PHP Manual

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Zeev Suraski

Edited by

Stig Sæther Bakken

Egon Schmid

PHP Manual

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

by

Edited by Stig Sæther Bakken

Edited by Egon Schmid

Published Tue May 23 18:07:18 CEST 2000

Copyright

Table of Contents

List of Tables

List of Figures

List of Examples

Preface

PHP, which stands for "PHP: Hypertext Preprocessor", 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 XML using the DocBook XML DTD (http://www.nwalsh.com/docbook/xml/), 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 is maintained by Stig Sæther Bakken (mailto:stig@php.net).

Daily HTML snapshots of the manual, including translations, can be found at http://snaps.php.net/manual/.

I. Getting Started

Chapter 1. Introduction

What is PHP?

PHP (officially "PHP: Hypertext Preprocessor") 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	InterBase	Solid
dBase	mSQL	Sybase
Empress	MySQL	Velocis
FilePro	Oracle	Unix dbm
Informix	PostgreSQL	

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 (end-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 (http://www.netcraft.com/) (see also Netcraft Web Server Survey (http://www.netcraft.com/survey/)) would be that PHP is in use on over 1,000,000 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/) or iODBC home page (http://www.iodbc.org)

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.ini-dist' 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://php.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

On by default. If changed to Off scripts will be terminated as soon as they try to output something after a client has aborted their connection. **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.default_host string

The default server host to use when connecting to the database server if no other host is specified.

mysql.default_user string

The default user name to use when connecting to the database server if no other name is specified.

mysql.default_password string

The default password to use when connecting to the database server if no other password is specified.

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:

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

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. When a string is enclosed in single quotes, the only escapes that will be understood are "\\" and "\'". This is for convenience, so that you can have single-quotes and backslashes in a single-quoted string. Variables will *not* be expanded inside a single-quoted string.

Another way to delimit strings is by using here doc syntax ("«<"). One should provide an identifier after «<, then the string, and then the same identifier to close the quotation. The closing identifier *must* begin in the first column of the line.

Example 6-1. Here doc string quoting example

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

Note: Here doc support was added in PHP 4.

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

Characters within strings may be accessed by treating the string as a numerically-indexed array of characters, using C-like syntax. See below for examples.

Example 6-2. Some string examples

```
<?php
/* Assigning a string. */
$str = "This is a string";

/* Appending to it. */
$str = $str . " with some more text";

/* Another way to append, includes an escaped newline. */
```

```
$str .= " and a newline at the end.\n";

/* This string will end up being '<p>Number: 9</p>' */
$num = 9;
$str = "<p>Number: $num</p>";

/* This one will be '<p>Number: $num</p>' */
$num = 9;
$str = '<p>Number: $num</p>';

/* Get the first character of a string  */
$str = 'This is a test.';
$first = $str[0];

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

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).

If you would like to test any of the examples in this section, you can cut and paste the examples and insert the following line to see for yourself what's going on:

```
echo "\$foo==$foo; type is " . gettype ($foo) . "<br>\n";
```

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. When you assign a value to an array variable using empty brackets, the value will be added onto the end of 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 associative indices)
$a[3]["bar"] = $f;            # (you can mix numeric and associative indices)

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

In PHP3 it is not possible to reference multidimensional arrays directly within strings. For instance, the following will not have the desired result:

```
$a[3]['bar'] = 'Bob';
echo "This won't work: $a[3][bar]";
```

In PHP3, the above will output `This won't work: Array[bar]`. The string concatenation operator, however, can be used to overcome this:

```
$a[3][’bar’] = ’Bob’;
echo "This will work: " . $a[3][bar];
```

In PHP4, however, the whole problem may be circumvented by enclosing the array reference (inside the string) in curly braces:

```
$a[3][’bar’] = ’Bob’;
echo "This will work: {$a[3][bar]}";
```

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"  => "tart",
          "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.

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

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

For a full discussion, please read the section Classes and Objects.

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()**.

If you would like to test any of the examples in this section, you can cut and paste the examples and insert the following line to see for yourself what's going on:

```
echo "\$foo==$foo; type is " . gettype ($foo) . "<br>\n";
```

Note: The behaviour of an automatic conversion to array is currently undefined.

