Once again the search for "a" covers the entire string, from right to left, so we are no better off. However, if the pattern is written as

```
^(?>.*)(?<=abcd)
```

then there can be no backtracking for the .* item; it can match only the entire string. The subsequent lookbehind assertion does a single test on the last four characters. If it fails, the match fails immediately. For long strings, this approach makes a significant difference to the processing time.

CONDITIONAL SUBPATTERNS

It is possible to cause the matching process to obey a subpattern conditionally or to choose between two alternative subpatterns, depending on the result of an assertion, or whether a previous capturing subpattern matched or not. The two possible forms of conditional subpattern are

```
(?(condition)yes-pattern)
(?(condition)yes-pattern|no-pattern)
```

If the condition is satisfied, the yes-pattern is used; otherwise the no-pattern (if present) is used. If there are more than two alternatives in the subpattern, a compile-time error occurs.

There are two kinds of condition. If the text between the parentheses consists of a sequence of digits, then the condition is satisfied if the capturing subpattern of that number has previously matched. Consider the following pattern, which contains non-significant white space to make it more readable (assume the PCRE_EXTENDED option) and to divide it into three parts for ease of discussion:

```
  ( \( )?    [^()]+    (?(1) \) )
```

The first part matches an optional opening parenthesis, and if that character is present, sets it as the first captured substring. The second part matches one or more characters that are not parentheses. The third part is a conditional subpattern that tests whether the first set of parentheses matched or not. If they did, that is, if subject started with an opening parenthesis, the condition is true, and so the yes-pattern is executed and a closing parenthesis is required. Otherwise, since no-pattern is not present, the subpattern matches nothing. In other words, this pattern matches a sequence of non-parentheses, optionally enclosed in parentheses.

If the condition is not a sequence of digits, it must be an assertion. This may be a positive or negative lookahead or lookbehind assertion. Consider this pattern, again containing non-significant white space, and with the two alternatives on the second line:

```
  (?(?=[^a-z]*[a-z])
  \d{2}[a-z]{3}-\d{2}  |  \d{2}-\d{2}-\d{2} )
```

The condition is a positive lookahead assertion that matches an optional sequence of non-letters followed by a letter. In other words, it tests for the presence of at least one letter in the subject. If a letter is found, the subject is matched against the first alternative; otherwise it is matched against the second. This pattern matches strings in one of the two forms dd-aaa-dd or dd-dd-dd, where aaa are letters and dd are digits.

COMMENTS

The sequence (?# marks the start of a comment which continues up to the next closing parenthesis. Nested parentheses are not permitted. The characters that make up a comment play no part in the pattern matching at all.

If the PCRE_EXTENDED option is set, an unescaped # character outside a character class introduces a comment that continues up to the next newline character in the pattern.

PERFORMANCE

Certain items that may appear in patterns are more efficient than others. It is more efficient to use a character class like [aeiou] than a set of alternatives such as (a|e|i|o|u). In general, the simplest construction that provides the required behaviour is usually the most efficient. Jeffrey Friedl's book contains a lot of discussion about optimizing regular expressions for efficient performance.

When a pattern begins with .* and the PCRE_DOTALL option is set, the pattern is implicitly anchored by PCRE, since it can match only at the start of a subject string. However, if PCRE_DOTALL is not set, PCRE cannot make this optimization, because the . metacharacter does not then match a newline, and if the subject string contains newlines, the pattern may match from the character immediately following one of them instead of from the very start. For example, the pattern

```
  (.*) second
```

matches the subject "first\nand second" (where \n stands for a newline character) with the first captured substring being "and". In order to do this, PCRE has to retry the match starting after every newline in the subject.

If you are using such a pattern with subject strings that do not contain newlines, the best performance is obtained by setting PCRE_DOTALL, or starting the pattern with ^.* to indicate explicit anchoring. That saves PCRE from having to scan along the subject looking for a newline to restart at.

XVII. PDF functions

You can use the PDF functions in PHP to create PDF files if you have the PDF library by Thomas Merz (available at http://www.pdflib.com/pdflib/index.html). Please consult the excelent documentation for pdflib shipped with the source distribution of pdflib. It provides a very good overview of what pdflib capable of doing. Most of the functions in pdflib and the PHP module have the same name. The parameters are also identical. You should also understand some of the concepts of PDF or Postscript to efficiently use this module. All lengths and coordinates are measured in Postscript points. There are generally 72 PostScript points to an inch, but this depends on the output resolution.

There is another PHP module for pdf document creation based on FastIO's ClibPDF. It has a slightly different API. Check the ClibPDF functions section for details.

Currently all versions of pdflib are supported. It is recommended that you use the newest version since it has more features and fixes some problems which required a patch for the old version. Unfortunately, the changes of the pdflib API in 2.x compared to 0.6 have been so severe that even some PHP functions had to be altered. Here is a list of changes:

- The Info structure does not exist anymore. Therefore the function `pdf_get_info` is obsolete and the functions `pdf_set_info_creator`, `pdf_set_info_title`, `pdf_set_info_author`, `pdf_set_info_subject` and `pdf_set_info_keywords` do not take the info structure as the first parameter but the pdf document. This also means that the pdf document must be opened before these functions can be called.
- The way a new document is opened has changed. The function `pdf_open` takes only one parameter which is the file handle of a file opened with `fopen`.

There were some more changes with the release 2.01 of pdflib which should be covered by PHP. Some functions are not required anymore (e.g. `pdf_put_image`). You will get a warning so don't be shocked.

The pdf module introduces two new types of variables (if pdflib 2.x is used it is only one new type). They are called *`pdfdoc`* and *`pdfinfo`* (*`pdfinfo`* is not existent if pdflib 2.x is used. *`pdfdoc`* is a pointer to a pdf document and almost all functions need it as its first parameter. *`pdfinfo`* contains meta data about the PDF document. It has to be set before `pdf_open` is called.

> **Note:** The following is only true for pdflib 0.6. Read the pdflib manual for newer version

In order to output text into a PDF document you will need to provide the afm file for each font. Afm files contain font metrics for a Postscript font. By default these afm files are searched for in a directory named 'fonts' relative to the directory where the PHP script is located. (Again, this was true for pdflib 0.6, newer versions do not not neccessarily need the afm files.)

Most of the functions are fairly easy to use. The most difficult part is probably to create a very simple pdf document at all. The following example should help to get started. It uses the PHP functions for pdflib

0.6. It creates the file test.pdf with one page. The page contains the text "Times-Roman" in an outlined 30pt font. The text is also underlined.

Example 1. Creating a PDF document with pdflib 0.6

```
<?php
$fp = fopen("test.pdf", "w");
$info = PDF_get_info();
pdf_set_info_author($info, "Uwe Steinmann");
PDF_set_info_title($info, "Test for PHP wrapper of PDFlib 0.6");
PDF_set_info_author($info, "Name of Author");
pdf_set_info_creator($info, "See Author");
pdf_set_info_subject($info, "Testing");
$pdf = PDF_open($fp, $info);
PDF_begin_page($pdf, 595, 842);
PDF_add_outline($pdf, "Page 1");
pdf_set_font($pdf, "Times-Roman", 30, 4);
pdf_set_text_rendering($pdf, 1);
PDF_show_xy($pdf, "Times Roman outlined", 50, 750);
pdf_moveto($pdf, 50, 740);
pdf_lineto($pdf, 330, 740);
pdf_stroke($pdf);
PDF_end_page($pdf);
PDF_close($pdf);
fclose($fp);
echo "<A HREF=getpdf.php3>finished</A>";
?>
```

The PHP script getpdf.php3 just outputs the pdf document.

```
<?php
$fp = fopen("test.pdf", "r");
header("Content-type: application/pdf");
fpassthru($fp);
fclose($fp);
?>
```

Doing the same with pdflib 2.x looks like the following:

Example 2. Creating a PDF document with pdflib 2.0

```
<?php
$fp = fopen("test.pdf", "w");
$pdf = PDF_open($fp);
pdf_set_info_author($pdf, "Uwe Steinmann");
PDF_set_info_title($pdf, "Test for PHP wrapper of PDFlib 2.0");
PDF_set_info_author($pdf, "Name of Author");
pdf_set_info_creator($pdf, "See Author");
pdf_set_info_subject($pdf, "Testing");
PDF_begin_page($pdf, 595, 842);
PDF_add_outline($pdf, "Page 1");
pdf_set_font($pdf, "Times-Roman", 30, 4);
pdf_set_text_rendering($pdf, 1);
PDF_show_xy($pdf, "Times Roman outlined", 50, 750);
pdf_moveto($pdf, 50, 740);
pdf_lineto($pdf, 330, 740);
pdf_stroke($pdf);
PDF_end_page($pdf);
PDF_close($pdf);
fclose($fp);
echo "<A HREF=getpdf.php3>finished</A>";
?>
```

The PHP script getpdf.php3 is the same as above.

The pdflib distribution contains a more complex example which creates a serious of pages with an analog clock. This example converted into PHP using pdflib 2.0 looks as the following (you can see the same example in the documentation for the clibpdf module):

Example 3. pdfclock example from pdflib 2.0 distribution

```
<?php
$pdffilename = "clock.pdf";
$radius = 200;
$margin = 20;
$pagecount = 40;

$fp = fopen($pdffilename, "w");
$pdf = pdf_open($fp);
pdf_set_info_creator($pdf, "pdf_clock.php3");
pdf_set_info_author($pdf, "Uwe Steinmann");
pdf_set_info_title($pdf, "Analog Clock");
```

```
while($pagecount- > 0) {
    pdf_begin_page($pdf, 2 * ($radius + $margin), 2 * ($radius + $margin));

    pdf_set_transition($pdf, 4);  /* wipe */
    pdf_set_duration($pdf, 0.5);

    pdf_translate($pdf, $radius + $margin, $radius + $margin);
    pdf_save($pdf);
    pdf_setrgbcolor($pdf, 0.0, 0.0, 1.0);

    /* minute strokes */
    pdf_setlinewidth($pdf, 2.0);
    for ($alpha = 0; $alpha < 360; $alpha += 6) {
        pdf_rotate($pdf, 6.0);
        pdf_moveto($pdf, $radius, 0.0);
        pdf_lineto($pdf, $radius-$margin/3, 0.0);
        pdf_stroke($pdf);
    }

    pdf_restore($pdf);
    pdf_save($pdf);

    /* 5 minute strokes */
    pdf_setlinewidth($pdf, 3.0);
    for ($alpha = 0; $alpha < 360; $alpha += 30) {
        pdf_rotate($pdf, 30.0);
        pdf_moveto($pdf, $radius, 0.0);
        pdf_lineto($pdf, $radius-$margin, 0.0);
        pdf_stroke($pdf);
    }

    $ltime = getdate();

    /* draw hour hand */
    pdf_save($pdf);
    pdf_rotate($pdf,-(($ltime[’minutes’]/60.0)+$ltime[’hours’]-3.0)*30.0);
    pdf_moveto($pdf, -$radius/10, -$radius/20);
    pdf_lineto($pdf, $radius/2, 0.0);
    pdf_lineto($pdf, -$radius/10, $radius/20);
    pdf_closepath($pdf);
    pdf_fill($pdf);
    pdf_restore($pdf);

    /* draw minute hand */
    pdf_save($pdf);
    pdf_rotate($pdf,-(($ltime['seconds']/60.0)+$ltime['minutes']-15.0)*6.0);
    pdf_moveto($pdf, -$radius/10, -$radius/20);
    pdf_lineto($pdf, $radius * 0.8, 0.0);
    pdf_lineto($pdf, -$radius/10, $radius/20);
    pdf_closepath($pdf);
    pdf_fill($pdf);
    pdf_restore($pdf);

    /* draw second hand */
    pdf_setrgbcolor($pdf, 1.0, 0.0, 0.0);
    pdf_setlinewidth($pdf, 2);
    pdf_save($pdf);
    pdf_rotate($pdf, -(($ltime['seconds'] - 15.0) * 6.0));
    pdf_moveto($pdf, -$radius/5, 0.0);
    pdf_lineto($pdf, $radius, 0.0);
    pdf_stroke($pdf);
    pdf_restore($pdf);

    /* draw little circle at center */
    pdf_circle($pdf, 0, 0, $radius/30);
    pdf_fill($pdf);

    pdf_restore($pdf);

    pdf_end_page($pdf);
}

$pdf = pdf_close($pdf);
fclose($fp);
echo "<A HREF=getpdf.php3?filename=".$pdffilename.">finished</A>";
?>
```

The PHP script getpdf.php3 just outputs the pdf document.

```
<?php
$fp = fopen($filename, "r");
header("Content-type: application/pdf");
fpassthru($fp);
fclose($fp);
?>
```

PDF_get_info

Name

`PDF_get_info` — Returns a default info structure for a pdf document

Description

info **`pdf_get_info`**(string *filename*);

> The `PDF_get_info` function returns a default info structure for the pdf document. It should be filled with appropriate information like the author, subject etc. of the document.
>
> **Note:** This functions is not available if pdflib 2.0 support is activated.

See also `PDF_set_info_creator`, `PDF_set_info_author`, `PDF_set_info_keywords`, `PDF_set_info_title`, `PDF_set_info_subject`.

PDF_set_info_creator

Name

`PDF_set_info_creator` — Fills the creator field of the info structure

Description

void **`pdf_set_info_creator`**(info *info*, string *creator*);

> The `PDF_set_info_creator` function sets the creator of a pdf document. It has to be called after `PDF_get_info` and before `PDF_open`. Calling it after `PDF_open` will have no effect on the document.

Note: This function is not part of the pdf library.

Note: This function takes a different first parameter if pdflib 2.0 support is activated. The first parameter has to be the identifier of the pdf document as returned by `pdf_open`. Consequently, `pdf_open` has to be called before this function.

See also `PDF_get_info`, `PDF_set_info_keywords`, `PDF_set_info_title`, `PDF_set_info_subject`.

PDF_set_info_title

Name

`PDF_set_info_title` — Fills the title field of the info structure

Description

```
void pdf_set_info_title(info info, string title);
```

The `PDF_set_info_title` function sets the title of a pdf document. It has to be called after `PDF_get_info` and before `PDF_open`. Calling it after `PDF_open` will have no effect on the document.

Note: This function is not part of the pdf library.

Note: This function takes a different first parameter if pdflib 2.0 support is activated. The first parameter has to be the identifier of the pdf document as returned by `pdf_open`. Consequently, `pdf_open` has to be called before this function.

See also `PDF_get_info`, `PDF_set_info_creator`, `PDF_set_info_author`, `PDF_set_info_keywords`, `PDF_set_info_subject`.

PDF_set_info_subject

Name

`PDF_set_info_subject` — Fills the subject field of the info structure

Description

void **`pdf_set_info_subject`** (info *info*, string *subject*);

> The `PDF_set_info_subject` function sets the subject of a pdf document. It has to be called after `PDF_get_info` and before `PDF_open`. Calling it after `PDF_open` will have no effect on the document.
>
> **Note:** This function is not part of the pdf library.
>
> **Note:** This function takes a different first parameter if pdflib 2.0 support is activated. The first parameter has to be the identifier of the pdf document as returned by `pdf_open`. Consequently, `pdf_open` has to be called before this function.

See also `PDF_get_info`, `PDF_set_info_creator`, `PDF_set_info_author`, `PDF_set_info_title`, `PDF_set_info_keywords`.

PDF_set_info_keywords

Name

`PDF_set_info_keywords` — Fills the keywords field of the info structure

Description

void **`pdf_set_info_keywords`** `(info` *`info`*`, string` *`keywords`*`);`

The `PDF_set_info_keywords` function sets the keywords of a pdf document. It has to be called after `PDF_get_info` and before `PDF_open`. Calling it after `PDF_open` will have no effect on the document.

Note: This function is not part of the pdf library.

Note: This function takes a different first parameter if pdflib 2.0 support is activated. The first parameter has to be the identifier of the pdf document as returned by `pdf_open`. Consequently, `pdf_open` has to be called before this function.

See also `PDF_get_info`, `PDF_set_info_creator`, `PDF_set_info_author`, `PDF_set_info_title`, `PDF_set_info_subject`.

PDF_set_info_author

Name

`PDF_set_info_author` — Fills the author field of the info structure

Description

void **`pdf_set_info_author`** `(info` *`info`*`, string` *`author`*`);`

The `PDF_set_info_author` function sets the author of a pdf document. It has to be called after `PDF_get_info` and before `PDF_open`. Calling it after `PDF_open` will have no effect on the document.

Note: This function is not part of the pdf library.

Note: This function takes a different first parameter if pdflib 2.0 support is activated. The first parameter has to be the identifier of the pdf document as returned by `pdf_open`. Consequently, `pdf_open` has to be called before this function.

See also `PDF_get_info`, `PDF_set_info_creator`, `PDF_set_info_keywords`, `PDF_set_info_title`, `PDF_set_info_subject`.

PDF_open

Name

`PDF_open` — Opens a new pdf document

Description

```
int pdf_open(int file, int info);
```

The `PDF_open` function opens a new pdf document. The corresponding file has to be opened with `fopen` and the file descriptor passed as argument *`file`*. *`info`* is the info structure that has to be created with `pdf_get_info`. The info structure will be deleted within this function.

Note: The return value is needed as the first parameter in all other functions writing to the pdf document.

Note: This function does not allow the second parameter if pdflib 2.0 support is activated.

See also `fopen`, `PDF_get_info`, `PDF_close`.

PDF_close

Name

`PDF_close` — Closes a pdf document

Description

void **pdf_close**(int *pdf document*);

The `PDF_close` function closes the pdf document.

Note: Due to an unclean implementation of the pdflib 0.6 the internal closing of the document also closes the file. This should not be done because pdflib did not open the file, but expects an already open file when `PDF_open` is called. Consequently it shouldn't close the file. In order to fix this just take out line 190 of the file p_basic.c in the pdflib 0.6 source distribution until the next release of pdflib will fix this.

Note: This function works properly without any patches to pdflib if pdflib 2.0 support is activated.

See also `PDF_open`, `fclose`.

PDF_begin_page

Name

`PDF_begin_page` — Starts new page

Description

void **pdf_begin_page**(int *pdf document*, double *height*, double *width*);

The `PDF_begin_page` function starts a new page with height *`height`* and width *`width`*. In order to create a valid document you must call this function and `PDF_end_page`.

See also `PDF_end_page`.

PDF_end_page

Name

`PDF_end_page` — Ends a page

Description

`void` **`pdf_end_page`**`(int` *`pdf document`*`);`

The `PDF_end_page` function ends a page. Once a page is ended it cannot be modified anymore.

See also `PDF_begin_page`.

PDF_show

Name

`PDF_show` — Output text at current position

Description

`void` **`pdf_show`**`(int` *`pdf document`*`, string` *`text`*`);`

The `PDF_show` function outputs the string *`text`* at the current position using the current font.

See also `PDF_show_xy`, `PDF_set_text_pos`, `PDF_set_font`.

PDF_show_xy

Name

`PDF_show_xy` — Output text at given position

Description

`void` **`pdf_show_xy`**`(int` *`pdf document`*`, string` *`text`*`, double` *`x-koor`*`, double` *`y-koor`*`);`

The `PDF_show_xy` function outputs the string *`text`* at position (*`x-koor`*, *`y-koor`*).

See also `PDF_show`.

PDF_set_font

Name

`PDF_set_font` — Selects a font face and size

Description

`void` **`pdf_set_font`**`(int` *`pdf document`*`, string` *`font name`*`, double` *`size`*`, int` *`encoding`*`, int` *`embed`*`);`

The `PDF_set_font` function sets the current font face, font size and encoding. You will need to provide the Adobe Font Metrics (afm-files) for the font in the font path (default is ./fonts). The last parameter *`encoding`* can take the following values: 0 = builtin, 1 = pdfdoc, 2 = macroman, 3 = macexpert, 4 = winansi. An encoding greater than 4 and less than 0 will default to winansi. winansi is often a good choice. If the last parameter is set to 1 the font is embedded into the pdf document otherwise it is not.

Note: This function has to be called after `PDF_begin_page` in order to create a valid pdf document.

Note: This function does not need the afm files for winansi encoding if pdflib 2.0 support is activated.

PDF_set_leading

Name

`PDF_set_leading` — Sets distance between text lines

Description

void **pdf_set leading**(int *pdf document*, double *distance*);

The `PDF_set_leading` function sets the distance between text lines. This will be used if text is output by `PDF_continue_text`.

See also `PDF_continue_text`.

PDF_set_text_rendering

Name

`PDF_set_text_rendering` — Determines how text is rendered

Description

void **pdf_set_text_rendering**(int *pdf document*, int *mode*);

The `PDF_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 cliping path, 5=stroke text and add it to clipping path, 6=fill and stroke text and add it to cliping path, 7=add it to clipping path.

PDF_set_horiz_scaling

Name

`PDF_set_horiz_scaling` — Sets horizontal scaling of text

Description

`void` **`pdf_set_horiz_scaling`** `(int` *`pdf document`*`, double` *`scale`*`);`

The `PDF_set_horiz_scaling` function sets the horizontal scaling to *`scale`* percent.

PDF_set_text_rise

Name

`PDF_set_text_rise` — Sets the text rise

Description

`void` **`pdf_set_text_rise`** `(int` *`pdf document`*`, double` *`value`*`);`

The `PDF_set_text_rise` function sets the text rising to *`value`* points.

PDF_set_text_matrix

Name

`PDF_set_text_matrix` — Sets the text matrix

Description

void **pdf_set_text_matrix** (int *pdf document*, array *matrix*);

The `PDF_set_text_matrix` function sets a matrix which describes a transformation applied on the current text font. The matrix has to passed as an array with six elements.

PDF_set_text_pos

Name

`PDF_set_text_pos` — Sets text position

Description

void **pdf_set_text_pos** (int *pdf document*, double *x-koor*, double *y-koor*);

The `PDF_set_text_pos` function sets the position of text for the next `pdf_show` function call.

See also `PDF_show`, `PDF_show_xy`.

PDF_set_char_spacing

Name

`PDF_set_char_spacing` — Sets character spacing

Description

```
void pdf_set_char_spacing(int pdf document, double space);
```

The `PDF_set_char_spacing` function sets the spacing between characters.

See also `PDF_set_word_spacing`, `PDF_set_leading`.

PDF_set_word_spacing

Name

`PDF_set_word_spacing` — Sets spacing between words

Description

```
void pdf_set_word_spacing(int pdf document, double space);
```

The `PDF_set_word_spacing` function sets the spacing between words.

See also `PDF_set_char_spacing`, `PDF_set_leading`.

PDF_continue_text

Name

`PDF_continue_text` — Outputs text in next line

Description

```
void pdf_continue_text(int pdf document, string text);
```

The `PDF_continue_text` function outputs the string in *`text`* in the next line. The distance between the lines can be set with `PDF_set_leading`.

See also `PDF_show_xy`, `PDF_set_leading`, `PDF_set_text_pos`.

PDF_stringwidth

Name

`PDF_stringwidth` — Returns width of text using current font

Description

```
double pdf_stringwidth(int pdf document, string text);
```

The `PDF_stringwidth` function returns the width of the string in *`text`*. It requires a font to be set before.

See also `PDF_set_font`.

PDF_save

Name

`PDF_save` — Saves the current environment

Description

```
void pdf_save(int pdf document);
```

The `PDF_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. `PDF_save` should always be followed by `PDF_restore`.