```
$a = 1;       // $a is an integer
$a[0] = "f";  // $a becomes an array, with $a[0] holding "f"
```

While the above example may seem like it should clearly result in $a becoming an array, the first element of which is 'f', consider this:

```
$a = "1";     // $a is a string
$a[0] = "f";  // What about string offsets? What happens?
```

Since PHP supports indexing into strings via offsets using the same syntax as array indexing, the example above leads to a problem: should $a become an array with its first element being "f", or should "f" become the first character of the string $a?

For this reason, as of PHP 3.0.12 and PHP 4.0b3-RC4, the result of this automatic conversion is considered to be undefined. Fixes are, however, being discussed.

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.

When casting from a scalar or a string variable to an array, the variable will become the first element of the array:

```
$var = 'ciao';
$arr = (array) $var;
echo $arr[0];  // outputs 'ciao'
```

When casting from a scalar or a string variable to an object, the variable will become an attribute of the object; the attribute name will be 'scalar':

```
$var = 'ciao';
$obj = (object) $var;
echo $obj->scalar;  // outputs 'ciao'
```

Chapter 7. Variables

Basics

Variables in PHP are represented by a dollar sign followed by the name of the variable. The variable name is case-sensitive.

Variable names follow the same rules as other labels in PHP. A valid variable name starts with a letter or underscore, followed by any number of letters, numbers, or underscores. As a regular expression, it would be expressed thus: '[a-zA-Z_\x7f-\xff][a-zA-Z0-9_\x7f-\xff]*'

> **Note:** For our purposes here, a letter is a-z, A-Z, and the ASCII characters from 127 through 255 (0x7f-0xff).

```
$var = "Bob";
$Var = "Joe";
echo "$var, $Var";      // outputs "Bob, Joe"

$4site = 'not yet';     // invalid; starts with a number
$_4site = 'not yet';    // valid; starts with an underscore
$täyte = 'mansikka';    // valid; 'ä' is ASCII 228.
```

In PHP3, variables are always assigned by value. That is to say, when you assign an expression to a variable, the entire value of the original expression is copied into the destination variable. This means, for instance, that after assigning one variable's value to another, changing one of those variables will have no effect on the other. For more information on this kind of assignment, see Expressions.

PHP4 offers another way to assign values to variables: *assign by reference*. This means that the new variable simply references (in other words, "becomes an alias for" or "points to") the original variable. Changes to the new variable affect the original, and vice versa. This also means that no copying is performed; thus, the assignment happens more quickly. However, any speedup will likely be noticed only in tight loops or when assigning large arrays or objects.

To assign by reference, simply prepend an ampersand (&) to the beginning of the variable which is being assigned (the source variable). For instance, the following code snippet outputs 'My name is Bob' twice:

```
<?php
$foo = 'Bob';              // Assign the value 'Bob' to $foo
$bar = &$foo;              // Reference $foo via $bar.
$bar = "My name is $bar";  // Alter $bar...
echo $foo;                 // $foo is altered too.
echo $bar;
?>
```

One important thing to note is that only named variables may be assigned by reference.

```
<?php
$foo = 25;
$bar = &$foo;      // This is a valid assignment.
$bar = &(24 * 7);  // Invalid; references an unnamed expression.
```

```
function test() {
   return 25;
}

$bar = &test();    // Invalid.
?>
```

Predefined variables

PHP provides a large number of predefined variables to any script which it runs. Many of these variables, however, cannot be fully documented as they are dependent upon which server is running, the version and setup of the server, and other factors. Some of these variables will not be available when PHP is run on the command-line.

Despite these factors, here is a list of predefined variables available under a stock installation of PHP 3 running as a module under a stock installation of Apache (http://www.apache.org/) 1.3.6.

For a list of all predefined variables (and lots of other useful information), please see (and use) **phpinfo()**.

> **Note:** This list is neither exhaustive nor intended to be. It is simply a guideline as to what sorts of predefined variables you can expect to have access to in your script.

Apache variables

These variables are created by the Apache (http://www.apache.org/) webserver. If you are running another webserver, there is no guarantee that it will provide the same variables; it may omit some, or provide others not listed here. That said, a large number of these variables are accounted for in the CGI 1.1 specification (http://hoohoo.ncsa.uiuc.edu/cgi/env.html), so you should be able to expect those.

Note that few, if any, of these will be available (or indeed have any meaning) if running PHP on the command line.

GATEWAY_INTERFACE

What revision of the CGI specification the server is using; i.e. 'CGI/1.1'.

SERVER_NAME

The name of the server host under which the current script is executing. If the script is running on a virtual host, this will be the value defined for that virtual host.

SERVER_SOFTWARE

Server identification string, given in the headers when responding to requests.

SERVER_PROTOCOL

Name and revision of the information protocol via which the page was requested; i.e. 'HTTP/1.0';

REQUEST_METHOD

Which request method was used to access the page; i.e. 'GET', 'HEAD', 'POST', 'PUT'.

QUERY_STRING

The query string, if any, via which the page was accessed.

DOCUMENT_ROOT

The document root directory under which the current script is executing, as defined in the server's configuration file.

HTTP_ACCEPT

Contents of the `Accept:` header from the current request, if there is one.

HTTP_ACCEPT_CHARSET

Contents of the `Accept-Charset:` header from the current request, if there is one. Example: 'iso-8859-1,*,utf-8'.

HTTP_ENCODING

Contents of the `Accept-Encoding:` header from the current request, if there is one. Example: 'gzip'.

HTTP_ACCEPT_LANGUAGE

Contents of the `Accept-Language:` header from the current request, if there is one. Example: 'en'.

HTTP_CONNECTION

Contents of the `Connection:` header from the current request, if there is one. Example: 'Keep-Alive'.

HTTP_HOST

Contents of the `Host:` header from the current request, if there is one.

HTTP_REFERER

The address of the page (if any) which referred the browser to the current page. This is set by the user's browser; not all browsers will set this.

HTTP_USER_AGENT

Contents of the `User_Agent:` header from the current request, if there is one. This is a string denoting the browser software being used to view the current page; i.e. `Mozilla/4.5 [en] (X11; U; Linux 2.2.9 i586)`. Among other things, you can use this value with **get_browser()** to tailor your page's functionality to the capabilities of the user's browser.

REMOTE_ADDR

The IP address from which the user is viewing the current page.

REMOTE_PORT

The port being used on the user's machine to communicate with the web server.

SCRIPT_FILENAME

The absolute pathname of the currently executing script.

SERVER_ADMIN

The value given to the SERVER_ADMIN (for Apache) directive in the web server configuration file. If the script is running on a virtual host, this will be the value defined for that virtual host.

SERVER_PORT

The port on the server machine being used by the web server for communication. For default setups, this will be ’80’; using SSL, for instance, will change this to whatever your defined secure HTTP port is.

SERVER_SIGNATURE

String containing the server version and virtual host name which are added to server-generated pages, if enabled.

PATH_TRANSLATED

Filesystem- (not document root-) based path to the current script, after the server has done any virtual-to-real mapping.

SCRIPT_NAME

Contains the current script’s path. This is useful for pages which need to point to themselves.

REQUEST_URI

The URI which was given in order to access this page; for instance, ’/index.html’.

Environment variables

These variables are imported into PHP’s global namespace from the environment under which the PHP parser is running. Many are provided by the shell under which PHP is running and different systems are likely running different kinds of shells, a definitive list is impossible. Please see your shell’s documentation for a list of defined environment variables.

Other environment variables include the CGI variables, placed there regardless of whether PHP is running as a server module or CGI processor.

PHP variables

These variables are created by PHP itself.

argv

Array of arguments passed to the script. When the script is run on the command line, this gives C-style access to the command line parameters. When called via the GET method, this will contain the query string.

argc

Contains the number of command line parameters passed to the script (if run on the command line).

PHP_SELF

The filename of the currently executing script, relative to the document root. If PHP is running as a command-line processor, this variable is not available.

HTTP_COOKIE_VARS

An associative array of variables passed to the current script via HTTP cookies. Only available if variable tracking has been turned on via either the track_vars configuration directive or the `<?php_track_vars?>` directive.

HTTP_GET_VARS

An associative array of variables passed to the current script via the HTTP GET method. Only available if variable tracking has been turned on via either the track_vars configuration directive or the `<?php_track_vars?>` directive.

HTTP_POST_VARS

An associative array of variables passed to the current script via the HTTP POST method. Only available if variable tracking has been turned on via either the track_vars configuration directive or the `<?php_track_vars?>` directive.

Variable scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. This single scope spans included and required files as well. For example:

```
$a = 1;
include "b.inc";
```

Here the $a variable will be available within the included b.inc script. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
$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 also provide one way to deal with recursive functions. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. You must make sure you have an adequate way of terminating the recursion. The following simple function recursively counts to 10, using the static variable $count to know when to stop:

```
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. i.e.

```
$$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";
```

i.e. they both produce: *hello world*.

In order to use variable variables with arrays, you have to resolve an ambiguity problem. That is, if you write $$a[1] then the parser needs to know if you meant to use $a[1] as a variable, or if you wanted $$a as the variable and then the [1] index from that variable. The syntax for resolving this ambiguity is: ${$a[1]} for the first case and ${$a}[1] for the second.

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
        <option value="stuttgarter">Stuttgarter Schwabenbräu
        </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.

Dots in incoming variable names

Typically, PHP does not alter the names of variables when they are passed into a script. However, it should be noted that the dot (period, full stop) is not a valid character in a PHP variable name. For the reason, look at it:

```
$varname.ext;  /* invalid variable name */
```

Now, what the parser sees is a variable named $varname, followed by the string concatenation operator, followed by the barestring (i.e. unquoted string which doesn't match any known key or reserved words) 'ext'. Obviously, this doesn't have the intended result.

For this reason, it is important to note that PHP will automatically replace any dots in incoming variable names with underscores.