See also `PDF_restore`.

PDF_restore

Name

`PDF_restore` — Restores formerly saved environment

Description

```
void pdf_restore(int pdf document);
```

The `PDF_restore` function restores the environment saved with `PDF_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 and Restore

```
<?php PDF_save($pdf);
```

```
// do all kinds of rotations, transformations, ...
PDF_restore($pdf) ?>
```

See also `PDF_save`.

PDF_translate

Name

`PDF_translate` — Sets origin of coordinate system

Description

`void` **`pdf_translate`**`(int` *`pdf document`*`, double` *`x-koor`*`, double` *`y-koor`*`);`

The `PDF_translate` function set the origin of coordinate system to the point (*`x-koor`*, *`y-koor`*). The following example draws a line from (0, 0) to (200, 200) relative to the initial coordinate system. You have to set the current point after `PDF_translate` and before you start drawing more objects.

Example 1. Translation

```
<?php PDF_moveto($pdf, 0, 0);
PDF_lineto($pdf, 100, 100);
PDF_stroke($pdf);
PDF_translate($pdf, 100, 100);
PDF_moveto($pdf, 0, 0);
PDF_lineto($pdf, 100, 100);
PDF_stroke($pdf);
?>
```

PDF_scale

Name

`PDF_scale` — Sets scaling

Description

void **pdf_scale**(int *pdf document*, double *x-scale*, double *y-scale*);

The `PDF_scale` function set the scaling factor in both directions. The following example scales x and y direction by 72. The following line will therefore be drawn one inch in both directions.

Example 1. Scaling

```
<?php PDF_scale($pdf, 72.0, 72.0);
PDF_lineto($pdf, 1, 1);
PDF_stroke($pdf);
?>
```

PDF_rotate

Name

`PDF_rotate` — Sets rotation

Description

void **pdf_rotate**(int *pdf document*, double *angle*);

The `PDF_rotate` function set the rotation in degress to *angle*.

PDF_setflat

Name

`PDF_setflat` — Sets flatness

Description

`void` **`pdf_setflat`**`(int` *`pdf document`*`, double` *`value`*`);`

The `PDF_setflat` function set the flatness to a value between 0 and 100.

PDF_setlinejoin

Name

`PDF_setlinejoin` — Sets linejoin parameter

Description

`void` **`pdf_setlinejoin`**`(int` *`pdf document`*`, long` *`value`*`);`

The `PDF_setlinejoin` function set the linejoin parameter between a value of 0 and 2.

PDF_setlinecap

Name

`PDF_setlinecap` — Sets linecap parameter

Description

`void` **`pdf_setlinecap`**`(int` *`pdf document`*`, int` *`value`*`);`

The `PDF_setlinecap` function set the linecap parameter between a value of 0 and 2.

PDF_setmiterlimit

Name

`PDF_setmiterlimit` — Sets miter limit

Description

`void` **`pdf_setmiterlimit`**`(int` *`pdf document`*`, double` *`value`*`);`

The `PDF_setmiterlimit` function set the miter limit to a value greater of equal than 1.

PDF_setlinewidth

Name

`PDF_setlinewidth` — Sets line width

Description

```
void pdf_setlinewidth(int pdf document, double width);
```

The `PDF_setlinewidth` function set the line width to *`width`*.

PDF_setdash

Name

`PDF_setdash` — Sets dash pattern

Description

```
void pdf_setdash(int pdf document, double white, double black);
```

The `PDF_setdash` function set the dash pattern *`white`* white points and *`black`* black points. If both are 0 a solid line is set.

PDF_moveto

Name

`PDF_moveto` — Sets current point

Description

`void` **`pdf_moveto`**`(int` *`pdf document`*`, double` *`x-koor`*`, double` *`y-koor`*`);`

The `PDF_moveto` function set the current point to the coordinates *`x-koor`* and *`y-koor`*.

PDF_curveto

Name

`PDF_curveto` — Draws a curve

Description

`void` **`pdf_curveto`**`(int` *`pdf document`*`, double` *`x1`*`, double` *`y1`*`, double` *`x2`*`, double` *`y2`*`, double` *`x3`*`, double` *`y3`*`);`

The `PDF_curveto` function draws a Bezier curve from the current point to the point (*`x3`*, *`y3`*) using (*`x1`*, *`y1`*) and (*`x2`*, *`y2`*) as control points.

See also `PDF_moveto`, `PDF_lineto`, `PDF_stroke`.

PDF_lineto

Name

`PDF_lineto` — Draws a line

Description

```
void pdf_lineto(int pdf document, double x-koor, double y-koor);
```

The `PDF_lineto` function draws a line from the current point to the point with coordinates (*`x-koor`*, *`y-koor`*).

See also `PDF_moveto`, `PDF_curveto`, `PDF_stroke`.

PDF_circle

Name

`PDF_circle` — Draws a circle

Description

```
void pdf_circle(int pdf document, double x-koor, double y-koor, double
radius);
```

The `PDF_circle` function draws a circle with center at point (*`x-koor`*, *`y-koor`*) and radius *`radius`*.

See also `PDF_arc`, `PDF_stroke`.

PDF_arc

Name

`PDF_arc` — Draws an arc

Description

```
void pdf_arc(int pdf document, double x-koor, double y-koor, double radius,
double start, double end);
```

The `PDF_arc` function draws an arc with center at point (*`x-koor`*, *`y-koor`*) and radius *`radius`*, starting at angle *`start`* and ending at angle *`end`*.

See also `PDF_circle`, `PDF_stroke`.

PDF_rect

Name

`PDF_rect` — Draws a rectangle

Description

```
void pdf_rect(int pdf document, double x-koor, double y-koor, double width,
double height);
```

The `PDF_rect` function draws a rectangle with its lower left corner at point (*`x-koor`*, *`y-koor`*). This width is set to *`widgth`*. This height is set to *`height`*.

See also `PDF_stroke`.

PDF_closepath

Name

`PDF_closepath` — Closes path

Description

`void` **`pdf_closepath`**`(int` *`pdf document`*`);`

The `PDF_closepath` function closes the current path. This means, it draws a line from current point to the point where the first line was started. Many functions like `PDF_moveto`, `PDF_circle` and `PDF_rect` start a new path.

PDF_stroke

Name

`PDF_stroke` — Draws line along path

Description

`void` **`pdf_stroke`**`(int` *`pdf document`*`);`

The `PDF_stroke` function draws a line along current path. The current path is the sum of all line drawing. Without this function the line would not be drawn.

See also `PDF_closepath`, `PDF_closepath_stroke`.

PDF_closepath_stroke

Name

`PDF_closepath_stroke` — Closes path and draws line along path

Description

```
void pdf_closepath_stroke(int pdf document);
```

The `PDF_closepath_stroke` function is a combination of `PDF_closepath` and `PDF_stroke`. Than clears the path.

See also `PDF_closepath`, `PDF_stroke`.

PDF_fill

Name

`PDF_fill` — Fills current path

Description

```
void pdf_fill(int pdf document);
```

The `PDF_fill` function fills the interior of the current path with the current fill color.

See also `PDF_closepath`, `PDF_stroke`, `PDF_setgray_fill`, `PDF_setgray`, `PDF_setrgbcolor_fill`, `PDF_setrgbcolor`.

PDF_fill_stroke

Name

`PDF_fill_stroke` — Fills and strokes current path

Description

void **pdf_fill_stroke**(int *pdf document*);

The `PDF_fill_stroke` function fills the interior of the current path with the current fill color and draws current path.

See also `PDF_closepath`, `PDF_stroke`, `PDF_fill`, `PDF_setgray_fill`, `PDF_setgray`, `PDF_setrgbcolor_fill`, `PDF_setrgbcolor`.

PDF_closepath_fill_stroke

Name

`PDF_closepath_fill_stroke` — Closes, fills and strokes current path

Description

void **pdf_closepath_fill_stroke**(int *pdf document*);

The `PDF_closepath_fill_stroke` function closes, fills the interior of the current path with the current fill color and draws current path.

See also `PDF_closepath`, `PDF_stroke`, `PDF_fill`, `PDF_setgray_fill`, `PDF_setgray`, `PDF_setrgbcolor_fill`, `PDF_setrgbcolor`.

PDF_endpath

Name

`PDF_endpath` — Ends current path

Description

```
void pdf_endpath(int pdf document);
```

The `PDF_endpath` function ends the current path but does not close it.

See also `PDF_closepath`.

PDF_clip

Name

`PDF_clip` — Clips to current path

Description

```
void pdf_clip(int pdf document);
```

The `PDF_clip` function clips all drawing to the current path.

PDF_setgray_fill

Name

`PDF_setgray_fill` — Sets filling color to gray value

Description

`void` **`pdf_setgray_fill`**`(int` *`pdf document`*`, double` *`value`*`);`

The `PDF_setgray_fill` function sets the current gray value to fill a path.

See also `PDF_setrgbcolor_fill`.

PDF_setgray_stroke

Name

`PDF_setgray_stroke` — Sets drawing color to gray value

Description

`void` **`pdf_setgray_stroke`**`(int` *`pdf document`*`, double` *`gray value`*`);`

The `PDF_setgray_stroke` function sets the current drawing color to the given gray value.

See also `PDF_setrgbcolor_stroke`.

PDF_setgray

Name

`PDF_setgray` — Sets drawing and filling color to gray value

Description

```
void pdf_setgray(int pdf document, double gray value);
```

The `PDF_setgray_stroke` function sets the current drawing and filling color to the given gray value.

See also `PDF_setrgbcolor_stroke`, `PDF_setrgbcolor_fill`.

PDF_setrgbcolor_fill

Name

`PDF_setrgbcolor_fill` — Sets filling color to rgb color value

Description

```
void pdf_setrgbcolor_fill(int pdf document, double red value, double green
value, double blue value);
```

The `PDF_setrgbcolor_fill` function sets the current rgb color value to fill a path.

See also `PDF_setrgbcolor_fill`.

PDF_setrgbcolor_stroke

Name

`PDF_setrgbcolor_stroke` — Sets drawing color to rgb color value

Description

void **pdf_setrgbcolor_stroke**(int *pdf document*, double *red value*, double *green value*, double *blue value*);

The `PDF_setrgbcolor_stroke` function sets the current drawing color to the given rgb color value.

See also `PDF_setrgbcolor_stroke`.

PDF_setrgbcolor

Name

`PDF_setrgbcolor` — Sets drawing and filling color to rgb color value

Description

void **pdf_setrgbcolor**(int *pdf document*, double *red value*, double *green value*, double *blue value*);

The `PDF_setrgbcolor_stroke` function sets the current drawing and filling color to the given rgb color value.

See also `PDF_setrgbcolor_stroke`, `PDF_setrgbcolor_fill`.

PDF_add_outline

Name

`PDF_add_outline` — Adds bookmark for current page

Description

void **pdf_add_outline**(int *pdf document*, string *text*);

The `PDF_add_outline` function adds a bookmark with text *`text`* that points to the current page.

Unfortunately pdflib does not make a copy of the string, which forces PHP to allocate the memory. Currently this piece of memory is not been freed by any PDF function but it will be taken care of by the PHP memory manager.

PDF_set_transition

Name

`PDF_set_transition` — Sets transition between pages

Description

void **pdf_set_transition**(int *pdf document*, int *transition*);

The `PDF_set_transition` 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)

See also `PDF_set_duration`.

PDF_set_duration

Name

`PDF_set_duration` — Sets duration between pages

Description

`void` **`pdf_set_duration`**`(int` *`pdf document`*`, double` *`duration`*`);`

The `PDF_set_duration` function set the duration between following pages in seconds.

See also `PDF_set_transition`.

PDF_open_gif

Name

`PDF_open_gif` — Opens a GIF image

Description

`int` **`pdf_open_gif`**`(int` *`pdf document`*`, string` *`filename`*`);`

The `PDF_open_gif` function opens an image stored in the file with the name *`filename`*. The format of the image has to be gif. The function returns a pdf image identifier.

Example 1. Including a gif image

```
<?php
$im = PDF_open_gif($pdf, "test.gif");
pdf_place_image($pdf, $im, 100, 100, 1);
pdf_close_image($pdf, $im);
?>
```

See also `PDF_close_image`, `PDF_open_jpeg`, `PDF_open_memory_image`, `PDF_execute_image`, `PDF_place_image`, `PDF_put_image`.

PDF_open_memory_image

Name

`PDF_open_memory_image` — Opens an image created with PHP's image functions

Description

`int` **`pdf_open_memory_image`**`(int` *`pdf document`*`, string` *`int image`*`);`

The `PDF_open_memory_image` function takes an image created with the PHP's image functions and makes it available for the pdf document. The function returns a pdf image identifier.

Example 1. Including a memory image

```
<?php
$im = ImageCreate(100, 100);
```

```
$col = ImageColorAllocate($im, 80, 45, 190);
ImageFill($im, 10, 10, $col);
$pim = PDF_open_memory_image($pdf, $im);
ImageDestroy($im);
pdf_place_image($pdf, $pim, 100, 100, 1);
pdf_close_image($pdf, $pim);
?>
```

See also `PDF_close_image`, `PDF_open_jpeg`, `PDF_open_gif`, `PDF_execute_image`, `PDF_place_image`, `PDF_put_image`.

PDF_open_jpeg

Name

`PDF_open_jpeg` — Opens a JPEG image

Description

`int` **`pdf_open_jpeg`**`(int` *`pdf document`*`, string` *`filename`*`);`

The `PDF_open_jpeg` function opens an image stored in the file with the name *`filename`*. The format of the image has to be jpeg. The function returns a pdf image identifier.

See also `PDF_close_image`, `PDF_open_gif`, `PDF_open_memory_image`, `PDF_execute_image`, `PDF_place_image`, `PDF_put_image`.

PDF_close_image

Name

`PDF_close_image` — Closes an image

Description

`void` **`pdf_close_image`**`(int` *`image`*`);`

The `PDF_close_image` function closes an image which has been opened with any of the `PDF_open_xxx` functions.

See also `PDF_open_jpeg`, `PDF_open_gif`, `PDF_open_memory_image`.

PDF_place_image

Name

`PDF_place_image` — Places an image on the page

Description

`void` **`pdf_place_image`**`(int` *`pdf document`*`, int` *`image`*`, double` *`x-koor`*`, double` *`y-koor`*`, double` *`scale`*`);`

The `PDF_place_image` function places an image on the page at postion (*`x-koor`*, *`x-koor`*). The image can be scaled at the same time.

See also `PDF_put_image`.

PDF_put_image

Name

`PDF_put_image` — Stores an image in the PDF for later use

Description

`void` **`pdf_put_image`**`(int` *`pdf document`*`, int` *`image`*`);`

The PDF_put_image function places an image in the PDF file without showing it. The stored image can be displayed with the `PDF_execute_image` function as many times as needed. This is useful when using the same image multiple times in order to keep the file size small. Using `PDF_put_image` and `PDF_execute_image` is highly recommended for larger images (several kb) if they show up more than once in the document.

Note: This function has become meaningless with version 2.01 of pdflib. It will just output a warning.

See also `PDF_put_image`, `PDF_place_image`, `PDF_execute_image`.

PDF_execute_image

Name

`PDF_execute_image` — Places a stored image on the page

Description

`void` **`pdf_execute_image`**`(int` *`pdf document`*`, int` *`image`*`, double` *`x-coor`*`, double` *`y-coor`*`, double` *`scale`*`);`

The PDF_execute_image function displays an image that has been put in the PDF file with the `PDF_put_image` function on the current page at the given coordinates.

The image can be scaled while displaying it. A scale of 1.0 will show the image in the original size.

Note: This function has become meaningless with version 2.01 of pdflib. It will just output a warning.

Example 1. Multiple show of an image

```
<?php
$im = ImageCreate(100, 100);
$col1 = ImageColorAllocate($im, 80, 45, 190);
ImageFill($im, 10, 10, $col1);
$pim = PDF_open_memory_image($pdf, $im);
pdf_put_image($pdf, $pim);
pdf_execute_image($pdf, $pim, 100, 100, 1);
pdf_execute_image($pdf, $pim, 200, 200, 2);
pdf_close_image($pdf, $pim);
?>
```

pdf_add_annotation

Name

`pdf_add_annotation` — Adds annotation

Description

`void` **`pdf_add_annotation`** `(int` *`pdf document`*`, double` *`llx`*`, double` *`lly`*`, double` *`urx`*`, double` *`ury`*`, string` *`title`*`, string` *`content`*`);`

The `pdf_add_annotation` adds a note with the lower left corner at (*`llx`*, *`lly`*) and the upper right corner at (*`urx`*, *`ury`*).

XVIII. PostgreSQL functions

Postgres, developed originally in the UC Berkeley Computer Science Department, pioneered many of the object-relational concepts now becoming available in some commercial databases. It provides SQL92/SQL3 language support, transaction integrity, and type extensibility. PostgreSQL is a public-domain, open source descendant of this original Berkeley code.

PostgreSQL is available without cost. The current version is available at www.PostgreSQL.org (http://www.postgresql.org/).

Since version 6.3 (03/02/1998) PostgreSQL use unix domain sockets, a table is given to this new possibilities. This socket will be found in `/tmp/.s.PGSQL.5432`. This option can be enabled with the ’-i’ flag to **postmaster** and it’s meaning is: "listen on TCP/IP sockets as well as Unix domain socket".

Table 1. Postmaster and PHP

Postmaster	PHP	Status
postmaster &	pg_connect("", "", "", "", "dbname");	OK
postmaster -i &	pg_connect("", "", "", "", "dbname");	OK
postmaster &	pg_connect("localhost", "", "", "", "dbname");	Unable to connect to PostgreSQL server: connectDB() failed: Is the postmaster running and accepting TCP/IP (with -i) connection at ’localhost’ on port ’5432’? in /path/to/file.php3 on line 20.
postmaster -i &	pg_connect("localhost", "", "", "", "dbname");	OK

One can also establish a connection with the following command: **$conn = pg_Connect("host=localhost port=5432 dbname=chris");**

To use the large object (lo) interface, it is necessary to enclose it within a transaction block. A transaction block starts with a **begin** and if the transaction was valid ends with **commit** and **end**. If the transaction fails the transaction should be closed with **abort** and **rollback**.

Example 1. Using Large Objects

```
<?php
$database = pg_Connect ("", "", "", "", "jacarta");
pg_exec ($database, "begin");
```

```
    $oid = pg_locreate ($database);
    echo ("$oid\n");
    $handle = pg_loopen ($database, $oid, "w");
    echo ("$handle\n");
    pg_lowrite ($handle, "gaga");
    pg_loclose ($handle);
pg_exec ($database, "commit")
pg_exec ($database, "end")
?>
```

pg_Close

Name

`pg_Close` — closes a PostgreSQL connection

Description

`bool` **`pg_close`**`(int` *`connection`*`);`

Returns false if connection is not a valid connection index, true otherwise. Closes down the connection to a PostgreSQL database associated with the given connection index.

pg_cmdTuples

Name

`pg_cmdTuples` — returns number of affected tuples

Description

`int` **`pg_cmdtuples`**`(int` *`result_id`*`);`

pg_cmdTuples() returns the number of tuples (instances) affected by INSERT, UPDATE, and DELETE queries. If no tuple is affected the function will return 0.

Example 1. pg_cmdtuples

```
<?php
$result = pg_exec($conn, "INSERT INTO verlag VALUES (’Autor’)");
$cmdtuples = pg_cmdtuples($result);
echo $cmdtuples . " <- cmdtuples affected.";
?>
```

pg_Connect

Name

`pg_Connect` — opens a connection

Description

```
int pg_connect(string host, string port, string options, string tty, string
dbname);
```

Returns a connection index on success, or false if the connection could not be made. Opens a connection to a PostgreSQL database. Each of the arguments should be a quoted string, including the port number. The options and tty arguments are optional and can be left out. This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple connections open at once.

A connection can also established with the following command: **$conn = pg_connect("dbname=marliese port=5432");** Other parameters besides *`dbname`* and *`port`* are *`host`*, *`tty`*, *`options`*, *`user`* and *`password`*.

See also `pg_pConnect`.

pg_DBname

Name

`pg_DBname` — database name

Description

```
string pg_dbname(int connection);
```

Returns the name of the database that the given PostgreSQL connection index is connected to, or false if connection is not a valid connection index.

pg_ErrorMessage

Name

`pg_ErrorMessage` — error message

Description

```
string pg_errormessage(int connection);
```

Returns a string containing the error message, false on failure. Details about the error probably cannot be retrieved using the pg_errormessage() function if an error occured on the last database action for which a valid connection exists, this function will return a string containing the error message generated by the backend server.

pg_Exec

Name

`pg_Exec` — execute a query

Description

```
int pg_exec(int connection, string query);
```

Returns a result index if query could be executed, false on failure or if connection is not a valid connection index. Details about the error can be retrieved using the `pg_ErrorMessage` function if connection is valid. Sends an SQL statement to the PostgreSQL database specified by the connection index. The connection must be a valid index that was returned by `pg_Connect`. The return value of this function is an index to be used to access the results from other PostgreSQL functions.

Note: PHP/FI returned 1 if the query was not expected to return data (inserts or updates, for example) and greater than 1 even on selects that did not return anything. No such assumption can be made in PHP.

pg_Fetch_Array

Name

`pg_Fetch_Array` — fetch row as array

Description

```
array pg_fetch_array(int result, int row, int [result_type] );
```

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

`pg_fetch_array` is an extended version of `pg_fetch_row`. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

The third optional argument *`result_type`* in `pg_fetch_array` is a constant and can take the following values: PGSQL_ASSOC, PGSQL_NUM, and PGSQL_BOTH.

Note: *`Result_type`* was added in PHP 4.0.

An important thing to note is that using `pg_fetch_array` is NOT significantly slower than using `pg_fetch_row`, while it provides a significant added value.

For further details, also see `pg_fetch_row`

Example 1. PostgreSQL fetch array

```
<?php
$conn = pg_pconnect("","","","","publisher");
if (!$conn) {
    echo "An error occured.\n";
    exit;
}

$result = pg_Exec ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occured.\n";
    exit;
}

$arr = pg_fetch_array ($result, 0);
echo $arr[0] . " <- array\n";

$arr = pg_fetch_array ($result, 1);
echo $arr["author"] . " <- array\n";
?>
```

pg_Fetch_Object

Name

`pg_Fetch_Object` — fetch row as object

Description

`object` **`pg_fetch_object`** `(int` *`result`*`, int` *`row`*`, int` *`[result_type]`* `);`

Returns: An object with properties that correspond to the fetched row, or false if there are no more rows.

`pg_fetch_object` is similar to `pg_fetch_array`, with one difference - an object is returned, instead of an array. Indirectly, that means that you can only access the data by the field names, and not by their offsets (numbers are illegal property names).

The third optional argument *`result_type`* in `pg_fetch_object` is a constant and can take the following values: PGSQL_ASSOC, PGSQL_NUM, and PGSQL_BOTH.

Note: *`Result_type`* was added in PHP 4.0.

Speed-wise, the function is identical to `pg_fetch_array`, and almost as quick as `pg_fetch_row` (the difference is insignificant).

See also: `pg_fetch_array` and `pg_fetch_row`.