Determining variable types

Because PHP determines the types of variables and converts them (generally) as needed, it is not always obvious what type a given variable is at any one time. PHP includes several functions which find out what type a variable is. 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.

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.

E_ALL

All of the E_* constants rolled into one. If used with **error_reporting()**, will cause any and all problems noticed by PHP to be reported.

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	Difference of $a and $b.
$a * $b	Multiplication	Product of $a and $b.
$a / $b	Division	Quotient of $a and $b.
$a % $b	Modulus	Remainder of $a divided by $b.

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!";
```

Note that the assignment copies the original variable to the new one (assignment by value), so changes to one will not affect the other. This may also have relevance if you need to copy something like a large array inside a tight loop. PHP4 supports assignment by reference, using the `$var = &$othervar;` syntax, but this is not possible in PHP3. 'Assignment by reference' means that both variables end up pointing at the same data, and nothing is copied anywhere.

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")

Comparison Operators

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

Table 10-3. Comparison Operators

example	name	result
$a == $b	Equal	True if $a is equal to $b.
$a === $b	Identical	True if $a is equal to $b, and they are of the same type. (PHP4 only)
$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 evaluates to *`expr2`* if *`expr1`* evaluates to true, and *`expr3`* if *`expr1`* evaluates to false.

Error control Operators

PHP supports one error control operator: the at sign (@). When prepended to an expression in PHP, any error messages that might be generated by that expression will be ignored.

If the track_errors feature is enabled, any error message generated by the expression will be saved in the global variable $php_errormsg. This variable will be overwritten on each error, so check early if you want to use it.

```
<?php
/* Intentional SQL error (extra quote): */
$res = @mysql_query( "select name, code from 'namelist" ) or
   die( "Query failed: error was '$php_errormsg'" );
?>
```

See also **error_reporting()**.

Execution Operators

PHP supports one execution operator: backticks (“). Note that these are not single-quotes! PHP will attempt to execute the contents of the backticks as a shell command; the output will be returned (i.e., it won’t simply be dumped to output; it can be assigned to a variable).

```
$output = ‘ls -al‘;
echo "<pre>$output</pre>";
```

See also **system()**, **passthru()**, **exec()**, **popen()**, and **escapeshellcmd()**.

Incrementing/Decrementing Operators

PHP supports C-style pre- and post-increment and decrement operators.

Table 10-4. Increment/decrement Operators

example	name	effect
++$a	Pre-increment	Increments $a by one, then returns $a.
$a++	Post-increment	Returns $a, then increments $a by one.
–$a	Pre-decrement	Decrements $a by one, then returns $a.
$a–	Post-decrement	Returns $a, then decrements $a by one.

Here’s a simple example script:

```
<?php
echo "<h3>Postincrement</h3>";
$a = 5;
echo "Should be 5: " . $a++ . "<br>\n";
echo "Should be 6: " . $a . "<br>\n";

echo "<h3>Preincrement</h3>";
$a = 5;
echo "Should be 6: " . ++$a . "<br>\n";
echo "Should be 6: " . $a . "<br>\n";

echo "<h3>Postdecrement</h3>";
$a = 5;
echo "Should be 5: " . $a- . "<br>\n";
echo "Should be 4: " . $a . "<br>\n";

echo "<h3>Predecrement</h3>";
$a = 5;
echo "Should be 4: " . -$a . "<br>\n";
echo "Should be 4: " . $a . "<br>\n";
?>
```

Logical Operators

Table 10-5. Logical Operators

example	name	result
$a and $b	And	True if both $a and $b are true.
$a or $b	Or	True if either $a or $b is true.
$a 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 if both $a and $b are true.
$a || $b	Or	True if either $a or $b is true.

The reason for the two different variations of "and" and "or" operators is that they operate at different precedences. (See Operator Precedence.)

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 10-6. Operator Precedence

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

String Operators

There are two string operators. The first is the concatenation operator (’.’), which returns the concatenation of its right and left arguments. The second is the concatenating assignment operator (’.=’). Please read Assignment Operators for more information.

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

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

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 elseif).

elseif

`elseif`, as its name suggests, is a combination of `if` and `else`. Like `else`, it extends an `if` statement to execute a different statement in case the original `if` expression evaluates to `FALSE`. However, unlike `else`, it will execute that alternative expression only if the `elseif` conditional expression evaluates to `TRUE`. For example, the following code would display `a is bigger than b`, `a equal to b` or `a is smaller than b`:

```
if ($a > $b) {
    print "a is bigger than b";
} elseif ($a == $b) {
    print "a is equal to b";
} else {
    print "a is smaller than b";
}
```

There may be several `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 control structures

PHP offers an alternative syntax for some of its control structures; namely, `if`, `while`, `for`, and `switch`. In each case, the basic form of the alternate syntax is to change the opening brace to a colon (:) and the closing brace to `endif;`, `endwhile;`, `endfor;`, or `endswitch;`, respectively.

```
<?php if ($a == 5): ?>
A is equal to 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;
```

See also while, for, and if for further examples.

while

`while` loops are the simplest type of loop in PHP. They behave just like their C counterparts. The basic form of a `while` statement is:

```
while (expr) statement
```

The meaning of a `while` statement is simple. It tells PHP to execute the nested statement(s) repeatedly, as long as the `while` expression evaluates to `TRUE`. The value of the expression is checked each time at the beginning of the loop, so even if this value changes during the execution of the nested statement(s), execution will not stop until the end of the iteration (each time PHP 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. PHP3 has no such construct; PHP4 does (see foreach). In PHP3, you can combine while with the **list()** and **each()** functions to achieve the same effect. See the documentation for these functions for an example.

foreach

PHP4 (not PHP3) includes a `foreach` construct, much like perl and some other languages. This simply gives an easy way to iterate over arrays. There are two syntaxes; the second is a minor but useful extension of the first:

```
foreach(array_expression as $value) statement
foreach(array_expression as $key => $value) statement
```

The first form loops over the array given by `array_expression`. On each loop, the value of the current element is assigned to `$value` and the internal array pointer is advanced by one (so on the next loop, you'll be looking at the next element).

The second form does the same thing, except that the current element's key will be assigned to the variable `$key` on each loop.

When `foreach` first starts executing, the internal array pointer is automatically reset to the first element of the array. This means that you do not need to call **reset()** before a `foreach` loop.

You may have noticed that the following are functionally identical:

```
reset ($arr);
while (list(, $value) = each ($arr)) {
    echo "Value: $value<br>\n";
}

foreach ($arr as $value) {
    echo "Value: $value<br>\n";
}
```

The following are also functionally identical:

```
reset ($arr);
while (list($key, $value) = each ($arr)) {
    echo "Key: $key; Value: $value<br>\n";
}

foreach ($arr as $key => $value) {
    echo "Key: $key; Value: $value<br>\n";
}
```

Some more examples to demonstrate usages:

```
/* foreach example 1: value only */

$a = array (1, 2, 3, 17);

foreach ($a as $v) {
   print "Current value of \$a: $v.\n";
}

/* foreach example 2: value (with key printed for illustration) */

$a = array (1, 2, 3, 17);
```

```
$i = 0; /* for illustrative purposes only */

foreach($a as $v) {
    print "\$a[$i] => $k.\n";
}

/* foreach example 3: key and value */

$a = array (
    "one" => 1,
    "two" => 2,
    "three" => 3,
    "seventeen" => 17
);

foreach($a as $k => $v) {
    print "\$a[$k] => $v.\n";
}
```

break

`break` ends execution of the current `for`, `while`, or `switch` structure.

`break` accepts an optional numeric argument which tells it how many nested enclosing structures are to be broken out of.

```
$arr = array( 'one', 'two', 'three', 'four', 'stop', 'five' );
while ( list( , $val ) = each( $arr ) ) {
   if ( $val == 'stop' ) {
      break;    /* You could also write 'break 1;' here. */
   }
   echo "$val<br>\n";
}

/* Using the optional argument. */
$i = 0;
while ( ++$i ) {
    switch ( $i ) {
    case 5:
        echo "At 5<br>\n";
        break 1;  /* Exit only the switch. */
    case 10:
        echo "At 10; quitting<br>\n";
        break 2;  /* Exit the switch and the while. */
    default:
        break;
    }
}
```

continue

`continue` is used within looping structures to skip the rest of the current loop iteration and continue execution at the beginning of the next iteration.

`continue` accepts an optional numeric argument which tells it how many levels of enclosing loops it should skip to the end of.

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

$i = 0;
while ($i++ < 5) {
    echo "Outer<br>\n";
    while (1) {
        echo "  Middle<br>\n";
        while (1) {
             echo "  Inner<br>\n";
             continue 3;
        }
        echo "This never gets output.<br>\n";
    }
    echo "Neither does this.<br>\n";
}
```

switch

The `switch` statement is similar to a series of IF statements on the same expression. In many occasions, you may want to compare the same variable (or expression) with many different values, and execute a different piece of code depending on which value it equals to. This is exactly what the `switch` statement is for.