Example 1. Postgres fetch object

```
<?php
$database = "verlag";
$db_conn = pg_connect ("localhost", "5432", "", "", $database);
if (!$db_conn): ?>
    <H1>Failed connecting to postgres database <? echo $database ?></H1> <?
    exit;
endif;

$qu = pg_exec ($db_conn, "SELECT * FROM verlag ORDER BY autor");
$row = 0; // postgres needs a row counter other dbs might not

while ($data = pg_fetch_object ($qu, $row)):
    echo $data->autor." (";
    echo $data->jahr ."): ";
    echo $data->titel."<BR>";
    $row++;
endwhile; ?>

<PRE><?php
$fields[] = Array ("autor", "Author");
$fields[] = Array ("jahr",  "  Year");
$fields[] = Array ("titel", " Title");

$row= 0; // postgres needs a row counter other dbs might not
while ($data = pg_fetch_object ($qu, $row)):
    echo "-------\n";
    reset ($fields);
    while (list (,$item) = each ($fields)):
        echo $item[1].": ".$data->$item[0]."\n";
    endwhile;
    $row++;
endwhile;
```

```
echo "-------\n"; ?>
</PRE> <?php
pg_freeResult ($qu);
pg_close ($db_conn);
?>
```

pg_Fetch_Row

Name

`pg_Fetch_Row` — get row as enumerated array

Description

`array` **`pg_fetch_row`**`(int` *`result`*`, int` *`row`*`);`

Returns: An array that corresponds to the fetched row, or false if there are no more rows.

`pg_fetch_row` fetches one row of data from the result associated with the specified result identifier. The row is returned as an array. Each result column is stored in an array offset, starting at offset 0.

Subsequent call to `pg_fetch_row` would return the next row in the result set, or false if there are no more rows.

See also: `pg_fetch_array`, `pg_fetch_object`, `pg_result`.

Example 1. Postgres fetch row

```
<?php
$conn = pg_pconnect("","","","","publisher");
if (!$conn) {
    echo "An error occured.\n";
    exit;
}

$result = pg_Exec ($conn, "SELECT * FROM authors");
if (!$result) {
    echo "An error occured.\n";
    exit;
}

$row = pg_fetch_row ($result, 0);
echo $row[0] . " <- row\n";

$row = pg_fetch_row ($result, 1);
echo $row[0] . " <- row\n";

$row = pg_fetch_row ($result, 2);
echo $row[1] . " <- row\n";
?>
```

pg_FieldIsNull

Name

`pg_FieldIsNull` — Test if a field is NULL

Description

`int` **`pg_fieldisnull`**`(int` *`result_id`*`, int` *`row`*`, mixed` *`field`*`);`

Test if a field is NULL or not. Returns 0 if the field in the given row is not NULL. Returns 1 if the field in the given row is NULL. Field can be specified as number or fieldname. Row numbering starts at 0.

pg_FieldName

Name

`pg_FieldName` — Returns the name of a field

Description

`string` **`pg_fieldname`**`(int` *`result_id`*`, int` *`field_number`*`);`

pg_FieldName() will return the name of the field occupying the given column number in the given PostgreSQL result identifier. Field numbering starts from 0.

pg_FieldNum

Name

`pg_FieldNum` — Returns the number of a column

Description

`int` **`pg_fieldnum`**`(int` *`result_id`*`, string` *`field_name`*`);`

pg_FieldNum() will return the number of the column slot that corresponds to the named field in the given PosgreSQL result identifier. Field numbering starts at 0. This function will return -1 on error.

pg_FieldPrtLen

Name

`pg_FieldPrtLen` — Returns the printed length

Description

`int` **`pg_fieldprtlen`**`(int` *`result_id`*`, int` *`row_number`*`, string` *`field_name`*`);`

pg_FieldPrtLen() will return the actual printed length (number of characters) of a specific value in a PostgreSQL result. Row numbering starts at 0. This function will return -1 on an error.

pg_FieldSize

Name

`pg_FieldSize` — Returns the internal storage size of the named field

Description

`int` **`pg_fieldsize`**`(int` *`result_id`*`, int` *`field_number`*`);`

`pg_FieldSize` will return the internal storage size (in bytes) of the field number in the given PostgreSQL result. Field numbering starts at 0. A field size of -1 indicates a variable length field. This function will return false on error.

pg_FieldType

Name

`pg_FieldType` — Returns the type name for the corresponding field number

Description

`int` **`pg_fieldtype`**`(int` *`result_id`*`, int` *`field_number`*`);`

pg_FieldType() will return a string containing the type name of the given field in the given PostgreSQL result identifier. Field numbering starts at 0.

pg_FreeResult

Name

`pg_FreeResult` — Frees up memory

Description

`int` **`pg_freeresult`**`(int` *`result_id`*`);`

`pg_FreeResult` only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call `pg_FreeResult` with the result identifier as an argument and the associated result memory will be freed.

pg_GetLastOid

Name

`pg_GetLastOid` — Returns the last object identifier

Description

`int` **`pg_getlastoid`**`(int` *`result_id`*`);`

`pg_GetLastOid` can be used to retrieve the Oid assigned to an inserted tuple if the result identifier is used from the last command sent via `pg_Exec` and was an SQL INSERT. This function will return a positive integer if there was a valid Oid. It will return -1 if an error occured or the last command sent via `pg_Exec` was not an INSERT.

pg_Host

Name

`pg_Host` — Returns the host name

Description

`string` **`pg_host`**`(int` *`connection_id`*`);`

pg_Host() will return the host name of the given PostgreSQL connection identifier is connected to.

pg_loclose

Name

`pg_loclose` — close a large object

Description

`void` **`pg_loclose`**`(int` *`fd`*`);`

`pg_loclose` closes an Inversion Large Object. *`fd`* is a file descriptor for the large object from `pg_loopen`.

pg_locreate

Name

`pg_locreate` — create a large object

Description

```
int pg_locreate(int conn);
```

`pg_locreate` creates an Inversion Large Object and returns the oid of the large object. *`conn`* specifies a valid database connection. PostgreSQL access modes INV_READ, INV_WRITE, and INV_ARCHIVE are not supported, the object is created always with both read and write access. INV_ARCHIVE has been removed from PostgreSQL itself (version 6.3 and above).

pg_loopen

Name

`pg_loopen` — open a large object

Description

```
int pg_loopen(int conn, int objoid, string mode);
```

`pg_loopen` open an Inversion Large Object and returns file descriptor of the large object. The file descriptor encapsulates information about the connection. Do not close the connection before closing the large object file descriptor. *`objoid`* specifies a valid large object oid and *`mode`* can be either "r", "w", or "rw".

pg_loread

Name

`pg_loread` — read a large object

Description

```
string pg_loread(int fd, int len);
```

`pg_loread` reads at most *`len`* bytes from a large object and returns it as a string. *`fd`* specifies a valid large object file descriptor and *`len`* specifies the maximum allowable size of the large object segment.

pg_loreadall

Name

`pg_loreadall` — read a entire large object

Description

```
void pg_loreadall(int fd);
```

`pg_loreadall` reads a large object and passes it straight through to the browser after sending all pending headers. Mainly intended for sending binary data like images or sound.

pg_lounlink

Name

`pg_lounlink` — delete a large object

Description

```
void pg_lounlink(int conn, int lobjid);
```

`pg_lounlink` deletes a large object with the *`lobjid`* identifier for that large object.

pg_lowrite

Name

`pg_lowrite` — write a large object

Description

```
int pg_lowrite(int fd, string buf);
```

`pg_lowrite` writes at most to a large object from a variable *`buf`* and returns the number of bytes actually written, or false in the case of an error. *`fd`* is a file descriptor for the large object from `pg_loopen`.

pg_NumFields

Name

`pg_NumFields` — Returns the number of fields

Description

`int` **`pg_numfields`**`(int` *`result_id`*`);`

pg_NumFields() will return the number of fields (columns) in a PostgreSQL result. The argument is a valid result identifier returned by `pg_Exec`. This function will return -1 on error.

pg_NumRows

Name

`pg_NumRows` — Returns the number of rows

Description

`int` **`pg_numrows`**`(int` *`result_id`*`);`

`pg_NumRows` will return the number of rows in a PostgreSQL result. The argument is a valid result identifier returned by `pg_Exec`. This function will return -1 on error.

pg_Options

Name

`pg_Options` — Returns options

Description

`string` **`pg_options`**`(int` *`connection_id`*`);`

pg_Options() will return a string containing the options specified on the given PostgreSQL connection identifier.

pg_pConnect

Name

`pg_pConnect` — make a persistent database connection

Description

`int` **`pg_pconnect`**`(string` *`host`*`, string` *`port`*`, string` *`options`*`, string` *`tty`*`, string` *`dbname`*`);`

Returns a connection index on success, or false if the connection could not be made. Opens a persistent connection to a PostgreSQL database. Each of the arguments should be a quoted string, including the port number. The options and tty arguments are optional and can be left out. This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple persistent connections open at once. See also `pg_Connect`.

A connection can also established with the following command: **$conn = pg_pconnect("dbname=marliese port=5432");** Other parameters besides *`dbname`* and *`port`* are *`host`*, *`tty`*, *`options`*, *`user`* and *`password`*.

pg_Port

Name

`pg_Port` — Returns the port number

Description

`int` **`pg_port`**`(int` *`connection_id`*`);`

pg_Port() will return the port number that the given PostgreSQL connection identifier is connected to.

pg_Result

Name

`pg_Result` — Returns values from a result identifier

Description

`mixed` **`pg_result`**`(int` *`result_id`*`, int` *`row_number`*`, mixed` *`fieldname`*`);`

pg_Result() will return values from a result identifier produced by `pg_Exec`. The *`row_number`* and *`fieldname`* sepcify what cell in the table of results to return. Row numbering starts from 0. Instead of naming the field, you may use the field index as an unquoted number. Field indices start from 0.

PostgreSQL has many built in types and only the basic ones are directly supported here. All forms of integer, boolean and oid types are returned as integer values. All forms of float, and real types are returned as double values. All other types, including arrays are returned as strings formatted in the same default PostgreSQL manner that you would see in the **psql** program.

pg_tty

Name

`pg_tty` — Returns the tty name

Description

```
string pg_tty(int connection_id);
```

pg_tty() will return the tty name that server side debugging output is sent to on the given PostgreSQL connection identifier.

XIX. POSIX functions

Needs to be written ASAP.

posix_kill

Name

`posix_kill` — Send a signal to a process

Description

`bool posix_kill(int pid, int sig);`

Send the signal *`sig`* to the process with the process identifier *`pid`*. Returns FALSE, if unable to send the signal, TRUE otherwise.

See also the kill(2) manual page of your POSIX system, which contains additional information about negative process identifiers, the special pid 0, the special pid -1, and the signal number 0.

posix_getpid

Name

`posix_getpid` — Return the current process identifier

Description

`int posix_getpid(void );`

Return the process identifier of the current process.

posix_getppid

Name

`posix_getppid` — Return the parent process identifier

Description

```
int posix_getppid(void );
```

Return the process identifier of parent process of the current process.

posix_getuid

Name

`posix_getuid` — Return the real user ID of the current process

Description

```
int posix_getuid(void );
```

Return the numeric real user ID of the current process. See also `posix_getpwuid` for information on how to convert this into a useable username.

posix_geteuid

Name

`posix_geteuid` — Return the effective user ID of the current process

Description

```
int posix_geteuid(void );
```

Return the numeric effective user ID of the current process. See also `posix_getpwuid` for information on how to convert this into a useable username.

posix_getgid

Name

`posix_getgid` — Return the real group ID of the current process

Description

```
int posix_getgid(void );
```

Return the numeric real group ID of the current process. See also `posix_getgrgid` for information on how to convert this into a useable group name.

posix_getegid

Name

`posix_getegid` — Return the effective group ID of the current process

Description

```
int posix_getegid(void );
```

Return the numeric effective group ID of the current process. See also `posix_getgrgid` for information on how to convert this into a useable group name.

posix_setuid

Name

`posix_setuid` — Set the effective UID of the current process

Description

```
bool posix_setuid(int uid);
```

Set the real user ID of the current process. This is a privileged function and you need appropriate privileges (usually root) on your system to be able to perform this function.

Returns TRUE on success, FALSE otherwise. See also `posix_setgid`.

posix_setgid

Name

`posix_setgid` — Set the effective GID of the current process

Description

```
bool posix_setgid(int gid);
```

Set the real group ID of the current process. This is a privileged function and you need appropriate privileges (usually root) on your system to be able to perform this function. The appropriate order of function calls is `posix_setgid` first, `posix_setuid` last.

Returns TRUE on success, FALSE otherwise.

posix_getgroups

Name

`posix_getgroups` — Return the group set of the current process

Description

```
array posix_getgroups(void );
```

Returns an array of integers containing the numeric group ids of the group set of the current process. See also `posix_getgrgid` for information on how to convert this into useable group names.

posix_getlogin

Name

`posix_getlogin` — Return login name

Description

```
string posix_getlogin(void );
```

Returns the login name of the user owning the current process. See `posix_getpwnam` for information how to get more information about this user.

posix_getpgrp

Name

`posix_getpgrp` — Return the current process group identifier

Description

```
int posix_getpgrp(void );
```

Return the process group identifier of the current process. See POSIX.1 and the getpgrp(2) manual page on your POSIX system for more information on process groups.

posix_setsid

Name

`posix_setsid` — Make the current process a session leader

Description

```
int posix_setsid(void );
```

Make the current process a session leader. See POSIX.1 and the setsid(2) manual page on your POSIX system for more informations on process groups and job control. Returns the session id.

posix_setpgid

Name

`posix_setpgid` — set process group id for job control

Description

```
int posix_setpgid(int pid, int pgid);
```

Let the process *`pid`* join the process group *`pgid`*. See POSIX.1 and the setsid(2) manual page on your POSIX system for more informations on process groups and job control. Returns TRUE on success, FALSE otherwise.

posix_getpgid

Name

`posix_getpgid` — Get process group id for job control

Description

```
int posix_getpgid(int pid);
```

Returns the process group identifier of the process *`pid`*.

This is not a POSIX function, but is common on BSD and System V systems. If your system does not support this function at system level, this PHP function will always return FALSE.

posix_setsid

Name

`posix_setsid` — Get the current sid of the process

Description

```
int posix_getsid(int pid);
```

Return the sid of the process *`pid`*. If *`pid`* is 0, the sid of the current process is returned.

This is not a POSIX function, but is common on System V systems. If your system does not support this function at system level, this PHP function will always return FALSE.

posix_uname

Name

`posix_uname` — Get system name

Description

```
array posix_uname(void );
```

Returns a hash of strings with information about the system. The indices of the hash are

- sysname - operating system name (e.g. Linux)
- nodename - system name (e.g. valiant)
- release - operating system release (e.g. 2.2.10)
- version - operating system version (e.g. #4 Tue Jul 20 17:01:36 MEST 1999)
- machine - system architecture (e.g. i586)

Posix requires that you must not make any assumptions about the format of the values, e.g. you cannot rely on three digit version numbers or anything else returned by this function.

posix_times

Name

`posix_times` — Get process times

Description

```
array posix_times(void );
```

Returns a hash of strings with information about the current process CPU usage. The indices of the hash are

- ticks - the number of clock ticks that have elapsed since reboot.
- utime - user time used by the current process.
- stime - system time used by the current process.
- cutime - user time used by current process and children.
- cstime - system time used by current process and children.

posix_ctermid

Name

`posix_ctermid` — Get path name of controlling terminal

Description

```
string posix_ctermid(void );
```

Needs to be written.

posix_ttyname

Name

`posix_ttyname` — Determine terminal device name

Description

`string` **`posix_ttyname`**`(int` *`fd`*`);`

Needs to be written.

posix_isatty

Name

`posix_isatty` — Determine if a file descriptor is an interactive terminal

Description

`bool` **`posix_isatty`**`(int` *`fd`*`);`

Needs to be written.

posix_getcwd

Name

`posix_getcwd` — Pathname of current directory

Description

`string` **`posix_getcwd`**`(void );`

Needs to be written ASAP.

posix_mkfifo

Name

`posix_mkfifo` — Create a fifo special file (a named pipe)

Description

`bool` **`posix_getcwd`**`(string` *`pathname`*`, int` *`mode`*`);`

Needs to be written ASAP..

posix_getgrnam

Name

`posix_getgrnam` — Return info about a group by name

Description

`array` **`posix_getgrnam`**`(string` *`name`*`);`

Needs to be written.

posix_getgrgid

Name

`posix_getgrgid` — Return info about a group by group id

Description

```
array posix_getgrgid(int gid);
```

Needs to be written.

posix_getpwnam

Name

`posix_getpwnam` — Return info about a user by name

Description

```
array posix_getpwnam(string name);
```

Needs to be written ASAP.

posix_getpwuid

Name

`posix_getpwuid` — Return info about a user by user id

Description

`array` **`posix_getpwuid`**`(int` *`uid`*`);`

Needs to be written ASAP.

posix_getrlimit

Name

`posix_getrlimit` — Return info about system ressource limits

Description

`array` **`posix_getrlimit`**`(void );`

Needs to be written ASAP.

XX. Regular expression functions

Regular expressions are used for complex string manipulation in PHP. The functions that support regular expressions are:

- `ereg`
- `ereg_replace`
- `eregi`
- `eregi_replace`
- `split`

These functions all take a regular expression string as their first argument. PHP uses the POSIX extended regular expressions as defined by POSIX 1003.2. For a full description of POSIX regular expressions see the regex man pages included in the regex directory in the PHP distribution.

Example 1. Regular expression examples

```
ereg("abc",$string);
/* Returns true if "abc"
   is found anywhere in $string. */

ereg("^abc",$string);
/* Returns true if "abc"
   is found at the beginning of $string. */

ereg("abc$",$string);
/* Returns true if "abc"
   is found at the end of $string. */

eregi("(ozilla.[23]|MSIE.3)",$HTTP_USER_AGENT);
/* Returns true if client browser
   is Netscape 2, 3 or MSIE 3. */

ereg("([[:alnum:]]+) ([[:alnum:]]+) ([[:alnum:]]+)",
     $string,$regs);
/* Places three space separated words
   into $regs[1], $regs[2] and $regs[3]. */

$string = ereg_replace("^","<BR>",$string);
/* Put a <BR> tag at the beginning of $string. */

$string = ereg_replace("$","<BR>",$string);
```

```
/* Put a <BR> tag at the end of $string. */

$string = ereg_replace("\n","",$string);
/* Get rid of any carriage return
   characters in $string. */
```

ereg

Name

`ereg` — regular expression match

Description

`int **ereg**(string *pattern*, string *string*, array *[regs]*);`

Searchs *`string`* for matches to the regular expression given in *`pattern`*.

If matches are found for parenthesized substrings of *`pattern`* and the function is called with the third argument *`regs`*, the matches will be stored in the elements of *`regs`*. $regs[1] will contain the substring which starts at the first left parenthesis; $regs[2] will contain the substring starting at the second, and so on. $regs[0] will contain a copy of *`string`*.

Searching is case sensitive.

Returns true if a match for pattern was found in string, or false if no matches were found or an error occurred.

The following code snippet takes a date in ISO format (YYYY-MM-DD) and prints it in DD.MM.YYYY format:

Example 1. ereg() example

```
if ( ereg( "([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs ) ) {
    echo "$regs[3].$regs[2].$regs[1]";
} else {
    echo "Invalid date format: $date";
}
```

See also `eregi`, `ereg_replace`, and `eregi_replace`.

ereg_replace

Name

`ereg_replace` — replace regular expression

Description

`string **ereg_replace**(string *pattern*, string *replacement*, string *string*);`

This function scans *`string`* for matches to *`pattern`*, then replaces the matched text with *`replacement`*.

The modified string is returned. (Which may mean that the original string is returned if there are no matches to be replaced.)

If *`pattern`* contains parenthesized substrings, *`replacement`* may contain substrings of the form `\\`*`digit`*, which will be replaced by the text matching the digit'th parenthesized substring; `\\0` will produce the entire contents of string. Up to nine substrings may be used. Parentheses may be nested, in which case they are counted by the opening parenthesis.

If no matches are found in *`string`*, then *`string`* will be returned unchanged.

For example, the following code snippet prints "This was a test" three times:

Example 1. ereg_replace() example

```
$string = "This is a test";
echo ereg_replace( " is", " was", $string );
echo ereg_replace( "( )is", "\\1was", $string );
echo ereg_replace( "(( )is)", "\\2was", $string );
```

See also `ereg`, `eregi`, and `eregi_replace`.

eregi

Name

`eregi` — case insensitive regular expression match

Description

```
int eregi(string pattern, string string, array [regs]);
```

This function is identical to `ereg` save that this ignores case distinction when matching alphabetic characters.

See also `ereg`, `ereg_replace`, and `eregi_replace`.

eregi_replace

Name

`eregi_replace` — replace regular expression case insensitive

Description

```
string eregi_replace(string pattern, string replacement, string string);
```

This function is identical to `ereg_replace` save that this ignores case distinction when matching alphabetic characters.

See also `ereg`, `eregi`, and `ereg_replace`.

split

Name

`split` — split string into array by regular expression

Description

```
array split(string pattern, string string, int [limit]);
```

Returns an array of strings, each of which is a substring of string formed by splitting it on boundaries formed by *`pattern`*. If an error occurs, returns false.

To get the first five fields from a line from `/etc/passwd`:

Example 1. split() example

```
$passwd_list = split( ":", $passwd_line, 5 );
```

Note that *`pattern`* is case-sensitive.

See also: `explode` and `implode`.

sql_regcase

Name

`sql_regcase` — make regular expression for case insensitive match

Description

```
string sql_regcase(string string);
```

Returns a valid regular expression which will match `string`, ignoring case. This expression is `string` with each character converted to a bracket expression; this bracket expression contains that character's uppercase and lowercase form if applicable, otherwise it contains the original character twice.

Example 1. sql_regcase() example

```
echo sql_regcase( "Foo bar" );
```

prints

```
[Ff][Oo][Oo][  ][Bb][Aa][Rr]
```

.

This can be used to achieve case insensitive pattern matching in products which support only case sensitive regular expressions.

XXI. Semaphore and shared memory functions

This module provides semaphore functions using System V semaphores. Semaphores may be used to provide exclusive access to resources on the current machine, or to limit the number of processes that may simultaneously use a resource.

This module provides also shared memory functions using System V shared memory. Shared memory may be used to provide access to global variables. Different httpd-daemons and even other programs (such as Perl, C, ...) are able to access this data to provide a global data-exchange. Remember, that shared memory is NOT safe against simultaneous access. Use semaphores for synchronization.

Table 1. Limits of shared memory by the Unix OS

SHMMAX	max size of shared memory, normally 131072 bytes
SHMMIN	minimum size of shared memory, normally 1 byte
SHMMNI	max amount of shared memory segments, normally 100
SHMSEG	max amount of shared memory per process, normally 6

sem_get

Name

`sem_get` — get a semaphore id

Description

`int` **`sem_get`**`(int` *`key`*`, int` *`[max_acquire]`* `, int` *`[perm]`* `);`

Returns: A positive semaphore identifier on success, or false on error.

`sem_get` returns an id that can be used to access the System V semaphore with the given key. The semaphore is created if necessary using the permission bits specified in perm (defaults to 0666). The number of processes that can acquire the semaphore simultaneously is set to max_acquire (defaults to 1). Actually this value is set only if the process finds it is the only process currently attached to the semaphore.

A second call to `sem_get` for the same key will return a different semaphore identifier, but both identifiers access the same underlying semaphore.

See also: `sem_acquire` and `sem_release`.

sem_acquire

Name

`sem_acquire` — acquire a semaphore

Description

`int` **`sem_acquire`**`(int` *`sem_identifier`*`);`

Returns: true on success, false on error

`sem_acquire` blocks (if necessary) until the semaphore can be acquired. A process attempting to acquire a semaphore which it has already acquired will block forever if acquiring the semaphore would cause its max_acquire value to be exceeded.

After processing a request, any semaphores acquired by the process but not explicitly released will be released automatically and a warning will be generated.

See also: `sem_get` and `sem_release`.

sem_release

Name

`sem_release` — release a semaphore

Description

`int` **`sem_release`**`(int` *`sem_identifier`*`);`

Returns: true on success, false on error

`sem_release` releases the semaphore if it is currently acquired by the calling process, otherwise a warning is generated.

After releasing the semaphore, `sem_acquire` may be called to re-acquire it.

See also: `sem_get` and `sem_acquire`.

shm_attach

Name

`shm_attach` — Creates or open a shared memory segment

Description

`int shm_attach(int key, int [memsize], int [perm]);`

`shm_attach` returns an id that that can be used to access the System V shared memory with the given key, the first call creates the shared memory segment with mem_size (default: sysvshm.init_mem in the configuration file, otherwise 10000 bytes) and the optional perm-bits (default: 0666).

A second call to `shm_attach` for the same *`key`* will return a different shared memory identifier, but both identifiers access the same underlying shared memory. *`memsize`* and *`perm`* will be ignored.

shm_detach

Name

`shm_detach` — Disconnects from shared memory segment

Description

`int shm_detach(int shm_identifier);`

`shm_detach` disconnects from the shared memory given by the *`shm_identifier`* created by `shm_attach`. Remember, that shared memory still exist in the Unix system and the data is still present.

shm_remove

Name

`shm_remove` — Removes shared memory from Unix systems

Description

`int` **`shm_remove`**`(int` *`shm_identifier`*`);`

Removes shared memory from Unix systems. All data will be destroyed.

shm_put_var

Name

`shm_put_var` — Inserts or updates a variable in shared memory

Description

`int` **`shm_put_var`**`(int` *`shm_identifier`*`, int` *`variable_key`*`, mixed` *`variable`*`);`

Inserts or updates a *`variable`* with a given *`variable_key`*. All variable-types (double, int, string, array) are supported.

shm_get_var

Name

`shm_get_var` — Returns a variable from shared memory

Description

`mixed` **`shm_get_var`**`(int` *`id`*`, int` *`variable_key`*`);`

`shm_get_var` returns the variable with a given *`variable_key`*. The variable is still present in the shared memory.

shm_remove_var

Name

`shm_remove_var` — Removes a variable from shared memory

Description

```
int shm_remove_var(int id, int variable_key);
```

Removes a variable with a given *`variable_key`* and frees the occupied memory.

XXII. Session handling functions

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

A visitor accessing your web site is assigned an unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.

The session support allows you to register arbitrary numbers of variables to be preserved across requests. When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through `session_start` or implicitly through `session_register`) whether a specific session id has been sent with the request. If this is the case, the prior saved environment is recreated.

All registered variables are serialized after the request finishes. Registered variables which are undefined are marked as being not defined. On subsequent accesses, these are not defined by the session module unless the user defines them later.

There are two methods to propagate a session id:

- Cookies
- URL parameter

The session module supports both methods. Cookies are optimal, but since they are not reliable (clients are not bound to accept them), we cannot rely on them. The second method embeds the session id directly into URLs.

PHP is capable of doing this transparently when compiled with `-enable-trans-sid`. If you enable this option, relative URIs will be changed to contain the session id automatically. Alternatively, you can use the constant `SID` which is defined, if the client did not send the appropiate cookie. `SID` is either of the form `session_name=session_id` or is an empty string.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

Example 1. Counting the number of hits of a single user

```
<?php

session_register("count");

$count++;

?>
```

```
Hello visitor, you have seen this page <? echo $count; ?> times.<p>

<?
# the <?=SID?> is necessary to preserve the session id
# in the case that the user has disabled cookies
?>

 To continue, <A HREF="nextpage.php?<?=SID?>">click here</A>
```

To implement database storage you need PHP code and a user level function `session_set_save_handler`. You would have to extend the following functions to cover MySQL or another database.

Example 2. Usage of `session_set_save_handler`

```
<?php

function open ($save_path, $session_name) {
    echo "open ($save_path, $session_name)\n";
    return true;
}

function close () {
    echo "close\n";
    return true;
}

function read ($key) {
    echo "write ($key, $val)\n";
    return "foo|i:1;";
}

function write ($key, $val) {
    echo "write ($key, $val)\n";
    return true;
}

function destroy ($key)
    return true;
}

function gc ($maxlifetime) {
    return true;
```

```
}

session_set_save_handler ("open", "close", "read", "write", "destroy", "gc");

session_start ();

$foo++;

?>
```

Will produce this results:

```
$ ./php save_handler.php
Content-Type: text/html
Set-cookie: PHPSESSID=f08b925af0ecb52bdd2de97d95cdbe6b

open (/tmp, PHPSESSID)
read (f08b925af0ecb52bdd2de97d95cdbe6b)
write (f08b925af0ecb52bdd2de97d95cdbe6b, foo|i:2;)
close
```

The <`?=SID?`> is not necessary, if `-enable-trans-sid` was used to compile PHP.

The session management system supports a number of configuration options which you can place in your php.ini file. We will give a short overview.

- `session.save_handler` defines the name of the handler which is used for storing and retrieving data associated with a session. Defaults to `files`.
- `session.save_path` defines the argument which is passed to the save handler. If you choose the default files handler, this is the path where the files are created. Defaults to `/tmp`.
- `session.name` specifies the name of the session which is used as cookie name. It should only contain alphanumeric characters. Defaults to `PHPSESSID`.
- `session.auto_start` specifies whether the session module start a session automatically on request startup. Defaults to `0` (disabled).
- `session.lifetime` specifies the lifetime of the cookie in seconds which is sent to the browser. The value 0 means "until the browser is closed." Defaults to `0`.

- `session.serialize_handler` defines the name of the handler which is used to serialize/deserialize data. Currently, a PHP internal format (name `php`) and WDDX is supported (name `wddx`). WDDX is only available, if PHP is compiled with WDDX support. Defaults to `php`.
- `session.gc_probability` specifies the probability that the gc (garbage collection) routine is started on each request in percent. Defaults to `1`.
- `session.gc_maxlifetime` specifies the number of seconds after which data will be seen as ’garbage’ and cleaned up.
- `session.extern_referer_check` determines whether session ids referred to by external sites will be eliminated. If session ids are propagated using the URL method, users not knowing about the impact might publish session ids. This can lead to security problems which this check tries to defeat. Defaults to `0`.
- `session.entropy_file` gives a path to an external resource (file) which will be used as an additional entropy source in the session id creation process. Examples are `/dev/random` or `/dev/urandom` which are available on many Unix systems.
- `session.entropy_length` specifies the number of bytes which will be read from the file specified above. Defaults to `0` (disabled).
- `session.use_cookies` specifies whether the module will use cookies to store the session id on the client side. Defaults to `1` (enabled).

Note: Session handling was added in PHP 4.0.

session_start

Name

`session_start` — Initialize session data

Description

```
bool session_start(void);
```

`session_start` creates a session (or resumes the current one based on the session id being passed via a GET variable or a cookie).

This function always returns true.

> **Note:** This function was added in PHP 4.0.

session_destroy

Name

`session_destroy` — Destroys all data registered to a session

Description

```
bool session_destroy(void);
```

`session_destroy` destroys all of the data associated with the current session.

This function always returns true.

> **Note:** This function was added in PHP 4.0.

session_name

Name

`session_name` — Get and/or set the current session name

Description

```
string session_name(string [name]);
```

`session_name` returns the name of the current session. If *`name`* is specified, the name of the current session is changed to its value.

Example 1. `session_name` examples

```
$username="foo";

if(isset($username)) {
    session_name($username);
}

echo "Your username is " . session_name();
```

> **Note:** This function was added in PHP 4.0.

session_module_name

Name

`session_module_name` — Get and/or set the current session module

Description

`string` **`session_module_name`**`(string` *`[module]`*`);`

`session_module_name` returns the name of the current session module. If *`module`* is specified, that module will be used instead.

Note: This function was added in PHP 4.0.

session_save_path

Name

`session_save_path` — Get and/or set the current session save path

Description

`string` **`session_save_path`**`(string` *`[path]`*`);`

`session_save_path` returns the path of the current directory used to save session data. If *`path`* is specified, the path to which data is saved will be changed.

Note: On some operating systems, you may want to specify a path on a filesystem that handles lots of small files efficiently. For example, on Linux, reiserfs may provide better performance than ext2fs.

Note: This function was added in PHP 4.0.

session_id

Name

`session_id` — Get and/or set the current session id

Description

```
string session_id(string [id]);
```

`session_id` returns the session id for the current session. If *`id`* is specified, it will replace the current session id.

directory used to save session data. If *`path`* is specified, the path to which data is saved will be changed.

> The constant SID can also be used to retrieve the current name and session id as a string suitable for adding to URLs.
>
> **Note:** This function was added in PHP 4.0.

session_register

Name

`session_register` — Register a variable with the current session

Description

```
bool session_register(string name);
```

`session_register` registers the global variable named *`name`* with the current session.

> This function returns true when the variable is successfully registered with the session.

Note: This function was added in PHP 4.0.

session_unregister

Name

`session_unregister` — Unregister a variable from the current session

Description

`bool` **`session_unregister`**`(string` *`name`*`);`

`session_unregister` unregisters (forgets) the global variable named *`name`* from the current session.

This function returns true when the variable is successfully unregistered from the session.

Note: This function was added in PHP 4.0.

session_is_registered

Name

`session_is_registered` — Find out if a variable is registered in a session

Description

`bool` **`session_is_registered`**`(string` *`name`*`);`

`session_is_registered` returns true if there is a variable with the name *`name`* registered in the current session.

Note: This function was added in PHP 4.0.

session_decode

Name

`session_decode` — Decodes session data from a string

Description

`bool session_decode(string data);`

`session_decode` decodes the session data in *`data`*, setting variables stored in the session.

Note: This function was added in PHP 4.0.

session_encode

Name

`session_encode` — Encodes the current session data as a string

Description

`bool session_encode(void);`

`session_encode` returns a string with the contents of the current session encoded within.

Note: This function was added in PHP 4.0.

XXIII. Solid functions

The Solid functions are deprecated, you probably want to use the Unified ODBC functions instead.

solid_close

Name

`solid_close` — close a Solid connection

Description

See `odbc_close`.

solid_connect

Name

`solid_connect` — connect to a Solid data source

Description

See `odbc_connect`.

solid_exec

Name

`solid_exec` — execute a Solid query

Description

See `odbc_exec`.

solid_fetchrow

Name

`solid_fetchrow` — fetch row of data from Solid query

Descriptio

See `odbc_fetch_row`

solid_fieldname

Name

`solid_fieldname` — get name of column from Solid query

Description

See `odbc_field_name`.

solid_fieldnum

Name

`solid_fieldnum` — get index of column from Solid query

Description

See `odbc_field_num`.

solid_freeresult

Name

`solid_freeresult` — free result memory from Solid query

Description

See `odbc_free_result`.

solid_numfields

Name

`solid_numfields` — get number of fields in Solid result

Description

See `odbc_num_fields`.

solid_numrows

Name

`solid_numrows` — get number of rows in Solid result

Description

See `odbc_num_rows`.

solid_result

Name

`solid_result` — get data from Solid results

Description

See `odbc_result`.

XXIV. SNMP functions

In order to use the SNMP functions on Unix you need to install the UCD SNMP (http://ucd-snmp.ucdavis.edu/) package. On Windows these functions are only available on NT and not on Win95/98.

Important: In order to use the UCD SNMP package, you need to define NO_ZEROLENGTH_COMMUNITY to 1 before compiling it. After configuring UCD SNMP, edit config.h and search for NO_ZEROLENGTH_COMMUNITY. Uncomment the #define line. It should look like this afterwards:

```
#define NO_ZEROLENGTH_COMMUNITY 1
```

If you see strange segmentation faults in combination with SNMP commands, you did not follow the above instructions. If you do not want to recompile UCD SNMP, you can compile PHP with the –enable-ucd-snmp-hack switch which will work around the misfeature.

snmpget

Name

`snmpget` — Fetch an SNMP object

Description

`string` **`snmpget`**`(string` *`hostname`*`, string` *`community`*`, string` *`object_id`*`, int` *`[timeout]`*`, int` *`[retries]`*`);`

Returns SNMP object value on success and false on error.

The `snmpget` function is used to read the value of an SNMP object specified by the *`object_id`*. SNMP agent is specified by the *`hostname`* and the read community is specified by the *`community`* parameter.

```
$syscontact = snmpget("127.0.0.1", "public", "system.SysContact.0")
```

snmpset

Name

`snmpset` — Set an SNMP object

Description

`string` **`snmpget`**`(string` *`hostname`*`, string` *`community`*`, string` *`object_id`*`, string` *`type`*`, mixed` *`value`*`, int` *`[timeout]`*`, int` *`[retries]`*`);`

Sets the specified SNMP object value, returning true on success and false on error.

The `snmpset` function is used to set the value of an SNMP object specified by the *`object_id`*. SNMP agent is specified by the *`hostname`* and the read community is specified by the *`community`* parameter.

snmpwalk

Name

`snmpwalk` — Fetch all the SNMP objects from an agent

Description

```
array snmpwalk(string hostname, string community, string object_id, int
[timeout] , int [retries] );
```

Returns an array of SNMP object values starting from the `object_id` as root and false on error.

`snmpwalk` function is used to read all the values from an SNMP agent specified by the *`hostname`*. *`Community`* specifies the read community for that agent. A null *`object_id`* is taken as the root of the SNMP objects tree and all objects under that tree are returned as an array. If *`object_id`* is specified, all the SNMP objects below that *`object_id`* are returned.

```
$a = snmpwalk("127.0.0.1", "public", "");
```

Above function call would return all the SNMP objects from the SNMP agent running on localhost. One can step through the values with a loop

```
for ($i=0; $i<count($a); $i++) {
    echo $a[$i];
}
```

snmpwalkoid

Name

`snmpwalkoid` — Query for a tree of information about a network entity

Description

```
array snmpwalkoid(string hostname, string community, string object_id, int
[timeout] , int [retries] );
```

Returns an associative array with object ids and their respective object value starting from the *`object_id`* as root and false on error.

`snmpwalkoid` function is used to read all object ids and their respective values from an SNMP agent specified by the hostname. Community specifies the read *`community`* for that agent. A null *`object_id`* is taken as the root of the SNMP objects tree and all objects under that tree are returned as an array. If *`object_id`* is specified, all the SNMP objects below that *`object_id`* are returned.

The existence of `snmpwalkoid` and `snmpwalk` has historical reasons. Both functions are provided for backward compatibility.

```
$a = snmpwalkoid("127.0.0.1", "public", "");
```

Above function call would return all the SNMP objects from the SNMP agent running on localhost. One can step through the values with a loop

```
for (reset($a); $i = key($a); next($a)) {
    echo "$i: $a[$i]<br>\n";
}
```

snmp_get_quick_print

Name

`snmp_get_quick_print` — Fetch the current value of the UCD library’s quick_print setting

Description

`boolean` **`snmp_get_quick_print`** `(void );`

Returns the current value stored in the UCD Library for quick_print. quick_print is off by default.

```
$quickprint = snmp_get_quick_print();
```

Above function call would return false if quick_print is on, and true if quick_print is on.

`snmp_get_quick_print` is only available when using the UCD SNMP library. This function is not available when using the Windows SNMP library.

See: `snmp_set_quick_print` for a full description of what quick_print does.

snmp_set_quick_print

Name

`snmp_set_quick_print` — Set the value of quick_print within the UCD SNMP library.

Description

`void` **`snmp_set_quick_print`** `(boolean` *`quick_print`*`);`

Sets the value of quick_print within the UCD SNMP library. When this is set (1), the SNMP library will return ’quick printed’ values. This means that just the value will be printed. When quick_print is not enabled (default) the UCD SNMP library prints extra information including the type of the value (i.e. IpAddress or OID). Additionally, if quick_print is not enabled, the library prints additional hex values for all strings of three characters or less.

Setting quick_print is often used when using the information returned rather then displaying it.

```
snmp_set_quick_print(0);
$a = snmpget("127.0.0.1", "public", ".1.3.6.1.2.1.2.2.1.9.1");
echo "$a<BR>\n";
snmp_set_quick_print(1);
$a = snmpget("127.0.0.1", "public", ".1.3.6.1.2.1.2.2.1.9.1");
echo "$a<BR>\n";
```

The first value printed might be: ’Timeticks: (0) 0:00:00.00’, whereas with quick_print enabled, just ’0:00:00.00’ would be printed.

By default the UCD SNMP library returns verbose values, quick_print is used to return only the value.

Currently strings are still returned with extra quotes, this will be corrected in a later release.

`snmp_set_quick_print` is only available when using the UCD SNMP library. This function is not available when using the Windows SNMP library.

XXV. String functions

These functions all manipulate strings in various ways. Some more specialized sections can be found in the regular expression and URL handling sections.

AddCSlashes

Name

`AddCSlashes` — Quote string with slashes in a C style

Description

```
string addcslashes(string str, string charlist);
```

Returns a string with backslashes before characters that are listed in *`charlist`* parameter. It escapes `\n`, `\r` etc. in C-like style, characters with ASCII code lower than 32 and higher than 126 are converted to octal representation. Be carefull when escaping alphanumeric characters. You can specify a range in *`charlist`* like "\0..\37", which would escape all characters with ASCII code between 0 and 31.

Example 1. addcslashes() example

```
$escaped = addcslashes($not_escaped, "\0..\37!@\177..\377");
```

> **Note:** Added in PHP4b3-dev.

See also `stripcslashes`, `stripslashes`, `htmlspecialchars`, `htmlspecialchars`, and `quotemeta`.

AddSlashes

Name

`AddSlashes` — Quote string with slashes

Description

`string **addslashes**(string *str*);`

Returns a string with backslashes before characters that need to be quoted in database queries etc. These characters are single quote (`'`), double quote (`"`), backslash (`\`) and NUL (the null byte).

See also `stripslashes`, `htmlspecialchars`, and `quotemeta`.

bin2hex

Name

`bin2hex` — Convert binary data into hexadecimal representation

Description

`string **bin2hex**(string *str*);`

Returns an ASCII string containing the hexadecimal representation of *`str`*. The conversion is done byte-wise with the high-nibble first.

Chop

Name

`Chop` — remove trailing whitespace

Description

```
string chop(string str);
```

Returns the argument string without trailing whitespace.

Example 1. chop() example

```
$trimmed = Chop($line);
```

See also `trim`.

Chr

Name

`Chr` — Return a specific character

Description

```
string chr(int ascii);
```

Returns a one-character string containing the character specified by *`ascii`*.

Example 1. `chr` example

```
$str .= chr(27); /* add an escape character at the end of $str */

/* Often this is more useful */
$str = sprintf("The string ends in escape: %c", 27);
```

This function complements `ord`. See also `sprintf` with a format string of `%c`.

chunk_split

Name

`chunk_split` — Split a string into smaller chunks

Description

```
string chunk_split(string string, int [chunklen] , string [end] );
```

Can be used to split a string into smaller chunks which is useful for e.g. converting base64_encode output to match RFC 2045 semantics. It inserts every *`chunklen`* (defaults to 76) chars the string *`end`* (defaults to "\r\n"). It returns the new string leaving the original string untouched.

Example 1. `chunk_split` example

```
# format $data using RFC 2045 semantics

$new_string = chunk_split(base64_encode($data));
```

This function is significantly faster than `ereg_replace`.

Note: This function was added in 3.0.6.

convert_cyr_string

Name

`convert_cyr_string` — Convert from one Cyrillic character set to another

Description

```
string convert_cyr_string(string str, string from, string to);
```

This function converts the given string from one Cyrillic character set to another. The *`from`* and *`to`* arguments are single characters that represent the source and target Cyrillic character sets. The supported types are:

- k - koi8-r
- w - windows-1251
- i - iso8859-5
- a - x-cp866
- d - x-cp866
- m - x-mac-cyrillic

crypt

Name

`crypt` — DES-encrypt a string

Description

```
string crypt(string str, string [salt] );
```

`crypt` will encrypt a string using the standard Unix DES encryption method. Arguments are a string to be encrypted and an optional two-character salt string to base the encryption on. See the Unix man page for your crypt function for more information.

If the salt argument is not provided, it will be randomly generated by PHP.

Some operating systems support more than one type of encryption. In fact, sometimes the standard DES encryption is replaced by an MD5 based encryption algorithm. The encryption type is triggered by the salt argument. At install time, PHP determines the capabilities of the crypt function and will accept salts for other encryption types. If no salt is provided, PHP will auto-generate a standard 2-character DES salt by default unless the default encryption type on the system is MD5 in which case a random MD5-compatible salt is generated. PHP sets a constant named CRYPT_SALT_LENGTH which tells you whether a regular 2-character salt applies to your system or the longer 12-char MD5 salt is applicable.

The standard DES encryption `crypt` contains the salt as the first two characters of the output.

On systems where the crypt() function supports multiple encryption types, the following constants are set to 0 or 1 depending on whether the given type is available:

- CRYPT_STD_DES - Standard DES encryption with a 2-char SALT
- CRYPT_EXT_DES - Extended DES encryption with a 9-char SALT
- CRYPT_MD5 - MD5 encryption with a 12-char SALT starting with $1$
- CRYPT_BLOWFISH - Extended DES encryption with a 16-char SALT starting with $2$

There is no decrypt function, since `crypt` uses a one-way algorithm.

echo

Name

`echo` — Output one or more strings

Description

```
echo(string arg1, string [argn]... );
```

Outputs all parameters.

`echo` is not actually a function (it is a language construct) so you are not required to use parantheses with it.

Example 1. echo example

```
echo "Hello World";
```

> **Note:** In fact, if you want to pass more than one parameter to echo, you must not enclose the parameters within parentheses.

See also: `print`, `printf`, and `flush`.

explode

Name

`explode` — Split a string by string

Description

```
array explode(string separator, string string);
```

Returns an array of strings containing the elements separated by *`separator`*.

Example 1. `explode` example

```
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
```

See also `split` and `implode`.

flush

Name

`flush` — Flush the output buffer

Description

`void` **`flush`**`(void);`

Flushes the output buffers of PHP and whatever backend PHP is using (CGI, a web server, etc.) This effectively tries to push all the output so far to the user's browser.

get_meta_tags

Name

`get_meta_tags` — Extracts all meta tag content attributes from a file and returns an array

Description

`array` **`get_meta_tags`**`(string` *`filename`*`, int` *`[use_include_path]`* `);`

Opens *`filename`* and parses it line by line for <meta> tags of the form

Example 1. Meta Tags Example

```
<meta name="author" content="name">
<meta name="tags" content="php3 documentation">
</head> <!- parsing stops here ->
```

(pay attention to line endings - PHP uses a native function to parse the input, so a Mac file won't work on Unix).

The value of the name property becomes the key, the value of the content property becomes the value of the returned array, so you can easily use standard array functions to traverse it or access single values. Special characters in the value of the name property are substituted with '_', the rest is converted to lower case.