The following two examples are two different ways to write the same thing, one using a series of `if` statements, and the other using the `switch` statement:

```
if ($i == 0) {
    print "i equals 0";
}
if ($i == 1) {
    print "i equals 1";
}
if ($i == 2) {
    print "i equals 2";
}

switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
```

```
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
}
```

It is important to understand how the `switch` statement is executed in order to avoid mistakes. The `switch` statement executes line by line (actually, statement by statement). In the beginning, no code is executed. Only when a `case` statement is found with a value that matches the value of the `switch` expression does PHP begin to execute the statements. PHP continues to execute the statements until the end of the `switch` block, or the first time it sees a `break` statement. If you don't write a `break` statement at the end of a case's statement list, PHP will go on executing the statements of the following case. For example:

```
switch ($i) {
    case 0:
        print "i equals 0";
    case 1:
        print "i equals 1";
    case 2:
        print "i equals 2";
}
```

Here, if $i 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).

In a `switch` statement, the condition is evaluated only once and the result is compared to each `case` statement. In an `elseif` statement, the condition is evaluated again. If your condition is more complicated than a simple compare and/or is in a tight loop, a `switch` may be faster.

The statement list for a case can also be empty, which simply passes control into the statement list for the next case.

```
switch ($i) {
    case 0:
    case 1:
    case 2:
        print "i is less than 3 but not negative";
        break;
    case 3:
        print "i is 3";
}
```

A special case is the default case. This case matches anything that wasn't matched by the other cases. For example:

```
switch ($i) {
    case 0:
        print "i equals 0";
        break;
    case 1:
```

```
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
    default:
        print "i is not equal to 0, 1 or 2";
}
```

The `case` expression may be any expression that evaluates to a simple type, that is, integer or floating-point numbers and strings. Arrays or objects cannot be used here unless they are dereferenced to a simple type.

The alternative syntax for control structures is supported with switches. For more information, see Alternative syntax for control structures .

```
switch ($i):
    case 0:
        print "i equals 0";
        break;
    case 1:
        print "i equals 1";
        break;
    case 2:
        print "i equals 2";
        break;
    default:
        print "i is not equal to 0, 1 or 2";
endswitch;
```

require()

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

An important note about how this works is that when a file is **include()**ed or **require()**ed, parsing drops out of PHP mode and into HTML mode at the beginning of the target file, and resumes PHP mode again at the end. For this reason, any code inside the target file which should be executed as PHP code must be enclosed within valid PHP start and end tags.

require() is not actually a function in PHP; rather, it is a language construct. It is subject to some different rules than functions are. For instance, **require()** is not subject to any containing control structures. For another, it does not return any value; attempting to read a return value from a **require()** call results in a parse error.

Unlike **include()**, **require()** will *always* read in the target file, *even if the line it's on never executes*. If you want to conditionally include a file, use **include()**. The conditional statement won't affect the **require()**. However, if the line on which the **require()** occurs is not executed, neither will any of the code in the target file be executed.

Similarly, looping structures do not affect the behaviour of **require()**. Although the code contained in the target file is still subject to the loop, the **require()** itself happens only once.

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');
```

Please note that both **include()** and **require()** actually pull the contents of the target file into the calling script file itself; they do not call the target via HTTP or anything like that. So any variable set in the scope in which the inclusion happens will be available within the included file automatically, since it has effectively become a part of the calling file.

```
require ("file.inc?varone=1&vartwo=2"); /* Won't work. */

$varone = 1;
$vartwo = 2;
require ("file.inc");  /* $varone and $vartwo will be available in file.inc */
```

Don't be misled by the fact that you can require or include files via HTTP using the Remote files feature; the above holds true regardless.

In PHP3, it is possible to execute a `return` statement inside a **require()**ed file, as long as that statement occurs in the global scope of the **require()**ed file. It may not occur within any block (meaning inside braces ({}). In PHP4, however, this ability has been discontinued. If you need this functionality, see **include()**.

include()

The **include()** statement includes and evaluates the specified file.

An important note about how this works is that when a file is **include()**ed or **require()**ed, parsing drops out of PHP mode and into HTML mode at the beginning of the target file, and resumes again at the end. For this reason, any code inside the target file which should be executed as PHP code must be enclosed within valid PHP start and end tags.

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 files.

```
$files = array ('first.inc', 'second.inc', 'third.inc');
for ($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);
}
```

In both PHP3 and PHP4, it is possible to execute a `return` statement inside an **include()**ed file, in order to terminate processing in that file and return to the script which called it. Some differences in the way this works exist, however. The first is that in PHP3, the `return` may not appear inside a block unless it's a function block, in which case the `return` applies to that function and not the whole file. In PHP4, however, this restriction does not exist. Also, PHP4 allows you to return values from **include()**ed files. You can take the value of the **include()** call as you would a normal function. This generates a parse error in PHP3.

Example 11-1. include() in PHP3 and PHP4

Assume the existence of the following file (named `test.inc`) in the same directory as the main file:

```
<?php
echo "Before the return <br>\n";
if (1) {
    return 27;
}
echo "After the return <br>\n";
?>
```

Assume that the main file (`main.html`) contains the following:

```
<?php
$retval = include ('test.inc');
echo "File returned: '$retval'<br>\n";
?>
```

When `main.html` is called in PHP3, it will generate a parse error on line 2; you can't take the value of an **include()** in PHP3. In PHP4, however, the result will be:

```
Before the return
File returned: '27'
```

Now, assume that `main.html` has been altered to contain the following:

```
<?php
include ('test.inc');
echo "Back in main.html<br>\n";
?>
```

In PHP4, the output will be:

```
Before the return
Back in main.html
```

However, PHP3 will give the following output:

```
Before the return
27Back in main.html
```

```
Parse error: parse error in /home/torben/public_html/phptest/main.html on line 5
```

The above parse error is a result of the fact that the `return` statement is enclosed in a non-function block within `test.inc`. When the return is moved outside of the block, the output is:

```
Before the return
27Back in main.html
```

The spurious '27' is due to the fact that PHP3 does not support `return`ning values from files like that.

Please note that both **include()** and **require()** actually pull the contents of the target file into the calling script file itself; they do not call the target via HTTP or anything like that. So any variable set in the scope in which the inclusion happens will be available within the included file automatically, since it has effectively become a part of the calling file.

```
include ("file.inc?varone=1&vartwo=2"); /* Won't work. */

$varone = 1;
$vartwo = 2;
include ("file.inc");  /* $varone and $vartwo will be available in file.inc */
```

Don't be misled by the fact that you can require or include files via HTTP using the Remote files feature; the above holds true regardless.

See also **readfile()**, **require()**, and **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.

In PHP3, functions must be defined before they are referenced. No such requirement exists in PHP4.

PHP does not support function overloading, nor is it possible to undefine or redefine previously-declared functions.

PHP3 does not support variable numbers of arguments to functions, although default arguments are supported (see Default argument values for more information). PHP4 supports both: see Variable-length argument lists and the function references for **func_num_args()**, **func_get_arg()**, and **func_get_args()** for more information.

Function arguments

Information may be passed to functions via the argument list, which is a comma-delimited list of variables and/or constants.

PHP supports passing arguments by value (the default), passing by reference, and default argument values. Variable-length argument lists are supported only in PHP4 and later; see Variable-length argument lists and the function references for **func_num_args()**, **func_get_arg()**, and **func_get_args()** for more information. A similar effect can be achieved in PHP3 by passing an array of arguments to a function:

```
function takes_array($input) {
    echo "$input[0] + $input[1] = ", $input[0]+$input[1];
}
```

Making arguments be passed by reference

By default, function arguments are passed by value (so that if you change the value of the argument within the function, it does not get changed outside of the function). If you wish to allow a function to modify its arguments, you must pass them by reference.

If you want an argument to a function to always be passed by reference, you can prepend an ampersand (&) to the argument name in the function definition:

```
function add_some_extra(&$string) {
    $string .= 'and something extra.';
}
```

```
$str = 'This is a string, ';
add_some_extra($str);
echo $str;    // outputs 'This is a string, and something extra.'
```

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.

Note that when using default arguments, any defaults should be on the right side of any non-default arguments; otherwise, things will not work as expected. Consider the following code snippet:

```
function makeyogurt ($type = "acidophilus", $flavour) {
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");   // won't work as expected
```

The output of the above example is:

```
Warning: Missing argument 2 in call to makeyogurt() in
/usr/local/etc/httpd/htdocs/php3test/functest.html on line 41
Making a bowl of raspberry .
```

Now, compare the above with this:

```
function makeyogurt ($flavour, $type = "acidophilus") {
    return "Making a bowl of $type $flavour.\n";
}

echo makeyogurt ("raspberry");   // works as expected
```

The output of this example is:

```
Making a bowl of acidophilus raspberry.
```

Variable-length argument lists

PHP4 has support for variable-length argument lists in user-defined functions. This is really quite easy, using the **func_num_args()**, **func_get_arg()**, and **func_get_args()** functions.

No special syntax is required, and argument lists may still be explicitly provided with function definitions and will behave as normal.

Returning values

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

```
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();
```

`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`.

Variable functions

PHP supports the concept of variable functions. This means that if a variable name has parentheses appended to it, PHP will look for a function with the same name as whatever the variable evaluates to, and will attempt to execute it. Among other things, this can be used to implement callbacks, function tables, and so forth.

Example 12-1. Variable function example

```
<?php
function foo() {
    echo "In foo()<br>\n";
}

function bar( $arg = ” ) {
    echo "In bar(); argument was ’$arg’.<br>\n";
}

$func = ’foo’;
$func();
$func = ’bar’;
$func( ’test’ );
?>
```

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. Multiple inheritance is not supported.