Setting *`use_include_path`* to 1 will result in PHP trying to open the file along the standard include path.

htmlentities

Name

`htmlentities` — Convert all applicable characters to HTML entities

Description

```
string htmlentities(string string);
```

This function is identical to `htmlspecialchars` in all ways, except that all characters which have HTML entity equivalents are translated into these entities.

At present, the ISO-8859-1 character set is used.

See also `htmlspecialchars` and `nl2br`.

htmlspecialchars

Name

`htmlspecialchars` — Convert special characters to HTML entities

Description

```
string htmlspecialchars(string string);
```

Certain characters have special significance in HTML, and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with these conversions made.

This function is useful in preventing user-supplied text from containing HTML markup, such as in a message board or guest book application.

At present, the translations that are done are:

- ’&’ (ampersand) becomes ’&amp;’
- ’"’ (double quote) becomes ’&quot;’
- ’<’ (less than) becomes ’&lt;’
- ’>’ (greater than) becomes ’&gt;’

Note that this functions does not translate anything beyond what is listed above. For full entity translation, see `htmlentities`.

See also `htmlentities` and `nl2br`.

implode

Name

`implode` — Join array elements with a string

Description

```
string implode(string glue, array pieces);
```

Returns a string containing a string representation of all the array elements in the same order, with the glue string between each element.

Example 1. implode() example

```
$colon_separated = implode(":", $array);
```

See also `explode`, `join`, and `split`.

join

Name

`join` — Join array elements with a string

Description

```
string join(string glue, array pieces);
```

`join` is an alias to `implode`, and is identical in every way.

ltrim

Name

`ltrim` — Strip whitespace from the beginning of a string

Description

```
string ltrim(string str);
```

This function strips whitespace from the start of a string and returns the stripped string.

See also `chop` and `trim`.

md5

Name

`md5` — Calculate the md5 hash of a string

Description

```
string md5(string str);
```

Calculates the MD5 hash of *`str`* using the RSA Data Security, Inc. MD5 Message-Digest Algorithm (http://ds.internic.net/rfc/rfc1321.txt).

Metaphone

Name

`Metaphone` — Calculate the metaphone key of a string

Description

```
string metaphone(string str);
```

Calculates the metaphone key of *`str`*.

Similar to `soundex` metaphone creates the same key for similar sounding words. It's more accurate than `soundex` as it knows the basic rules of English pronunciation. the metaphone generated keys are of variable length.

> Metaphone was developed by Lawrence Philips <lphilips@verity.com>. It is described in ["Practical Algorithms for Programmers", Binstock & Rex, Addison Wesley, 1995].

> **Note:** This function was added in PHP 4.0.

nl2br

Name

`nl2br` — Converts newlines to HTML line breaks.

Description

```
string nl2br(string string);
```

Returns *`string`* with '<BR>' inserted before all newlines.

See also `htmlspecialchars` and `htmlentities`.

Ord

Name

`Ord` — Return ASCII value of character

Description

```
int ord(string string);
```

Returns the ASCII value of the first character of *`string`*. This function complements `chr`.

Example 1. `ord` example

```
if (ord($str) == 10) {
    echo "The first character of \$str is a line feed.\n";
}
```

See also `chr`.

parse_str

Name

`parse_str` — Parses the string into variables

Description

`void` **`parse_str`**`(string` *`str`*`);`

Parses *`str`* as if it were the query string passed via an URL and sets variables in the current scope.

Example 1. Using `parse_str`

```
$str = "first=value&second[]=this+works&second[]=another";
parse_str($str);
echo $first; /* prints "value" */
echo $second[0]; /* prints "this works" */
echo $second[1]; /* prints "another" */
```

print

Name

`print` — Output a string

Description

`print``(string` *`arg`*`);`

Outputs *`arg`*.

See also: `echo` `printf` `flush`

printf

Name

`printf` — output a formatted string

Description

`int **printf**(string *format*, mixed *[args]*... );`

Produces output according to *`format`*, which is described in the documentation for `sprintf`.

See also: `print`, `sprintf`, and `flush`.

quoted_printable_decode

Name

`quoted_printable_decode` — Convert a quoted-printable string to an 8 bit string

Description

`string **quoted_printable_decode** (string *str*);`

This function returns an 8-bit binary string corresponding to the decoded quoted printable string. This function is similar to `imap_qprint`, except this one does not require the IMAP module to work.

QuoteMeta

Name

`QuoteMeta` — quote meta characters

Description

```
string quotemeta(string str);
```

Returns a version of str with a backslash character (\) before every character that is among these:

```
. \\ + * ? [ ^ ] ( $ )
```

See also `addslashes`, `htmlentities`, `htmlspecialchars`, `nl2br`, and `stripslashes`.

rawurldecode

Name

`rawurldecode` — Decode URL-encoded strings

Description

```
string rawurldecode(string str);
```

Returns a string in which the sequences with percent (%) signs followed by two hex digits have been replaced with literal characters. For example, the string

```
foo%20bar%40baz
```

decodes into

```
foo
     bar@baz
```

See also `rawurlencode`.

rawurlencode

Name

`rawurlencode` — URL-encode according to RFC1738

Description

```
string rawurlencode(string str);
```

Returns a string in which all non-alphanumeric characters except

`-_.`

have been replaced with a percent (`%`) sign followed by two hex digits. This is the encoding described in RFC1738 for protecting literal characters from being interpreted as special URL delimiters, and for protecting URL's from being mangled by transmission media with character conversions (like some email systems). For example, if you want to include a password in an ftp url:

Example 1. `rawurlencode` example 1

```
echo '<A HREF="ftp://user:', rawurlencode ('foo @+%/'),
     '@ftp.my.com/x.txt">';
```

Or, if you pass information in a path info component of the url:

Example 2. `rawurlencode` example 2

```
echo '<A HREF="http://x.com/department_list_script/',
     rawurlencode ('sales and marketing/Miami'), '">';
```

See also `rawurldecode`.

setlocale

Name

`setlocale` — Set locale information

Description

`string **setlocale**(string *category*, string *locale*);`

`category` is a string specifying the category of the functions affected by the locale setting:

- LC_ALL for all of the below
- LC_COLLATE for string comparison - not currently implemented in PHP
- LC_CTYPE for character classification and conversion, for example `strtoupper`
- LC_MONETARY for localeconv() - not currently implemented in PHP
- LC_NUMERIC for decimal separator
- LC_TIME for date and time formatting with `strftime`

If *`locale`* is the empty string `""`, the locale names will be set from the values of environment variables with the same names as the above categories, or from "LANG".

If locale is zero or `"0"`, the locale setting is not affected, only the current setting is returned.

Setlocale returns the new current locale, or false if the locale functionality is not implemented in the plattform, the specified locale does not exist or the category name is invalid. An invalid category name also causes a warning message.

similar_text

Name

`similar_text` — Calculate the similarity between two strings

Description

`int` **`similar_text`**`(string` *`first`*`, string` *`second`*`, double` *`[percent]`* `);`

This calculates the similarity between two strings as described in Oliver [1993]. Note that this implementation does not use a stack as in Oliver's pseudo code, but recursive calls which may or may not speed up the whole process. Note also that the complexity of this algorithm is O(N**3) where N is the length of the longest string.

By passing a reference as third argument, `similar_text` will calculate the similarity in percent for you. It returns the number of matching chars in both strings.

soundex

Name

`soundex` — Calculate the soundex key of a string

Description

`string` **`soundex`**`(string` *`str`*`);`

Calculates the soundex key of *`str`*.

Soundex keys have the property that words pronounced similarly produce the same soundex key, and can thus be used to simplify searches in databases where you know the pronunciation but not the spelling. This soundex function returns a string 4 characters long, starting with a letter.

This particular soundex function is one described by Donald Knuth in "The Art Of Computer Programming, vol. 3: Sorting And Searching", Addison-Wesley (1973), pp. 391-392.

Example 1. Soundex Examples

```
soundex("Euler") == soundex("Ellery") == 'E460';
soundex("Gauss") == soundex("Ghosh") == 'G200';
soundex("Knuth") == soundex("Kant") == 'H416';
soundex("Lloyd") == soundex("Ladd") == 'L300';
soundex("Lukasiewicz") == soundex("Lissajous") == 'L222';
```

sprintf

Name

`sprintf` — Return a formatted string

Description

```
string sprintf(string format, mixed [args]... );
```

Returns a string produced according to the formatting string *`format`*.

The format string is composed by zero or more directives: ordinary characters (excluding `%`) that are copied directly to the result, and *conversion specifications*, each of which results in fetching its own parameter. This applies to both `sprintf` and `printf`.

Each conversion specification consists of these elements, in order:

1. An optional *padding specifier* that says what character will be used for padding the results to the right string size. This may be a space character or a `0` (zero character). The default is to pad with spaces. An alternate padding character can be specified by prefixing it with a single quote (`'`). See the examples below.

2. An optional *alignment specifier* that says if the result should be left-justified or right-justified. The default is right-justified; a `-` character here will make it left-justified.
3. An optional number, a *width specifier* that says how many characters (minimum) this conversion should result in.
4. An optional *precision specifier* that says how many decimal digits should be displayed for floating-point numbers. This option has no effect for other types than double. (Another function useful for formatting numbers is `number_format`.)
5. A *type specifier* that says what type the argument data should be treated as. Possible types:

   ```
   % - a literal percent character. No argument is required.
   b - the argument is treated as an integer, and presented as a binary number.
   c - the argument is treated as an integer, and presented as the character with that ASC
   d - the argument is treated as an integer, and presented as a decimal number.
   f - the argument is treated as a double, and presented as a floating-point number.
   o - the argument is treated as an integer, and presented as an octal number.
   s - the argument is treated as and presented as a string.
   x - the argument is treated as an integer and presented as a hexadecimal number (with l
   X - the argument is treated as an integer and presented as a hexadecimal number (with u
   ```

See also: `printf` and `number_format`.

Examples

Example 1. sprintf: zero-padded integers

```
$isodate = sprintf("%04d-%02d-%02d", $year, $month, $day);
```

Example 2. sprintf: formatting currency

```
$money1 = 68.75;
$money2 = 54.35;
$money = $money1 + $money2;
// echo $money will output "123.1";
$formatted = sprintf ("%01.2f", $money);
// echo $formatted will output "123.10"
```

strcasecmp

Name

`strcasecmp` — Binary safe case-insensitive string comparison

Description

`int` **`strcasecmp`**`(string` *`str1`*`, string` *`str2`*`);`

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

See also `ereg`, `strcmp`, `substr`, `stristr`, and `strstr`.

strchr

Name

`strchr` — Find the first occurrence of a character.

Description

`string` **`strchr`**`(string` *`haystack`*`, string` *`needle`*`);`

This function is an alias for `strstr`, and is identical in every way.

strcmp

Name

`strcmp` — Binary safe string comparison

Description

```
int strcmp(string str1, string str2);
```

Returns < 0 if *`str1`* is less than *`str2`*; > 0 if *`str1`* is greater than *`str2`*, and 0 if they are equal.

Note that this comparison is case sensitive.

See also `ereg`, `strcasecmp`, `substr`, `stristr`, and `strstr`.

strcspn

Name

`strcspn` — Find length of initial segment not matching mask

Description

```
int strcspn(string str1, string str2);
```

Returns the length of the initial segment of *`str1`* which does *not* contain any of the characters in *`str2`*.

See also `strspn`.

strip_tags

Name

`strip_tags` — Strip HTML and PHP tags from a string

Description

```
string strip_tags(string str, string [allowable_tags]);
```

This function tries to strip all HTML and PHP tags from the given string. It errors on the side of caution in case of incomplete or bogus tags. It uses the same tag stripping state machine as the `fgetss` function.

You can use the optional second parameter to specify tags which should not be stripped.

Note: *`allowable_tags`* was added in PHP 3.0.13, PHP4B3.

StripCSlashes

Name

`StripCSlashes` — un-quote string quoted with addcslashes

Description

```
string stripcslashes(string str);
```

Returns a string with backslashes stripped off. Recognizes C-like `\n`, `\r` ..., octal and hexadecimal representation.

Note: Added in PHP4b3-dev.

See also `addcslashes`.

StripSlashes

Name

`StripSlashes` — Un-quote string quoted with addslashes

Description

```
string stripslashes(string str);
```

Returns a string with backslashes stripped off. (`\'` becomes `'` and so on.) Double backslashes are made into a single backslash.

See also `addslashes`.

stristr

Name

`stristr` — Case-insensitive `strstr`

Description

```
string stristr(string haystack, string needle);
```

Returns all of *`haystack`* from the first occurrence of *`needle`* to the end. *`needle`* and *`haystack`* are examined in a case-insensitive manner.

If *`needle`* is not found, returns false.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

See also `strchr`, `strrchr`, `substr`, and `ereg`.

strlen

Name

`strlen` — Get string length

Description

```
int strlen(string str);
```

Returns the length of *`string`*.

strpos

Name

`strpos` — Find position of first occurrence of a string

Description

```
int strpos(string haystack, string needle, int [offset] );
```

Returns the numeric position of the first occurrence of *`needle`* in the *`haystack`* string. Unlike the `strrpos`, this function can take a full string as the *`needle`* parameter and the entire string will be used.

If *`needle`* is not found, returns false.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

The optional *`offset`* parameter allows you to specify which character in *`haystack`* to start searching. The position returned is still relative to the the beginning of *`haystack`*.

See also `strrpos`, `strrchr`, `substr`, `stristr`, and `strstr`.

strrchr

Name

`strrchr` — Find the last occurrence of a character in a string

Description

```
string strrchr(string haystack, string needle);
```

This function returns the portion of *`haystack`* which starts at the last occurrence of *`needle`* and goes until the end of *`haystack`*.

Returns false if *`needle`* is not found.

If *`needle`* contains more than one character, the first is used.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

Example 1. `strrchr` example

```
// get last directory in $PATH
$dir = substr( strrchr( $PATH, ":" ), 1 );

// get everything after last newline
$text = "Line 1\nLine 2\nLine 3";
$last = substr( strrchr( $text, 10 ), 1 );
```

See also `substr`, `stristr`, and `strstr`.

str_repeat

Name

`str_repeat` — Repeat a string

Description

string **`str_repeat`**(string *`input`*, int *`multiplier`*);

Returns *`input_str`* repeated *`multiplier`* times. *`multiplier`* has to be greater than 0.

Example 1. `str_repeat` example

```
echo str_repeat("-=", 10);
```

This will output "-=-=-=-=-=-=-=-=-=-=".

> **Note:** This function was added in PHP 4.0.

strrev

Name

`strrev` — Reverse a string

Description

string **`strrev`**(string *`string`*);

Returns *`string`*, reversed.

strrpos

Name

`strrpos` — Find position of last occurrence of a char in a string

Description

```
int strrpos(string haystack, char needle);
```

Returns the numeric position of the last occurrence of *`needle`* in the *`haystack`* string. Note that the needle in this case can only be a single character. If a string is passed as the needle, then only the first character of that string will be used.

If *`needle`* is not found, returns false.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

See also `strpos`, `strrchr`, `substr`, `stristr`, and `strstr`.

strspn

Name

`strspn` — Find length of initial segment matching mask

Description

```
int strspn(string str1, string str2);
```

Returns the length of the initial segment of *`str1`* which consists entirely of characters in *`str2`*.

See also `strcspn`.

strstr

Name

`strstr` — Find first occurrence of a string.

Description

```
string strstr(string haystack, string needle);
```

Returns all of *`haystack`* from the first occurrence of *`needle`* to the end.

If *`needle`* is not found, returns false.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

See also `stristr`, `strrchr`, `substr`, and `ereg`.

strtok

Name

`strtok` — Tokenize string

Description

```
string strtok(string arg1, string arg2);
```

`strtok` is used to tokenize a string. That is, if you have a string like "This is an example string" you could tokenize this string into its individual words by using the space character as the token.

Example 1. `strtok` example

```
    $string = "This is an example string";
    $tok = strtok($string," ");
```

```
while($tok) {
    echo "Word=$tok<br>";
    $tok = strtok(" ");
}
```

Note that only the first call to strtok uses the string argument. Every subsequent call to strtok only needs the token to use, as it keeps track of where it is in the current string. To start over, or to tokenize a new string you simply call strtok with the string argument again to initialize it. Note that you may put multiple tokens in the token parameter. The string will be tokenized when any one of the characters in the argument are found.

Also be careful that your tokens may be equal to "0". This evaluates to false in conditional expressions.

See also `split` and `explode`.

strtolower

Name

`strtolower` — Make a string lowercase

Description

`string` **`strtolower`**`(string` *`str`*`);`

Returns *`string`* with all alphabetic characters converted to lowercase.

Note that ’alphabetic’ is determined by the current locale. This means that in i.e. the default "C" locale, characters such as umlaut-A (Ä) will not be converted.

See also `strtoupper` and `ucfirst`.

strtoupper

Name

`strtoupper` — Make a string uppercase

Description

```
string strtoupper(string string);
```

Returns *`string`* with all alphabetic characters converted to uppercase.

Note that ’alphabetic’ is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

See also `strtolower` and `ucfirst`.

str_replace

Name

`str_replace` — Replace all occurrences of needle in haystack with str

Description

```
string str_replace(string needle, string str, string haystack);
```

This function replaces all occurences of *`needle`* in *`haystack`* with the given *`str`*. If you don’t need fancy replacing rules, you should always use this function instead of `ereg_replace`.

Example 1. str_replace() example

```
$bodytag = str_replace("%body%", "black", "<body text=%body%>");
```

This function is binary safe.

See also `ereg_replace`.

strtr

Name

`strtr` — Translate certain characters

Description

```
string strtr(string str, string from, string to);
```

This function operates on *`str`*, translating all occurrences of each character in *`from`* to the corresponding character in *`to`* and returning the result.

If *`from`* and *`to`* are different lengths, the extra characters in the longer of the two are ignored.

Example 1. strtr() example

```
$addr = strtr($addr, "äåö", "aao");
```

See also `ereg_replace`.

substr

Name

`substr` — Return part of a string

Description

```
string substr(string string, int start, int [length] );
```

Substr returns the portion of *`string`* specified by the *`start`* and *`length`* parameters.

If *`start`* is positive, the returned string will start at the *`start`*'th character of *`string`*.

Examples:

```
$rest = substr("abcdef", 1); // returns "bcdef"
$rest = substr("abcdef", 1, 3); // returns "bcd"
```

If *`start`* is negative, the returned string will start at the *`start`*'th character from the end of *`string`*.

Examples:

```
$rest = substr("abcdef", -1); // returns "f"
$rest = substr("abcdef", -2); // returns "ef"
$rest = substr("abcdef", -3, 1); // returns "d"
```

If *`length`* is given and is positive, the string returned will end *`length`* characters from *`start`*. If this would result in a string with negative length (because the start is past the end of the string), then the returned string will contain the single character at *`start`*.

If *`length`* is given and is negative, the string returned will end *`length`* characters from the end of *`string`*. If this would result in a string with negative length, then the returned string will contain the single character at *`start`*.

Examples:

```
$rest = substr("abcdef", 1, -1); // returns "bcde"
```

See also `strrchr` and `ereg`.

trim

Name

`trim` — Strip whitespace from the beginning and end of a string

Description

`string` **`trim`**`(string` *`str`*`);`

This function strips whitespace from the start and the end of a string and returns the stripped string.

See also `chop` and `ltrim`.

ucfirst

Name

`ucfirst` — Make a string’s first character uppercase

Description

`string` **`ucfirst`**`(string` *`str`*`);`

Capitalizes the first character of *`str`* if that character is alphabetic.

Note that ’alphabetic’ is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

See also `strtoupper` and `strtolower`.

ucwords

Name

`ucwords` — Uppercase the first character of each word in a string

Description

```
string ucwords(string str);
```

Capitalizes the first character of each word in *`str`* if that character is alphabetic.

See also `strtoupper`, `strtolower` and `ucfirst`.

XXVI. URL functions

base64_decode

Name

`base64_decode` — decodes data encoded with MIME base64

Description

`string` **`base64_decode`**`(string` *`encoded_data`*`);`

`base64_decode` decodes *`encoded_data`* and returns the original data. The returned data may be binary.

See also: `base64_encode`, RFC-2045 section 6.8.

base64_encode

Name

`base64_encode` — encodes data with MIME base64

Description

`string` **`base64_encode`**`(string` *`data`*`);`

`base64_encode` returns *`data`* encoded with base64. This encoding is designed to make binary data survive transport through transport layers that are not 8-bit clean, such as mail bodies.

Base64-encoded data takes about 33% more space than the original data.

See also: `base64_decode`, `chunk_split`, RFC-2045 section 6.8.

parse_url

Name

`parse_url` — parse a URL and return its components

Description

```
array parse_url(string url);
```

This function returns an associative array returning any of the various components of the URL that are present. This includes the "scheme", "host", "port", "user", "pass", "path", "query", and "fragment".

urldecode

Name

`urldecode` — decodes URL-encoded string

Description

```
string urldecode(string str);
```

Decodes any `%##` encoding in the given string. The decoded string is returned.

Example 1. urldecode() example

```
$a = split ('&', $querystring);
$i = 0;
while ($i < count ($a)) {
  $b = split ('=', $a [$i]);
  echo 'Value for parameter ', htmlspecialchars (urldecode ($b [0])),
       ' is ', htmlspecialchars (urldecode ($b [1])), "<BR>";
  $i++;
}
```

See also `urlencode`

urlencode

Name

`urlencode` — URL-encodes string

Description

```
string urlencode(string str);
```

Returns a string in which all non-alphanumeric characters except `-_.` have been replaced with a percent (%) sign followed by two hex digits and spaces encoded as plus (+) signs. It is encoded the same way that the posted data from a WWW form is encoded, that is the same way as in `application/x-www-form-urlencoded` media type. This differs from the RFC1738 encoding (see `rawurlencode`) in that for historical reasons, spaces are encoded as plus (+) signs. This function is convenient when encoding a string to be used in a query part of an URL, as a convenient way to pass variables to the next page:

Example 1. urlencode() example

```
echo '<A HREF="mycgi?foo=', urlencode ($userinput), '">';
```

See also `urldecode`

XXVII. Variable functions

doubleval

Name

`doubleval` — Get double value of a variable.

Description

`double **doubleval**(mixed *var*);`

Returns the double (floating point) value of *`var`*.

`var` may be any scalar type. You cannot use `doubleval` on arrays or objects.

See also `intval`, `strval`, `settype` and Type juggling.

empty

Name

`empty` — determine whether a variable is set

Description

`int **empty**(mixed *var*);`

Returns false if *`var`* is set and has a non-empty or non-zero value; true otherwise.

See also `isset` and `unset`.

gettype

Name

`gettype` — Get the type of a variable.

Description

```
string gettype(mixed var);
```

Returns the type of the PHP variable *`var`*.

Possibles values for the returned string are:

- "integer"
- "double"
- "string"
- "array"
- "object"
- "unknown type"

See also `settype`.

intval

Name

`intval` — Get integer value of a variable.

Description

`int` **`intval`**`(mixed` *`var`*`, int` *`[base]`*`);`

Returns the integer value of *`var`*, using the specified base for the conversion (the default is base 10).

`var` may be any scalar type. You cannot use `intval` on arrays or objects.

See also `doubleval`, `strval`, `settype` and Type juggling.

is_array

Name

`is_array` — Finds whether a variable is an array.

Description

`int` **`is_array`**`(mixed` *`var`*`);`

Returns true if *`var`* is an array, false otherwise.

See also `is_double`, `is_float`, `is_int`, `is_integer`, `is_real`, `is_string`, `is_long`, and `is_object`.

is_double

Name

`is_double` — Finds whether a variable is a double.

Description

```
int is_double(mixed var);
```

Returns true if *`var`* is a double, false otherwise.

See also `is_array`, `is_float`, `is_int`, `is_integer`, `is_real`, `is_string`, `is_long`, and `is_object`.

is_float

Name

`is_float` — Finds whether a variable is a float.

Description

```
int is_float(mixed var);
```

This function is an alias for `is_double`.

See also `is_double`, `is_real`, `is_int`, `is_integer`, `is_string`, `is_object`, `is_array`, and `is_long`.

is_int

Name

`is_int` — Find whether a variable is an integer.

Description

`int **is_int**(mixed *var*);`

This function is an alias for `is_long`.

See also `is_double`, `is_float`, `is_integer`, `is_string`, `is_real`, `is_object`, `is_array`, and `is_long`.

is_integer

Name

`is_integer` — Find whether a variable is an integer.

Description

`int **is_integer**(mixed *var*);`

This function is an alias for `is_long`.

See also `is_double`, `is_float`, `is_int`, `is_string`, `is_real`, `is_object`, `is_array`, and `is_long`.

is_long

Name

`is_long` — Finds whether a variable is an integer.

Description

`int **is_long**(mixed *var*);`

Returns true if *`var`* is an integer (long), false otherwise.

See also `is_double`, `is_float`, `is_int`, `is_real`, `is_string`, `is_object`, `is_array`, and `is_integer`.

is_object

Name

`is_object` — Finds whether a variable is an object.

Description

`int **is_object**(mixed *var*);`

Returns true if *`var`* is an object, false otherwise.

See also `is_long`, `is_int`, `is_integer`, `is_float`, `is_double`, `is_real`, `is_string`, and `is_array`.

is_real

Name

`is_real` — Finds whether a variable is a real.

Description

`int` **`is_real`**`(mixed` *`var`*`);`

This function is an alias for `is_double`.

See also `is_long`, `is_int`, `is_integer`, `is_float`, `is_double`, `is_object`, `is_string`, and `is_array`.

is_string

Name

`is_string` — Finds whether a variable is a string.

Description

`int` **`is_string`**`(mixed` *`var`*`);`

Returns true if *`var`* is a string, false otherwise.

See also `is_long`, `is_int`, `is_integer`, `is_float`, `is_double`, `is_real`, `is_object`, and `is_array`.

isset

Name

`isset` — determine whether a variable is set

Description

```
int isset(mixed var);
```

Returns true if `var` exists; false otherwise.

If a variable has been unset with `unset`, it will no longer be `isset`.

```
$a = "test";
echo isset($a); // true
unset($a);
echo isset($a); // false
```

See also `empty` and `unset`.

settype

Name

`settype` — Set the type of a variable.

Description

```
int settype(string var, string type);
```

Set the type of variable `var` to `type`.

Possibles values of `type` are:

- "integer"
- "double"
- "string"
- "array"
- "object"

Returns true if successful; otherwise returns false.

See also `gettype`.

strval

Name

`strval` — Get string value of a variable.

Description

`string` **`strval`**`(mixed` *`var`*`);`

Returns the string value of *`var`*.

`var` may be any scalar type. You cannot use `strval` on arrays or objects.

See also `doubleval`, `intval`, `settype` and Type juggling.

unset

Name

`unset` — Unset a given variable

Description

`int` **`unset`**`(mixed` *`var`*`);`

`unset` destroys the specified variable and returns true.

Example 1. `unset` example

```
unset( $foo );
unset( $bar['quux'] );
```

See also `isset` and `empty`.

XXVIII. Vmailmgr functions

These functions require qmail (http://www.qmail.org/) and the vmailmgr package (http://www.qcc.sk.ca/~bguenter/distrib/vmailmgr/) by Bruce Guenter.

For all functions, the following two variables are defined as: string vdomain the domain name of your virtual domain (vdomain.com) string basepwd the password of the ’real’ user that holds the virtual users

Only up to 8 characters are recognized in passwords for virtual users

Return status for all functions matches response in response.h

O ok

1 bad

2 error

3 error connecting

Known problems: `vm_deluser` does not delete the user directory as it should. `vm_addalias` currently does not work correctly.

```
<?php
dl("php3_vmailmgr.so"); //load the shared library
$vdomain="vdomain.com";
$basepwd="password";
?>
```

vm_adduser

Name

`vm_adduser` — Add a new virtual user with a password

Description

`int` **`vm_adduser`**`(string` *`vdomain`*`, string` *`basepwd`*`, string` *`newusername`*`, string` *`newuserpassword`*`);`

Add a new virtual user with a password. *`newusername`* is the email login name and *`newuserpassword`* the password for this user.

vm_addalias

Name

`vm_addalias` — Add an alias to a virtual user

Description

`int` **`vm_addalias`**`(string` *`vdomain`*`, string` *`basepwd`*`, string` *`username`*`, string` *`alias`*`);`

Add an alias to a virtual user. *`username`* is the email login name and *`alias`* is an alias for this vuser.

vm_passwd

Name

`vm_passwd` — Changes a virtual users password

Description

`int` **`vm_passwd`**`(string` *`vdomain`*`, string` *`username`*`, string` *`password`*`, string` *`newpassword`*`);`

Changes a virtual users password. *`username`* is the email login name, *`password`* the old password for the vuser, and *`newpassword`* the new password.

vm_delalias

Name

`vm_delalias` — Removes an alias

Description

`int` **`vm_delalias`**`(string` *`vdomain`*`, string` *`basepwd`*`, string` *`alias`*`);`

Removes an alias.

vm_deluser

Name

`vm_deluser` — Removes a virtual user

Description

`int` **`vm_deluser`**`(string` *`vdomain`*`, string` *`username`*`);`

Removes a virtual user..

XXIX. WDDX functions

These functions are intended for work with WDDX (http://www.wddx.org).

Note that all the functions that serialize variables use the first element of an array to determine whether the array is to be serialized into an array or structure. If the first element has string key, then it is serialized into a structure, otherwise, into an array.

Example 1. Serializing a single value

```
<?php
print wddx_serialize_value("PHP to WDDX packet example", "PHP packet");
?>
```

This example will produce:

```
<wddxPacket version='0.9'><header comment='PHP packet'/><data>
<string>PHP to WDDX packet example</string></data></wddxPacket>
```

Example 2. Using incremental packets

```
<?php
$pi = 3.1415926;
$packet_id = wddx_packet_start("PHP");
wddx_add_vars($packet_id, "pi");

/* Suppose $cities came from database */
$cities = array("Austin", "Novato", "Seattle");
wddx_add_vars($packet_id, "cities");

$packet = wddx_packet_end($packet_id);
print $packet;
?>
```

This example will produce:

```
<wddxPacket version='0.9'><header comment='PHP'/><data><struct>
<var name='pi'><number>3.1415926</number></var><var name='cities'>
<array length='3'><string>Austin</string><string>Novato</string>
<string>Seattle</string></array></var></struct></data></wddxPacket>
```

wddx_serialize_value

Name

`wddx_serialize_value` — Serialize a single value into a WDDX packet

Description

```
string wddx_serialize_value(mixed var, string [comment]);
```

`wddx_serialize_value` is used to create a WDDX packet from a single given value. It takes the value contained in *`var`*, and an optional *`comment`* string that appears in the packet header, and returns the WDDX packet.

wddx_serialize_vars

Name

`wddx_serialize_vars` — Serialize variables into a WDDX packet

Description

```
string wddx_serialize_vars(string var_name | array var_names [, ... ] );
```

`wddx_serialize_vars` is used to create a WDDX packet with a structure that contains the serialized representation of the passed variables.

`wddx_serialize_vars` takes a variable number of arguments, each of which can be either a string naming a variable or an array containing strings naming the variables or another array, etc.

Example 1. wddx_serialize_vars example

```
<?php
```

```
$a = 1;
$b = 5.5;
$c = array("blue", "orange", "violet");
$d = "colors";

$clvars = array("c", "d");
print wddx_serialize_vars("a", "b", $clvars);
?>
```

The above example will produce:

```
<wddxPacket version='0.9'><header/><data><struct><var name='a'><number>1</number></var
<var name='b'><number>5.5</number></var><var name='c'><array length='3'>
<string>blue</string><string>orange</string><string>violet</string></array></var>
<var name='d'><string>colors</string></var></struct></data></wddxPacket>
```

wddx_packet_start

Name

`wddx_packet_start` — Starts a new WDDX packet with structure inside it

Description

`int` **`wddx_packet_start`**`(string` *`[comment]`*`);`

Use `wddx_packet_start` to start a new WDDX packet for incremental addition of variables. It takes an optional *`comment`* string and returns a packet ID for use in later functions. It automatically creates a structure definition inside the packet to contain the variables.

wddx_packet_end

Name

`wddx_packet_end` — Ends a WDDX packet with the specified ID

Description

`string` **`wddx_packet_end`**`(int` *`packet_id`*`);`

`wddx_packet_end` ends the WDDX packet specified by the *`packet_id`* and returns the string with the packet.

wddx_add_vars

Name

`wddx_add_vars` — Ends a WDDX packet with the specified ID

Description

`wddx_add_vars``(int` *`packet_id`*`, ...);`

`wddx_add_vars` is used to serialize passed variables and add the result to the packet specified by the *`packet_id`*. The variables to be serialized are specified in exactly the same way as `wddx_serialize_vars`.

wddx_deserialize

Name

`wddx_deserialize` — Deserializes a WDDX packet

Description

`mixed` **`wddx_deserialize`**`(string` *`packet`*`);`

`wddx_deserialized` takes a *`packet`* string and deserializes it. It returns the result which can be string, number, or array. Note that structures are deserialized into associative arrays.

XXX. Compression functions

This module uses the functions of zlib (http://www.cdrom.com/pub/infozip/zlib/) by Jean-loup Gailly and Mark Adler to transparently read and write gzip (.gz) compressed files. You have to use a zlib version >= 1.0.9 with this module.

This module contains versions of most of the filesystem functions which work with gzip-compressed files (and uncompressed files, too, but not with sockets).

Small code example

Opens a temporary file and writes a test string to it, then it prints out the content of this file twice.

Example 1. Small Zlib example

```
<?php
  $filename = tempnam('/tmp', 'zlibtest').'.gz';
  print "<html>\n<head></head>\n<body>\n<pre>\n";
  $s = "Only a test, test, test, test, test, test, test, test!\n";
  // open file for writing with maximum compression
  $zp = gzopen($filename, "w9");
  // write string to file
  gzwrite($zp, $s);
  // close file
  gzclose($zp);
  // open file for reading
  $zp = gzopen($filename, "r");
  // read 3 char
  print gzread($zp, 3);
  // output until end of the file and close it.
  gzpassthru($zp);
  print "\n";
  // open file and print content (the 2nd time).
  if (readgzfile($filename) != strlen($s)) {
          echo "Error with zlib functions!";
  }
  unlink($filename);
  print "<pre>\n</h1></body>\n</html>\n";
?>
```

gzclose

Name

`gzclose` — close an open gz-file pointer

Description

```
int gzclose(int zp);
```

The gz-file pointed to by zp is closed.

Returns true on success and false on failure.

The gz-file pointer must be valid, and must point to a file successfully opened by `gzopen`.

gzeof

Name

`gzeof` — test for end-of-file on a gz-file pointer

Description

```
int gzeof(int zp);
```

Returns true if the gz-file pointer is at EOF or an error occurs; otherwise returns false.

The gz-file pointer must be valid, and must point to a file successfully opened by `gzopen`.

gzfile

Name

`gzfile` — read entire gz-file into an array

Description

```
array gzfile(string filename, int [use_include_path]);
```

Identical to `readgzfile`, except that gzfile() returns the file in an array.

You can use the optional second parameter and set it to "1", if you want to search for the file in the include_path, too.

See also `readgzfile`, and `gzopen`.

gzgetc

Name

`gzgetc` — get character from gz-file pointer

Description

```
string gzgetc(int zp);
```

Returns a string containing a single (uncompressed) character read from the file pointed to by zp. Returns FALSE on EOF (as does `gzeof`).

The gz-file pointer must be valid, and must point to a file successfully opened by `gzopen`.

See also `gzopen`, and `gzgets`.

gzgets

Name

`gzgets` — get line from file pointer

Description

`string` **`gzgets`**`(int` *`zp`*`, int` *`length`*`);`

Returns a (uncompressed) string of up to length - 1 bytes read from the file pointed to by fp. Reading ends when length - 1 bytes have been read, on a newline, or on EOF (whichever comes first).

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by `gzopen`.

See also `gzopen`, `gzgetc`, and `fgets`.

gzgetss

Name

`gzgetss` — get line from gz-file pointer and strip HTML tags

Description

`string` **`gzgetss`**`(int` *`zp`*`, int` *`length`*`, string` *`[allowable_tags]`*`);`

Identical to `gzgets`, except that gzgetss attempts to strip any HTML and PHP tags from the text it reads.

You can use the optional third parameter to specify tags which should not be stripped.

Note: *`allowable_tags`* was added in PHP 3.0.13, PHP4B3.

See also `gzgets`, `gzopen`, and `strip_tags`.

gzopen

Name

`gzopen` — open gz-file

Description

```
int gzopen(string filename, string mode, int [use_include_path]);
```

Opens a gzip (.gz) file for reading or writing. The mode parameter is as in `fopen` ("rb" or "wb") but can also include a compression level ("wb9") or a strategy: ’f’ for filtered data as in "wb6f", ’h’ for Huffman only compression as in "wb1h". (See the description of deflateInit2 in zlib.h for more information about the strategy parameter.)

Gzopen can be used to read a file which is not in gzip format; in this case `gzread` will directly read from the file without decompression.

Gzopen returns a file pointer to the file opened, after that, everything you read from this file descriptor will be transparently decompressed and what you write gets compressed.

If the open fails, the function returns false.

You can use the optional third parameter and set it to "1", if you want to search for the file in the include_path, too.

Example 1. gzopen() example

```
$fp = gzopen("/tmp/file.gz", "r");
```

See also `gzclose`.

gzpassthru

Name

`gzpassthru` — output all remaining data on a gz-file pointer

Description

`int` **`gzpassthru`**`(int` *`zp`*`);`

Reads to EOF on the given gz-file pointer and writes the (uncompressed) results to standard output.

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by `gzopen`.

The gz-file is closed when `gzpassthru` is done reading it (leaving *`zp`* useless).

gzputs

Name

`gzputs` — write to a gz-file pointer

Description

`int` **`gzputs`**`(int` *`zp`*`, string` *`str`*`, int` *`[length]`*`);`

`gzputs` is an alias to `gzwrite`, and is identical in every way.

gzread

Name

`gzread` — Binary-safe gz-file read

Description

```
string gzread(int zp, int length);
```

`gzread` reads up to *`length`* bytes from the gz-file pointer referenced by *`zp`*. Reading stops when *`length`* (uncompressed) bytes have been read or EOF is reached, whichever comes first.

```
// get contents of a gz-file into a string
$filename = "/usr/local/something.txt.gz";
$zd = gzopen( $filename, "r" );
$contents = gzread( $zd, 10000 );
gzclose( $zd );
```

See also `gzwrite`, `gzopen`, `gzgets`, `gzgetss`, `gzfile`, and `gzpassthru`.

gzrewind

Name

`gzrewind` — rewind the position of a gz-file pointer

Description

```
int gzrewind(int zp);
```

Sets the file position indicator for zp to the beginning of the file stream.

If an error occurs, returns 0.

The file pointer must be valid, and must point to a file successfully opened by `gzopen`.

See also `gzseek` and `gztell`.

gzseek

Name

`gzseek` — seek on a gz-file pointer

Description

`int gzseek(int zp, int offset);`

Sets the file position indicator for the file referenced by zp to offset bytes into the file stream. Equivalent to calling (in C) `gzseek( zp, offset, SEEK_SET )`.

If the file is opened for reading, this function is emulated but can be extremely slow. If the file is opened for writing, only forward seeks are supported; gzseek then compresses a sequence of zeroes up to the new starting position.

Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

See also `gztell` and `gzrewind`.

gztell

Name

`gztell` — tell gz-file pointer read/write position

Description

`int gztell(int zp);`

Returns the position of the file pointer referenced by zp; i.e., its offset into the file stream.

If an error occurs, returns false.

The file pointer must be valid, and must point to a file successfully opened by `gzopen`.

See also `gzopen`, `gzseek` and `gzrewind`.

gzwrite

Name

`gzwrite` — Binary-safe gz-file write

Description

`int gzwrite(int zp, string string, int [length]);`

`gzwrite` writes the contents of *`string`* to the gz-file stream pointed to by *`zp`*. If the *`length`* argument is given, writing will stop after *`length`* (uncompressed) bytes have been written or the end of *`string`* is reached, whichever comes first.

Note that if the *`length`* argument is given, then the magic_quotes_runtime configuration option will be ignored and no slashes will be stripped from *`string`*.

See also `gzread`, `gzopen`, and `gzputs`.

readgzfile

Name

`readgzfile` — output a gz-file

Description

```
int readgzfile(string filename, int [use_include_path]);
```

Reads a file, decompresses it and writes it to standard output.

Readgzfile() can be used to read a file which is not in gzip format; in this case readgzfile() will directly read from the file without decompression.

Returns the number of (uncompressed) bytes read from the file. If an error occurs, false is returned and unless the function was called as @readgzfile, an error message is printed.

The file *`filename`* will be opened from the filesystem and its contents written to standard output.

You can use the optional second parameter and set it to "1", if you want to search for the file in the include_path, too.

See also `gzpassthru`, `gzfile`, and `gzopen`.

XXXI. XML parser functions

Introduction

About XML

XML (eXtensible Markup Language) is a data format for structured document interchange on the Web. It is a standard defined by The World Wide Web consortium (W3C). Information about XML and related technologies can be found at http://www.w3.org/XML/.

Installation

This extension uses expat, which can be found at http://www.jclark.com/xml/. The Makefile that comes with expat does not build a library by default, you can use this make rule for that:

```
libexpat.a: $(OBJS)
ar -rc $@ $(OBJS)
ranlib $@
```

A source RPM package of expat can be found at http://www.guardian.no/~ssb/phpxml.html.

Note that if you are using Apache-1.3.7 or later, you already have the required expat library. Simply configure PHP using `-with-xml` (without any additional path) and it will automatically use the expat library built into Apache.

On UNIX, run **configure** with the `-with-xml` option. The expat library should be installed somewhere your compiler can find it. If you compile PHP as a module for Apache 1.3.9 or later, PHP will automatically use the bundled expat library from Apache. You may need to set CPPFLAGS and LDFLAGS in your environment before running configure if you have installed expat somewhere exotic.

Build PHP. *Tada!* That should be it.

About This Extension

This PHP extension implements support for James Clark's expat in PHP. This toolkit lets you parse, but not validate, XML documents. It supports three source character encodings also provided by PHP: `US-ASCII`, `ISO-8859-1` and `UTF-8`. `UTF-16` is not supported.

This extension lets you create XML parsers and then define *handlers* for different XML events. Each XML parser also has a few parameters you can adjust.

The XML event handlers defined are:

Table 1. Supported XML handlers

PHP function to set handler	**Event description**
`xml_set_element_handler`	Element events are issued whenever the XML parser encounters start or end tags. There are separate handlers for start tags and end tags.
`xml_set_character_data_handler`	Character data is roughly all the non-markup contents of XML documents, including whitespace between tags. Note that the XML parser does not add or remove any whitespace, it is up to the application (you) to decide whether whitespace is significant.
`xml_set_processing_instruction_handler`	PHP programmers should be familiar with processing instructions (PIs) already. <?php ?> is a processing instruction, where *`php is called the "PI target". The handling of these are application-specific, except that all PI targets starting with "XML" are reserved.`*
`xml_set_default_handler`	What goes not to another handler goes to the default handler. You will get things like the XML and document type declarations in the default handler.
`xml_set_unparsed_entity_decl_handler`	This handler will be called for declaration of an unparsed (NDATA) entity.
`xml_set_notation_decl_handler`	This handler is called for declaration of a notation.
`xml_set_external_entity_ref_handler`	This handler is called when the XML parser finds a reference to an external parsed general entity. This can be a reference to a file or URL, for example. See the external entity example for a demonstration.

Case Folding

The element handler functions may get their element names *case-folded*. Case-folding is defined by the XML standard as "a process applied to a sequence of characters, in which those identified as non-uppercase are replaced by their uppercase equivalents". In other words, when it comes to XML, case-folding simply means uppercasing.

By default, all the element names that are passed to the handler functions are case-folded. This behaviour can be queried and controlled per XML parser with the `xml_parser_get_option` and `xml_parser_set_option` functions, respectively.

Error Codes

The following constants are defined for XML error codes (as returned by `xml_parse`):

XML_ERROR_NONE
XML_ERROR_NO_MEMORY
XML_ERROR_SYNTAX
XML_ERROR_NO_ELEMENTS
XML_ERROR_INVALID_TOKEN
XML_ERROR_UNCLOSED_TOKEN
XML_ERROR_PARTIAL_CHAR
XML_ERROR_TAG_MISMATCH
XML_ERROR_DUPLICATE_ATTRIBUTE
XML_ERROR_JUNK_AFTER_DOC_ELEMENT
XML_ERROR_PARAM_ENTITY_REF
XML_ERROR_UNDEFINED_ENTITY
XML_ERROR_RECURSIVE_ENTITY_REF
XML_ERROR_ASYNC_ENTITY
XML_ERROR_BAD_CHAR_REF
XML_ERROR_BINARY_ENTITY_REF
XML_ERROR_ATTRIBUTE_EXTERNAL_ENTITY_REF
XML_ERROR_MISPLACED_XML_PI
XML_ERROR_UNKNOWN_ENCODING
XML_ERROR_INCORRECT_ENCODING

XML_ERROR_UNCLOSED_CDATA_SECTION
XML_ERROR_EXTERNAL_ENTITY_HANDLING

Character Encoding

PHP’s XML extension supports the Unicode (http://www.unicode.org/) character set through different *character encoding*s. There are two types of character encodings, *source encoding* and *target encoding*. PHP’s internal representation of the document is always encoded with `UTF-8`.

Source encoding is done when an XML document is parsed. Upon creating an XML parser, a source encoding can be specified (this encoding can not be changed later in the XML parser’s lifetime). The supported source encodings are `ISO-8859-1`, `US-ASCII` and `UTF-8`. The former two are single-byte encodings, which means that each character is represented by a single byte. `UTF-8` can encode characters composed by a variable number of bits (up to 21) in one to four bytes. The default source encoding used by PHP is `ISO-8859-1`.

Target encoding is done when PHP passes data to XML handler functions. When an XML parser is created, the target encoding is set to the same as the source encoding, but this may be changed at any point. The target encoding will affect character data as well as tag names and processing instruction targets.

If the XML parser encounters characters outside the range that its source encoding is capable of representing, it will return an error.

If PHP encounters characters in the parsed XML document that can not be represented in the chosen target encoding, the problem characters will be "demoted". Currently, this means that such characters are replaced by a question mark.

Some Examples

Here are some example PHP scripts parsing XML documents.

XML Element Structure Example

This first example displays the stucture of the start elements in a document with indentation.

Example 1. Show XML Element Structure

```
$file = "data.xml";
$depth = array();

function startElement($parser, $name, $attrs)
{
    global $depth;
    for ($i = 0; $i < $depth[$parser]; $i++) {
        print "  ";
    }
    print "$name\n";
    $depth[$parser]++;
}

function endElement($parser, $name)
{
    global $depth;
    $depth[$parser]-;
}

$xml_parser = xml_parser_create();
xml_set_element_handler($xml_parser, "startElement", "endElement");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}
while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
xml_parser_free($xml_parser);
```

XML Tag Mapping Example

Example 2. Map XML to HTML

This example maps tags in an XML document directly to HTML tags. Elements not found in the "map array" are ignored. Of course, this example will only work with a specific XML document type.

```
$file = "data.xml";
$map_array = array(
    "BOLD"     => "B",
    "EMPHASIS" => "I",
    "LITERAL"  => "TT"
);

function startElement($parser, $name, $attrs)
{
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "<$htmltag>";
    }
}

function endElement($parser, $name)
{
    global $map_array;
    if ($htmltag = $map_array[$name]) {
        print "</$htmltag>";
    }
}

function characterData($parser, $data)
{
    print $data;
}

$xml_parser = xml_parser_create();
// use case-folding so we are sure to find the tag in $map_array
xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
xml_set_element_handler($xml_parser, "startElement", "endElement");
xml_set_character_data_handler($xml_parser, "characterData");
if (!($fp = fopen($file, "r"))) {
    die("could not open XML input");
}
while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
xml_parser_free($xml_parser);
```

XML External Entity Example

This example highlights XML code. It illustrates how to use an external entity reference handler to include and parse other documents, as well as how PIs can be processed, and a way of determining "trust" for PIs containing code.

XML documents that can be used for this example are found below the example (`xmltest.xml` and `xmltest2.xml`.)

Example 3. External Entity Example

```
$file = "xmltest.xml";

function trustedFile($file)
{
    // only trust local files owned by ourselves
    if (!eregi("^([a-z]+)://", $file) && fileowner($file) == getmyuid()) {
        return true;
    }
    return false;
}

function startElement($parser, $name, $attribs)
{
    print "&lt;<font color=\"#0000cc\">$name</font>";
    if (sizeof($attribs)) {
        while (list($k, $v) = each($attribs)) {
            print " <font color=\"#009900\">$k</font>=\"<font color=\"#990000\">$v</font>
        }
    }
    print "&gt;";
}
```

```
function endElement($parser, $name)
{
    print "&lt;/<font color=\"#0000cc\">$name</font>&gt;";
}

function characterData($parser, $data)
{
    print "<b>$data</b>";
}

function PIHandler($parser, $target, $data)
{
    switch (strtolower($target)) {
        case "php":
            global $parser_file;
            // If the parsed document is "trusted", we say it is safe
            // to execute PHP code inside it.  If not, display the code
            // instead.
            if (trustedFile($parser_file[$parser])) {
                eval($data);
            } else {
                printf("Untrusted PHP code: <i>%s</i>", htmlspecialchars($data));
            }
            break;
    }
}

function defaultHandler($parser, $data)
{
    if (substr($data, 0, 1) == "&" && substr($data, -1, 1) == ";") {
        printf(’<font color="#aa00aa">%s</font>’, htmlspecialchars($data));
    } else {
        printf(’<font size="-1">%s</font>’, htmlspecialchars($data));
    }
}

function externalEntityRefHandler($parser, $openEntity-
Names, $base, $systemId,
                                  $publicId)
{
    if ($systemId) {
        if (!list($parser, $fp) = new_xml_parser($systemId)) {
            printf("Could not open entity %s at %s\n", $openEntityNames,
                   $systemId);
            return false;
        }
        while ($data = fread($fp, 4096)) {
            if (!xml_parse($parser, $data, feof($fp))) {
                printf("XML error: %s at line %d while parsing entity %s\n",
                       xml_error_string(xml_get_error_code($parser)),
                       xml_get_current_line_number($parser), $openEntityNames);
                xml_parser_free($parser);
                return false;
            }
        }
        xml_parser_free($parser);
        return true;
    }
    return false;
}


function new_xml_parser($file) {
    global $parser_file;

    $xml_parser = xml_parser_create();
    xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, 1);
    xml_set_element_handler($xml_parser, "startElement", "endElement");
    xml_set_character_data_handler($xml_parser, "characterData");
    xml_set_processing_instruction_handler($xml_parser, "PIHandler");
    xml_set_default_handler($xml_parser, "defaultHandler");
    xml_set_external_entity_ref_handler($xml_parser, "externalEntityRefHandler");

    if (!($fp = @fopen($file, "r"))) {
        return false;
    }
    if (!is_array($parser_file)) {
        settype($parser_file, "array");
    }
    $parser_file[$xml_parser] = $file;
    return array($xml_parser, $fp);
}

if (!(list($xml_parser, $fp) = new_xml_parser($file))) {
    die("could not open XML input");
}

print "<pre>";
while ($data = fread($fp, 4096)) {
    if (!xml_parse($xml_parser, $data, feof($fp))) {
        die(sprintf("XML error: %s at line %d\n",
                    xml_error_string(xml_get_error_code($xml_parser)),
                    xml_get_current_line_number($xml_parser)));
    }
}
print "</pre>";
print "parse complete\n";
xml_parser_free($xml_parser);

?>
```

Example 4. xmltest.xml

```
<?xml version='1.0'?>
<!DOCTYPE chapter SYSTEM "/just/a/test.dtd" [
<!ENTITY plainEntity "FOO entity">
<!ENTITY systemEntity SYSTEM "xmltest2.xml">
]>
<chapter>
 <TITLE>Title &plainEntity;</TITLE>
 <para>
  <informaltable>
   <tgroup cols="3">
    <tbody>
     <row><entry>a1</entry><entry morerows="1">b1</entry><entry>c1</entry></row>
     <row><entry>a2</entry><entry>c2</entry></row>
     <row><entry>a3</entry><entry>b3</entry><entry>c3</entry></row>
    </tbody>
   </tgroup>
  </informaltable>
 </para>
 &systemEntity;
 <sect1 id="about">
  <title>About this Document</title>
  <para>
   <!- this is a comment ->
   <?php print 'Hi!  This is PHP version '.phpversion(); ?>
  </para>
 </sect1>
</chapter>
```

This file is included from `xmltest.xml`:

Example 5. xmltest2.xml

```
<?xml version="1.0"?>
<!DOCTYPE foo [
<!ENTITY testEnt "test entity">
]>
<foo>
 <element attrib="value"/>
 &testEnt;
 <?php print "This is some more PHP code being executed."; ?>
</foo>
```

xml_parser_create

Name

`xml_parser_create` — create an XML parser

Description

`int` **`xml_parser_create`** `(string` *`[encoding]`*`);`

`encoding` (optional)

Which character encoding the parser should use. The following character encodings are supported:

`ISO-8859-1 (default)`

`US-ASCII`

`UTF-8`

This function creates an XML parser and returns a handle for use by other XML functions. Returns `false` on failure.

xml_set_element_handler

Name

`xml_set_element_handler` — set up start and end element handlers

Description

`int` **`xml_set_element_handler`** `(int` *`parser`*`, string` *`startElementHandler`*`, string` *`endElementHandler`*`);`

Sets the element handler functions for the XML parser *`parser`*. *`startElementHandler`* and *`endElementHandler`* are strings containing the names of functions that must exist when `xml_parse` is called for *`parser`*.

The function named by *`startElementHandler`* must accept three parameters:

```
startElementHandler(int parser, string name, string attribs);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`name`

The second parameter, *`name`*, contains the name of the element for which this handler is called. If case-folding is in effect for this parser, the element name will be in uppercase letters.

`attribs`

The third parameter, *`attribs`*, contains an associative array with the element's attributes (if any). The keys of this array are the attribute names, the values are the attribute values. Attribute names are case-folded on the same criteria as element names. Attribute values are *not* case-folded.

The original order of the attributes can be retrieved by walking through *`attribs`* the normal way, using `each`. The first key in the array was the first attribute, and so on.

The function named by *`endElementHandler`* must accept two parameters:

```
endElementHandler(int parser, string name);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`name`

The second parameter, *`name`*, contains the name of the element for which this handler is called. If case-folding is in effect for this parser, the element name will be in uppercase letters.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handlers are set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_character_data_handler

Name

`xml_set_character_data_handler` — set up character data handler

Description

```
int xml_set_character_data_handler (int parser, string handler);
```

Sets the character data handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

The function named by *`handler`* must accept two parameters:

```
handler(int parser, string data);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`data`

The second parameter, *`data`*, contains the character data as a string.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_processing_instruction_handler

Name

`xml_set_processing_instruction_handler` — set up processing instruction (PI) handler

Description

```
int xml_set_processing_instruction_handler (int parser, string handler);
```

Sets the processing instruction (PI) handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

A processing instruction has the following format:

```
<?target data?>
```

You can put PHP code into such a tag, but be aware of one limitation: in an XML PI, the PI end tag (`?>`) can not be quoted, so this character sequence should not appear in the PHP code you embed with PIs in XML documents. If it does, the rest of the PHP code, as well as the "real" PI end tag, will be treated as character data.

The function named by *`handler`* must accept three parameters:

```
handler(int parser, string target, string data);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`target`

The second parameter, *`target`*, contains the PI target.

`data`

The third parameter, *`data`*, contains the PI data.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_default_handler

Name

`xml_set_default_handler` — set up default handler

Description

```
int xml_set_default_handler (int parser, string handler);
```

Sets the default handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

The function named by *`handler`* must accept two parameters:

```
handler(int parser, string data);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`data`

The second parameter, *`data`*, contains the character data. This may be the XML declaration, document type declaration, entities or other data for which no other handler exists.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_unparsed_entity_decl_handler

Name

`xml_set_unparsed_entity_decl_handler` — set up unparsed entity declaration handler

Description

```
int xml_set_unparsed_entity_decl_handler (int parser, string handler);
```

Sets the unparsed entity declaration handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

This handler will be called if the XML parser encounters an external entity declaration with an NDATA declaration, like the following:

```
<!ENTITY name {publicId | systemId} NDATA notationName>
```

See section 4.2.2 of the XML 1.0 spec (http://www.w3.org/TR/1998/REC-xml-19980210#sec-external-ent) for the definition of notation declared external entities.

The function named by *`handler`* must accept six parameters:

```
handler(int parser, string entityName, string base, string systemId, string
publicId, string notationName);
```

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`entityName`

The name of the entity that is about to be defined.

`base`

This is the base for resolving the system identifier (*`systemId`*) of the external entity. Currently this parameter will always be set to an empty string.

`systemId`

System identifier for the external entity.

`publicId`

Public identifier for the external entity.

`notationName`

Name of the notation of this entity (see `xml_set_notation_decl_handler`).

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_notation_decl_handler

Name

`xml_set_notation_decl_handler` — set up notation declaration handler

Description

`int` **`xml_set_notation_decl_handler`** `(int` *`parser`*`, string` *`handler`*`);`

Sets the notation declaration handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

A notation declaration is part of the document's DTD and has the following format:

`<!NOTATION` *`name`* `{`*`systemId`* `|` *`publicId`*`}>`

See section 4.7 of the XML 1.0 spec (http://www.w3.org/TR/1998/REC-xml-19980210#Notations) for the definition of notation declarations.

The function named by *`handler`* must accept five parameters:

`handler``(int `*`parser`*`, string `*`notationName`*`, string `*`base`*`, string `*`systemId`*`, string `*`publicId`*`);`

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`notationName`

This is the notation's *`name`*, as per the notation format described above.

`base`

This is the base for resolving the system identifier (*`systemId`*) of the notation declaration. Currently this parameter will always be set to an empty string.

`systemId`

System identifier of the external notation declaration.

`publicId`

Public identifier of the external notation declaration.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_set_external_entity_ref_handler

Name

`xml_set_external_entity_ref_handler` — set up external entity reference handler

Description

`int` **`xml_set_external_entity_ref_handler`** `(int` *`parser`*`, string` *`handler`*`);`

Sets the notation declaration handler function for the XML parser *`parser`*. *`handler`* is a string containing the name of a function that must exist when `xml_parse` is called for *`parser`*.

The function named by *`handler`* must accept five parameters, and should return an integer value. If the value returned from the handler is false (which it will be if no value is returned), the XML parser will stop parsing and `xml_get_error_code` will return XML_ERROR_EXTERNAL_ENTITY_HANDLING.

`int` *`handler`*`(int` *`parser`*`, string` *`openEntityNames`*`, string` *`base`*`, string` *`systemId`*`, string` *`publicId`*`);`

`parser`

The first parameter, *`parser`*, is a reference to the XML parser calling the handler.

`openEntityNames`

The second parameter, *`openEntityNames`*, is a space-separated list of the names of the entities that are open for the parse of this entity (including the name of the referenced entity).

`base`

This is the base for resolving the system identifier (*`systemid`*) of the external entity. Currently this parameter will always be set to an empty string.

`systemId`

The fourth parameter, *`systemId`*, is the system identifier as specified in the entity declaration.

`publicId`

The fifth parameter, *`publicId`*, is the public identifier as specified in the entity declaration, or an empty string if none was specified; the whitespace in the public identifier will have been normalized as required by the XML spec.

If a handler function is set to an empty string, or `false`, the handler in question is disabled.

True is returned if the handler is set up, false if *`parser`* is not a parser.

There is currently no support for object/method handlers.

xml_parse

Name

`xml_parse` — start parsing an XML document

Description

`int **xml_parse**(int *parser*, string *data*, int *[isFinal]*);`

`parser`

A reference to the XML parser to use.

`data`

Chunk of data to parse. A document may be parsed piece-wise by calling `xml_parse` several times with new data, as long as the *`isFinal`* parameter is set and true when the last data is parsed.

`isFinal` (optional)

If set and true, *`data`* is the last piece of data sent in this parse.

When the XML document is parsed, the handlers for the configured events are called as many times as necessary, after which this function returns true or false.

True is returned if the parse was successful, false if it was not successful, or if *`parser`* does not refer to a valid parser. For unsuccessful parses, error information can be retrieved with `xml_get_error_code`, `xml_error_string`, `xml_get_current_line_number`, `xml_get_current_column_number` and `xml_get_current_byte_index`.

xml_get_error_code

Name

`xml_get_error_code` — get XML parser error code

Description

`int` **`xml_get_error_code`** `(int` *`parser`*`);`

`parser`

A reference to the XML parser to get error code from.

This function returns false if *`parser`* does not refer to a valid parser, or else it returns one of the error codes listed in the error codes section.

xml_error_string

Name

`xml_error_string` — get XML parser error string

Description

`string` **`xml_error_string`** `(int` *`code`*`);`

`code`

An error code from `xml_get_error_code`.

Returns a string with a textual description of the error code *`code`*, or false if no description was found.

xml_get_current_line_number

Name

`xml_get_current_line_number` — get current line number for an XML parser

Description

`int` **`xml_get_current_line_number`** `(int` *`parser`*`);`

`parser`

A reference to the XML parser to get line number from.

This function returns false if *`parser`* does not refer to a valid parser, or else it returns which line the parser is currently at in its data buffer.

xml_get_current_column_number

Name

`xml_get_current_column_number` — get current column number for an XML parser

Description

`int` **`xml_get_current_column_number`** `(int` *`parser`*`);`

`parser`

A reference to the XML parser to get column number from.

This function returns false if *`parser`* does not refer to a valid parser, or else it returns which column on the current line (as given by `xml_get_current_line_number`) the parser is currently at.

xml_get_current_byte_index

Name

`xml_get_current_byte_index` — get current byte index for an XML parser

Description

`int` **`xml_get_current_byte_index`** `(int` *`parser`*`);`

`parser`

A reference to the XML parser to get byte index from.

This function returns false if *`parser`* does not refer to a valid parser, or else it returns which byte index the parser is currently at in its data buffer (starting at 0).

xml_parser_free

Name

`xml_parser_free` — free an XML parser

Description

string **`xml_parser_free`**(int *parser*);

parser

A reference to the XML parser to free.

This function returns false if *`parser`* does not refer to a valid parser, or else it frees the parser and returns true.

xml_parser_set_option

Name

`xml_parser_set_option` — set options in an XML parser

Description

int **`xml_parser_set_option`**(int *parser*, int *option*, mixed *value*);

parser

A reference to the XML parser to set an option in.

option

Which option to set. See below.

value

The option’s new value.

This function returns false if *`parser`* does not refer to a valid parser, or if the option could not be set. Else the option is set and true is returned.

The following options are available:

Table 1. XML parser options

Option constant	Data type	Description
XML_OPTION_CASE_FOLDING	integer	Controls whether case-folding is enabled for this XML parser. Enabled by default.
XML_OPTION_TARGET_ENCODING	string	Sets which target encoding to use in this XML parser. By default, it is set to the same as the source encoding used by `xml_parser_create`. `Supported target encodings are ISO-8859-1, US-ASCII and UTF-8.`

xml_parser_get_option

Name

`xml_parser_get_option` — get options from an XML parser

Description

`mixed` **`xml_parser_get_option`** `(int` *`parser`*`, int` *`option`*`);`

`parser`

A reference to the XML parser to get an option from.

`option`

Which option to fetch. See `xml_parser_set_option` for a list of options.

This function returns false if *`parser`* does not refer to a valid parser, or if the option could not be set. Else the option's value is returned.

See `xml_parser_set_option` for the list of options.

utf8_decode

Name

`utf8_decode` — converts a UTF-8 encoded string to ISO-8859-1

Description

`string utf8_decode(string data);`

This function decodes *`data`*, assumed to be `UTF-8` encoded, to `ISO-8859-1`.

See `utf8_encode` for an explaination of UTF-8 encoding.

utf8_encode

Name

`utf8_encode` — encodes an ISO-8859-1 string to UTF-8

Description

`string utf8_encode(string data);`

This function encodes the string *`data`* to `UTF-8`, and returns the encoded version. `UTF-8` is a standard mechanism used by Unicodefor encoding *wide character* values into a byte stream. `UTF-8` is transparent to plain ASCII characters, is self-synchronized (meaning it is possible for a program to figure out where in the bytestream characters start) and can be used with normal string comparison functions for sorting and such. PHP encodes `UTF-8` characters in up to four bytes, like this:

Table 1. UTF-8 encoding

bytes	bits	representation
1	7	0bbbbbbb
2	11	110bbbbb 10bbbbbb
3	16	1110bbbb 10bbbbbb 10bbbbbb
4	21	11110bbb 10bbbbbb 10bbbbbb 10bbbbbb

Each *`b`* represents a bit that can be used to store character data.

V. Appendixes

Appendix A. Migrating from PHP/FI 2.0 to PHP 3.0

About the incompatbilities in 3.0

PHP 3.0 is rewritten from the ground up. It has a proper parser that is much more robust and consistent than 2.0's. 3.0 is also significantly faster, and uses less memory. However, some of these improvements have not been possible without compatibility changes, both in syntax and functionality.

In addition, PHP's developers have tried to clean up both PHP's syntax and semantics in version 3.0, and this has also caused some incompatibilities. In the long run, we believe that these changes are for the better.

This chapter will try to guide you through the incompatibilities you might run into when going from PHP/FI 2.0 to PHP 3.0 and help you resolve them. New features are not mentioned here unless necessary.

A conversion program that can automatically convert your old PHP/FI 2.0 scripts exists. It can be found in the `convertor` subdirectory of the PHP 3.0 distribution. This program only catches the syntax changes though, so you should read this chapter carefully anyway.

Start/end tags

The first thing you probably will notice is that PHP's start and end tags have changed. The old `<? >` form has been replaced by three new possible forms:

Example A-1. Migration: old start/end tags

```
<? echo "This is PHP/FI 2.0 code.\n"; >
```

As of version 2.0, PHP/FI also supports this variation:

Example A-2. Migration: first new start/end tags

```
<? echo "This is PHP 3.0 code!\n"; ?>
```

Notice that the end tag now consists of a question mark and a greater-than character instead of just greater-than. However, if you plan on using XML on your server, you will get problems with the first new variant, because PHP may try to execute the XML markup in XML documents as PHP code. Because of this, the following variation was introduced:

Example A-3. Migration: second new start/end tags

```
<?php echo "This is PHP 3.0 code!\n"; ?>
```

Some people have had problems with editors that don't understand the processing instruction tags at all. Microsoft FrontPage is one such editor, and as a workaround for these, the following variation was introduced as well:

Example A-4. Migration: third new start/end tags

```
<script language="php">

  echo "This is PHP 3.0 code!\n";

</script>
```

if..endif syntax

The 'alternative' way to write if/elseif/else statements, using if(); elseif(); else; endif; cannot be efficiently implemented without adding a large amount of complexity to the 3.0 parser. Because of this, the syntax has been changed:

Example A-5. Migration: old if..endif syntax

```
if ($foo);
    echo "yep\n";
elseif ($bar);
    echo "almost\n";
else;
    echo "nope\n";
endif;
```

Example A-6. Migration: new if..endif syntax

```
if ($foo):
    echo "yep\n";
```

```
elseif ($bar):
    echo "almost\n";
else:
    echo "nope\n";
endif;
```

Notice that the semicolons have been replaced by colons in all statements but the one terminating the expression (endif).

while syntax

Just like with if..endif, the syntax of while..endwhile has changed as well:

Example A-7. Migration: old while..endwhile syntax

```
while ($more_to_come);
    ...
endwhile;
```

Example A-8. Migration: new while..endwhile syntax

```
while ($more_to_come):
    ...
endwhile;
```

> **Warning**
>
> If you use the old while..endwhile syntax in PHP 3.0, you will get a never-ending loop.

Expression types

PHP/FI 2.0 used the left side of expressions to determine what type the result should be. PHP 3.0 takes both sides into account when determining result types, and this may cause 2.0 scripts to behave unexpectedly in 3.0.

Consider this example:

```
$a[0]=5;
$a[1]=7;

$key = key($a);
while ("" != $key) {
    echo "$keyn";
    next($a);
}
```

In PHP/FI 2.0, this would display both of $a's indices. In PHP 3.0, it wouldn't display anything. The reason is that in PHP 2.0, because the left argument's type was string, a string comparison was made, and indeed `""` does not equal `"0"`, and the loop went through. In PHP 3.0, when a string is compared with an integer, an integer comparison is made (the string is converted to an integer). This results in comparing `atoi("")` which is `0`, and `variablelist` which is also `0`, and since `0==0`, the loop doesn't go through even once.

The fix for this is simple. Replace the while statement with:

```
while ((string)$key != "") {
```

Error messages have changed

PHP 3.0's error messages are usually more accurate than 2.0's were, but you no longer get to see the code fragment causing the error. You will be supplied with a file name and a line number for the error, though.

Short-circuited boolean evaluation

In PHP 3.0 boolean evaluation is short-circuited. This means that in an expression like `(1 || test_me())`, the function `test_me` would not be executed since nothing can change the result of the expression after the `1`.

This is a minor compatibility issue, but may cause unexpected side-effects.

Function true/false return values

Most internal functions have been rewritten so they return TRUE when successful and FALSE when failing, as opposed to 0 and -1 in PHP/FI 2.0, respectively. The new behaviour allows for more logical code, like `$fp = fopen("/your/file") or fail("darn!");`. Because PHP/FI 2.0 had no clear rules for what functions should return when they failed, most such scripts will probably have to be checked manually after using the 2.0 to 3.0 convertor.

Example A-9. Migration from 2.0: return values, old code

```
$fp = fopen($file, "r");
if ($fp == -1);
    echo("Could not open $file for reading<br>\n");
endif;
```

Example A-10. Migration from 2.0: return values, new code

```
$fp = @fopen($file, "r") or print("Could not open $file for reading<br>\n");
```

Other incompatibilities

- The PHP 3.0 Apache module no longer supports Apache versions prior to 1.2. Apache 1.2 or later is required.
- `echo` no longer supports a format string. Use the `printf` function instead.
- In PHP/FI 2.0, an implementation side-effect caused `$foo[0]` to have the same effect as `$foo`. This is not true for PHP 3.0.
- Reading arrays with `$array[]` is no longer supported

  That is, you cannot traverse an array by having a loop that does `$data = $array[]`. Use `current` and `next` instead.

  Also, `$array1[] = $array2` does not append the values of `$array2` to `$array1`, but appends `$array2` as the last entry of `$array1`. See also multidimensional array support.

- `"+"` is no longer overloaded as a concatenation operator for strings, instead it converts it's arguments to numbers and performs numeric addition. Use `"."` instead.

Example A-11. Migration from 2.0: concatenation for strings

```
echo "1" + "1";
```

In PHP 2.0 this would echo 11, in PHP 3.0 it would echo 2. Instead use:

```
echo "1"."1";
$a = 1;
$b = 1;
echo $a + $b;
```

This would echo 2 in both PHP 2.0 and 3.0.

```
$a = 1;
$b = 1;
echo $a.$b;
```

This will echo 11 in PHP 3.0.

Appendix B. PHP development

Adding functions to PHP3

Function Prototype

All functions look like this:

```
void php3_foo(INTERNAL_FUNCTION_PARAMETERS) {

}
```

Even if your function doesn't take any arguments, this is how it is called.

Function Arguments

Arguments are always of type pval. This type contains a union which has the actual type of the argument. So, if your function takes two arguments, you would do something like the following at the top of your function:

Example B-1. Fetching function arguments

```
pval *arg1, *arg2;
if (ARG_COUNT(ht) != 2 || getParameters(ht,2,&arg1,&arg2)==FAILURE) {
   WRONG_PARAM_COUNT;
}
```

NOTE: Arguments can be passed either by value or by reference. In both cases you will need to pass &(pval *) to getParameters. If you want to check if the n'th parameter was sent to you by reference or not, you can use the function, ParameterPassedByReference(ht,n). It will return either 1 or 0.

When you change any of the passed parameters, whether they are sent by reference or by value, you can either start over with the parameter by calling pval_destructor on it, or if it's an ARRAY you want to add to, you can use functions similar to the ones in internal_functions.h which manipulate return_value as an ARRAY.

Also if you change a parameter to IS_STRING make sure you first assign the new estrdup()'ed string and the string length, and only later change the type to IS_STRING. If you change the string of a parameter which already IS_STRING or IS_ARRAY you should run pval_destructor on it first.

Variable Function Arguments

A function can take a variable number of arguments. If your function can take either 2 or 3 arguments, use the following:

Example B-2. Variable function arguments

```
pval *arg1, *arg2, *arg3;
int arg_count = ARG_COUNT(ht);

if (arg_count < 2 || arg_count > 3 ||
    getParameters(ht,arg_count,&arg1,&arg2,&arg3)==FAILURE) {
    WRONG_PARAM_COUNT;
}
```

Using the Function Arguments

The type of each argument is stored in the pval type field. This type can be any of the following:

Table B-1. PHP Internal Types

IS_STRING	String
IS_DOUBLE	Double-precision floating point
IS_LONG	Long integer
IS_ARRAY	Array
IS_EMPTY	None
IS_USER_FUNCTION	??
IS_INTERNAL_FUNCTION	?? (if some of these cannot be passed to a function - delete)
IS_CLASS	??

IS_OBJECT	??

If you get an argument of one type and would like to use it as another, or if you just want to force the argument to be of a certain type, you can use one of the following conversion functions:

```
convert_to_long(arg1);
convert_to_double(arg1);
convert_to_string(arg1);
convert_to_boolean_long(arg1); /* If the string is "" or "0" it be-
comes 0, 1 otherwise */
convert_string_to_number(arg1);  /* Converts string to either LONG or DOU-
BLE depending on string */
```

These function all do in-place conversion. They do not return anything.

The actual argument is stored in a union; the members are:

- IS_STRING: arg1->value.str.val
- IS_LONG: arg1->value.lval
- IS_DOUBLE: arg1->value.dval

Memory Management in Functions

Any memory needed by a function should be allocated with either emalloc() or estrdup(). These are memory handling abstraction functions that look and smell like the normal malloc() and strdup() functions. Memory should be freed with efree().

There are two kinds of memory in this program: memory which is returned to the parser in a variable, and memory which you need for temporary storage in your internal function. When you assign a string to a variable which is returned to the parser you need to make sure you first allocate the memory with either emalloc() or estrdup(). This memory should NEVER be freed by you, unless you later in the same function overwrite your original assignment (this kind of programming practice is not good though).

For any temporary/permanent memory you need in your functions/library you should use the three emalloc(), estrdup(), and efree() functions. They behave EXACTLY like their counterpart functions. Anything you emalloc() or estrdup() you have to efree() at some point or another, unless it’s supposed to stick around until the end of the program; otherwise, there will be a memory leak. The meaning of "the functions behave exactly like their counterparts" is: if you efree() something which was not emalloc()’ed nor estrdup()’ed you might get a segmentation fault. So please take care and free all of your wasted memory.

If you compile with "-DDEBUG", PHP3 will print out a list of all memory that was allocated using emalloc() and estrdup() but never freed with efree() when it is done running the specified script.

Setting Variables in the Symbol Table

A number of macros are available which make it easier to set a variable in the symbol table:

- SET_VAR_STRING(name,value) [1]
- SET_VAR_DOUBLE(name,value)
- SET_VAR_LONG(name,value)

[1]

Symbol tables in PHP 3.0 are implemented as hash tables. At any given time, &symbol_table is a pointer to the ’main’ symbol table, and active_symbol_table points to the currently active symbol table (these may be identical like in startup, or different, if you’re inside a function).

The following examples use ’active_symbol_table’. You should replace it with &symbol_table if you specifically want to work with the ’main’ symbol table. Also, the same functions may be applied to arrays, as explained below.

Example B-3. Checking whether $foo exists in a symbol table

```
if (hash_exists(active_symbol_table,"foo",sizeof("foo"))) { exists... }
else { doesn’t exist }
```

Example B-4. Finding a variable’s size in a symbol table

```
hash_find(active_symbol_table,"foo",sizeof("foo"),&pvalue);
check(pvalue.type);
```

Arrays in PHP 3.0 are implemented using the same hashtables as symbol tables. This means the two above functions can also be used to check variables inside arrays.

If you want to define a new array in a symbol table, you should do the following.

First, you may want to check whether it exists and abort appropiately, using hash_exists() or hash_find().

Next, initialize the array:

Example B-5. Initializing a new array

```
pval arr;

if (array_init(&arr) == FAILURE) { failed... };
hash_update(active_symbol_table,"foo",sizeof("foo"),&arr,sizeof(pval),NULL);
```

This code declares a new array, named $foo, in the active symbol table. This array is empty.

Here's how to add new entries to it:

Example B-6. Adding entries to a new array

```
pval entry;

entry.type = IS_LONG;
entry.value.lval = 5;

/* defines $foo["bar"] = 5 */
hash_update(arr.value.ht,"bar",sizeof("bar"),&entry,sizeof(pval),NULL);

/* defines $foo[7] = 5 */
hash_index_update(arr.value.ht,7,&entry,sizeof(pval),NULL);

/* defines the next free place in $foo[],
 * $foo[8], to be 5 (works like php2)
 */
hash_next_index_insert(arr.value.ht,&entry,sizeof(pval),NULL);
```

If you'd like to modify a value that you inserted to a hash, you must first retrieve it from the hash. To prevent that overhead, you can supply a pval ** to the hash add function, and it'll be updated with the pval * address of the inserted element inside the hash. If that value is NULL (like in all of the above examples) - that parameter is ignored.

hash_next_index_insert() uses more or less the same logic as "$foo[] = bar;" in PHP 2.0.

If you are building an array to return from a function, you can initialize the array just like above by doing:

```
if (array_init(return_value) == FAILURE) { failed...; }
```

...and then adding values with the helper functions:

```
add_next_index_long(return_value,long_value);
add_next_index_double(return_value,double_value);
add_next_index_string(return_value,estrdup(string_value));
```

Of course, if the adding isn’t done right after the array initialization, you’d probably have to look for the array first:

```
pval *arr;

if (hash_find(active_symbol_table,"foo",sizeof("foo"),(void **)&arr)==FAILURE) { can’t find
else { use arr->value.ht... }
```

Note that hash_find receives a pointer to a pval pointer, and not a pval pointer.

Just about any hash function returns SUCCESS or FAILURE (except for hash_exists(), which returns a boolean truth value).

Returning simple values

A number of macros are available to make returning values from a function easier.

The RETURN_* macros all set the return value and return from the function:

- RETURN
- RETURN_FALSE
- RETURN_TRUE
- RETURN_LONG(l)
- RETURN_STRING(s,dup) If dup is true, duplicates the string
- RETURN_STRINGL(s,l,dup) Return string (s) specifying length (l).
- RETURN_DOUBLE(d)

The RETVAL_* macros set the return value, but do not return.

- RETVAL_FALSE

- RETVAL_TRUE
- RETVAL_LONG(l)
- RETVAL_STRING(s,dup) If dup is true, duplicates the string
- RETVAL_STRINGL(s,l,dup) Return string (s) specifying length (l).
- RETVAL_DOUBLE(d)

The string macros above will all estrdup() the passed ’s’ argument, so you can safely free the argument after calling the macro, or alternatively use statically allocated memory.

If your function returns boolean success/error responses, always use RETURN_TRUE and RETURN_FALSE respectively.

Returning complex values

Your function can also return a complex data type such as an object or an array.

Returning an object:

1. Call object_init(return_value).
2. Fill it up with values. The functions available for this purpose are listed below.
3. Possibly, register functions for this object. In order to obtain values from the object, the function would have to fetch "this" from the active_symbol_table. Its type should be IS_OBJECT, and it’s basically a regular hash table (i.e., you can use regular hash functions on .value.ht). The actual registration of the function can be done using:

   ```
   add_method( return_value, function_name, function_ptr );
   ```

The functions used to populate an object are:

- add_property_long(return_value, property_name, l) - Add a property named ’property_name’, of type long, equal to ’l’
- add_property_double(return_value, property_name, d) - Same, only adds a double
- add_property_string(return_value, property_name, str) - Same, only adds a string
- add_property_stringl(return_value, property_name, str, l) - Same, only adds a string of length ’l’

Returning an array:

1. Call array_init(return_value).
2. Fill it up with values. The functions available for this purpose are listed below.

The functions used to populate an array are:

- add_assoc_long(return_value,key,l) - add associative entry with key 'key' and long value 'l'
- add_assoc_double(return_value,key,d)
- add_assoc_string(return_value,key,str,duplicate)
- add_assoc_stringl(return_value,key,str,length,duplicate) specify the string length
- add_index_long(return_value,index,l) - add entry in index 'index' with long value 'l'
- add_index_double(return_value,index,d)
- add_index_string(return_value,index,str)
- add_index_stringl(return_value,index,str,length) - specify the string length
- add_next_index_long(return_value,l) - add an array entry in the next free offset with long value 'l'
- add_next_index_double(return_value,d)
- add_next_index_string(return_value,str)
- add_next_index_stringl(return_value,str,length) - specify the string length

Using the resource list

PHP 3.0 has a standard way of dealing with various types of resources. This replaces all of the local linked lists in PHP 2.0.

Available functions:

- php3_list_insert(ptr, type) - returns the 'id' of the newly inserted resource
- php3_list_delete(id) - delete the resource with the specified id
- php3_list_find(id,*type) - returns the pointer of the resource with the specified id, updates 'type' to the resource's type

Typically, these functions are used for SQL drivers but they can be used for anything else; for instance, maintaining file descriptors.

Typical list code would look like this:

Example B-7. Adding a new resource

```
RESOURCE *resource;

/* ...allocate memory for resource and acquire resource... */
/* add a new resource to the list */
return_value-
>value.lval = php3_list_insert((void *) resource, LE_RESOURCE_TYPE);
return_value->type = IS_LONG;
```

Example B-8. Using an existing resource

```
pval *resource_id;
RESOURCE *resource;
int type;

convert_to_long(resource_id);
resource = php3_list_find(resource_id->value.lval, &type);
if (type != LE_RESOURCE_TYPE) {
php3_error(E_WARNING,"resource index %d has the wrong type",resource_id-
>value.lval);
RETURN_FALSE;
}
/* ...use resource... */
```

Example B-9. Deleting an existing resource

```
pval *resource_id;
RESOURCE *resource;
int type;

convert_to_long(resource_id);
php3_list_delete(resource_id->value.lval);
```

The resource types should be registered in php3_list.h, in enum list_entry_type. In addition, one should add shutdown code for any new resource type defined, in list.c’s list_entry_destructor() (even if you don’t have anything to do on shutdown, you must add an empty case).

Using the persistent resource table

PHP 3.0 has a standard way of storing persistent resources (i.e., resources that are kept in between hits). The first module to use this feature was the MySQL module, and mSQL followed it, so one can get the general impression of how a persistent resource should be used by reading mysql.c. The functions you should look at are:

php3_mysql_do_connect
php3_mysql_connect()
php3_mysql_pconnect()

The general idea of persistence modules is this:

1. Code all of your module to work with the regular resource list mentioned in section (9).
2. Code extra connect functions that check if the resource already exists in the persistent resource list. If it does, register it as in the regular resource list as a pointer to the persistent resource list (because of 1., the rest of the code should work immediately). If it doesn't, then create it, add it to the persistent resource list AND add a pointer to it from the regular resource list, so all of the code would work since it's in the regular resource list, but on the next connect, the resource would be found in the persistent resource list and be used without having to recreate it. You should register these resources with a different type (e.g. LE_MYSQL_LINK for non-persistent link and LE_MYSQL_PLINK for a persistent link).

If you read mysql.c, you'll notice that except for the more complex connect function, nothing in the rest of the module has to be changed.

The very same interface exists for the regular resource list and the persistent resource list, only 'list' is replaced with 'plist':

- php3_plist_insert(ptr, type) - returns the 'id' of the newly inserted resource
- php3_plist_delete(id) - delete the resource with the specified id
- php3_plist_find(id,*type) - returns the pointer of the resource with the specified id, updates 'type' to the resource's type

However, it's more than likely that these functions would prove to be useless for you when trying to implement a persistent module. Typically, one would want to use the fact that the persistent resource list is really a hash table. For instance, in the MySQL/mSQL modules, when there's a pconnect() call (persistent connect), the function builds a string out of the host/user/passwd that were passed to the function, and hashes the SQL link with this string as a key. The next time someone calls a pconnect() with the same host/user/passwd, the same key would be generated, and the function would find the SQL link in the persistent list.

Until further documented, you should look at mysql.c or msql.c to see how one should use the plist's hash table abilities.

One important thing to note: resources going into the persistent resource list must *NOT* be allocated with PHP's memory manager, i.e., they should NOT be created with emalloc(), estrdup(), etc. Rather, one should use the regular malloc(), strdup(), etc. The reason for this is simple - at the end of the request (end of the hit), every memory chunk that was allocated using PHP's memory manager is deleted. Since the persistent list isn't supposed to be erased at the end of a request, one mustn't use PHP's memory manager for allocating resources that go to it.

When you register a resource that's going to be in the persistent list, you should add destructors to it both in the non-persistent list and in the persistent list. The destructor in the non-persistent list destructor shouldn't do anything. The one in the persistent list destructor should properly free any resources obtained by that type (e.g. memory, SQL links, etc). Just like with the non-persistent resources, you *MUST* add destructors for every resource, even it requires no destructotion and the destructor would be empty. Remember, since emalloc() and friends aren't to be used in conjunction with the persistent list, you mustn't use efree() here either.

Adding runtime configuration directives

Many of the features of PHP3 can be configured at runtime. These configuration directives can appear in either the designated php3.ini file, or in the case of the Apache module version in the Apache .conf files. The advantage of having them in the Apache .conf files is that they can be configured on a per-directory basis. This means that one directory may have a certain safemodeexecdir for example, while another directory may have another. This configuration granularity is especially handy when a server supports multiple virtual hosts.

The steps required to add a new directive:

1. Add directive to php3_ini_structure struct in mod_php3.h.
2. In main.c, edit the php3_module_startup function and add the appropriate cfg_get_string() or cfg_get_long() call.
3. Add the directive, restrictions and a comment to the php3_commands structure in mod_php3.c. Note the restrictions part. RSRC_CONF are directives that can only be present in the actual Apache .conf files. Any OR_OPTIONS directives can be present anywhere, include normal .htaccess files.
4. In either php3take1handler() or php3flaghandler() add the appropriate entry for your directive.
5. In the configuration section of the _php3_info() function in functions/info.c you need to add your new directive.

6. And last, you of course have to use your new directive somewhere. It will be addressable as php3_ini.directive.

Calling User Functions

To call user functions from an internal function, you should use the `call_user_function` function.

`call_user_function` returns SUCCESS on success, and FAILURE in case the function cannot be found. You should check that return value! If it returns SUCCESS, you are responsible for destroying the retval pval yourself (or return it as the return value of your function). If it returns FAILURE, the value of retval is undefined, and you mustn't touch it.

All internal functions that call user functions *must* be reentrant. Among other things, this means they must not use globals or static variables.

`call_user_function` takes six arguments:

HashTable *function_table

This is the hash table in which the function is to be looked up.

pval *object

This is a pointer to an object on which the function is invoked. This should be NULL if a global function is called. If it's not NULL (i.e. it points to an object), the function_table argument is ignored, and instead taken from the object's hash. The object *may* be modified by the function that is invoked on it (that function will have access to it via $this). If for some reason you don't want that to happen, send a copy of the object instead.

pval *function_name

The name of the function to call. Must be a pval of type IS_STRING with function_name.str.val and function_name.str.len set to the appropriate values. The function_name is modified by call_user_function() - it's converted to lowercase. If you need to preserve the case, send a copy of the function name instead.

pval *retval

A pointer to a pval structure, into which the return value of the invoked function is saved. The structure must be previously allocated - `call_user_function` does NOT allocate it by itself.

int param_count

The number of parameters being passed to the function.

pval *params[]

An array of pointers to values that will be passed as arguments to the function, the first argument being in offset 0, the second in offset 1, etc. The array is an array of pointers to pval's; The pointers are sent as-is to the function, which means if the function modifies its arguments, the original values are changed (passing by reference). If you don't want that behavior, pass a copy instead.

Reporting Errors

To report errors from an internal function, you should call the `php3_error` function. This takes at least two parameters – the first is the level of the error, the second is the format string for the error message (as in a standard `printf` call), and any following arguments are the parameters for the format string. The error levels are:

E_NOTICE

Notices are not printed by default, and indicate that the script encountered something that could indicate an error, but could also happen in the normal course of running a script. For example, trying to access the value of a variable which has not been set, or calling `stat` on a file that doesn't exist.

E_WARNING

Warnings are printed by default, but do not interrupt script execution. These indicate a problem that should have been trapped by the script before the call was made. For example, calling `ereg` with an invalid regular expression.

E_ERROR

Errors are also printed by default, and execution of the script is halted after the function returns. These indicate errors that can not be recovered from, such as a memory allocation problem.

E_PARSE

Parse errors should only be generated by the parser. The code is listed here only for the sake of completeness.

E_CORE_ERROR

This is like an E_ERROR, except it is generated by the core of PHP. Functions should not generate this type of error.

E_CORE_WARNING

This is like an E_WARNING, except it is generated by the core of PHP. Functions should not generate this type of error.

Notes

Be careful here. The value part must be malloc'ed manually because the memory management code will try to free this pointer later. Do not pass statically allocated memory into a SET_VAR_STRING.

Appendix C. The PHP Debugger

Using the Debugger

PHP’s internal debugger is useful for tracking down evasive bugs. The debugger works by connecting to a TCP port for every time PHP starts up. All error messages from that request will be sent to this TCP connection. This information is intended for "debugging server" that can run inside an IDE or programmable editor (such as Emacs).

How to set up the debugger:

1. Set up a TCP port for the debugger in the configuration file (debugger.port) and enable it (debugger.enabled).
2. Set up a TCP listener on that port somewhere (for example **socket -l -s 1400** on UNIX).
3. In your code, run "debugger_on(*`host`*)", where *`host`* is the IP number or name of the host running the TCP listener.

Now, all warnings, notices etc. will show up on that listener socket, *even if you them turned off with `error_reporting`*.

Debugger Protocol

The debugger protocol is line-based. Each line has a *type*, and several lines compose a *message*. Each message starts with a line of the type `start` and terminates with a line of the type `end`. PHP may send lines for different messages simultaneously.

A line has this format:

`date` *`time`* *`host`*(*`pid`*) *`type`*: *`message-data`*

`date`

Date in ISO 8601 format (*`yyyy`*-*`mm`*-*`dd`*)

`time`

Time including microseconds: *`hh`*:*`mm`*:*`uuuuuu`* *`host`*

DNS name or IP address of the host where the script error was generated.

`pid`

PID (process id) on *`host`* of the process with the PHP script that generated this error.

`type`

Type of line. Tells the receiving program about what it should treat the following data as:

Table C-1. Debugger Line Types

Name	Meaning
`start`	Tells the receiving program that a debugger message starts here. The contents of *`data will be the type of error message, listed below.`*
`message`	The PHP error message.
`location`	File name and line number where the error occured. The first `location line will always contain the top-level location.` *`data will contain file:line. There will always be a location line after message and after every function.`*
`frames`	Number of frames in the following stack dump. If there are four frames, expect information about four levels of called functions. If no "frames" line is given, the depth should be assumed to be 0 (the error occured at top-level).
`function`	Name of function where the error occured. Will be repeated once for every level in the function call stack.
`end`	Tells the receiving program that a debugger message ends here.

`data`

Line data.

Table C-2. Debugger Error Types

Debugger	PHP Internal
warning	E_WARNING
error	E_ERROR
parse	E_PARSE
notice	E_NOTICE
core-error	E_CORE_ERROR
core-warning	E_CORE_WARNING
unknown	(any other)

Example C-1. Example Debugger Message

```
1998-04-05 23:27:400966 lucifer.guardian.no(20481) start: notice
1998-04-05 23:27:400966 lucifer.guardian.no(20481) message: Uninitialized variable
1998-04-05 23:27:400966 lucifer.guardian.no(20481) location: (null):7
1998-04-05 23:27:400966 lucifer.guardian.no(20481) frames: 1
1998-04-05 23:27:400966 lucifer.guardian.no(20481) function: display
1998-04-05 23:27:400966 lucifer.guardian.no(20481) location: /home/ssb/public_html/test.php3:10
1998-04-05 23:27:400966 lucifer.guardian.no(20481) end: notice
```