```
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 extends Cart {
    function Constructor_Cart ($item = "10", $num = 1) {
        $this->add_item ($item, $num);
    }
}

// Shop the same old boring stuff.

$default_cart   = new Constructor_Cart;

// Shop for real...

$different_cart = new Constructor_Cart ("20", 17);
```

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. Apache-specific Functions

apache_lookup_uri (PHP3 >= 3.0.4, PHP4)

Perform a partial request for the specified URI and return all info about it

```
class apache_lookup_uri (string filename)
```

This performs a partial request for a URI. It goes just far enough to obtain all the important information about the given resource and returns this information in a class. The properties of the returned class are:

status
the_request
status_line
method
content_type
handler
uri
filename
path_info
args
boundary
no_cache
no_local_copy
allowed
send_bodyct
bytes_sent
byterange
clength
unparsed_uri
mtime
request_time

Note: Apache_lookup_uri() only works when PHP is installed as an Apache module.

apache_note (PHP3 >= 3.0.2, PHP4)

Get and set apache request notes

```
string apache_note (string note_name [, string note_value])
```

Apache_note() is an Apache-specific function which gets and sets values in a request's `notes` table. If called with one argument, it returns the current value of note `note_name`. If called with two arguments, it sets the value of note `note_name` to `note_value` and returns the previous value of note `note_name`.

getallheaders (PHP3 , PHP4)

Fetch all HTTP request headers

```
array getallheaders(void);
```

This function returns an associative array of all the HTTP headers in the current request.

Note: You can also get at the value of the common CGI variables by reading them from the environment, which works whether or not you are using PHP as an Apache module. Use **phpinfo()** to see a list of all of the environment variables defined this way.

Example 1. getallheaders() Example

```
$headers = getallheaders();
while (list ($header, $value) = each ($headers)) {
    echo "$header: $value<br>\n";
}
```

This example will display all the request headers for the current request.

Note: Getallheaders() is currently only supported when PHP runs as an Apache module.

virtual (PHP3 , PHP4)

Perform an Apache sub-request

```
int virtual (string filename)
```

Virtual() is an Apache-specific function which is equivalent to <!–#include virtual...–> in mod_include. It performs an Apache sub-request. It is useful for including CGI scripts or .shtml files, or anything else that you would parse through Apache. Note that for a CGI script, the script must generate valid CGI headers. At the minimum that means it must generate a Content-type header. For PHP files, you need to use **include()** or **require()**; **virtual()** cannot be used to include a document which is itself a PHP file.

II. Arbitrary precision mathematics functions

These functions are only available if PHP was configured with `-enable-bcmath`.

bcadd (PHP3 , PHP4)

Add two arbitrary precision numbers.

```
string bcadd (string left operand, string right operand [, int scale])
```

Adds the *left operand* to the *right operand* and returns the sum in a string. The optional *scale* parameter is used to set the number of digits after the decimal place in the result.

See also **bcsub()**.

bccomp (PHP3 , PHP4)

Compare two arbitrary precision numbers.

```
int bccomp (string left operand, string right operand [, int scale])
```

Compares the *left operand* to the *right operand* and returns the result as an integer. The optional *scale* parameter is used to set the number of digits after the decimal place which will be used in the comparion. The return value is 0 if the two operands are equal. If the *left operand* is larger than the *right operand* the return value is +1 and if the *left operand* is less than the *right operand* the return value is -1.

bcdiv (PHP3 , PHP4)

Divide two arbitrary precision numbers.

```
string bcdiv (string left operand, string right operand [, int scale])
```

Divides the *left operand* by the *right operand* and returns the result. The optional *scale* sets the number of digits after the decimal place in the result.

See also **bcmul()**.

bcmod (PHP3 , PHP4)

Get modulus of an arbitrary precision number.

```
string bcmod (string left operand, string modulus)
```

Get the modulus of the *left operand* using *modulus*.

See also **bcdiv()**.

bcmul (PHP3 , PHP4)

Multiply two arbitrary precision number.

```
string bcmul (string left operand, string right operand [, int scale])
```

Multiply the *left operand* by the *right operand* and returns the result. The optional *scale* sets the number of digits after the decimal place in the result.

See also **bcdiv()**.

bcpow (PHP3 , PHP4)

Raise an arbitrary precision number to another.

```
string bcpow (string x, string y [, int scale])
```

Raise *x* to the power *y*. The optional *scale* can be used to set the number of digits after the decimal place in the result.

See also **bcsqrt()**.

bcscale (PHP3 , PHP4)

Set default scale parameter for all bc math functions.

```
string bcscale (int scale)
```

This function sets the default scale parameter for all subsequent bc math functions that do not explicitly specify a scale parameter.

bcsqrt (PHP3 , PHP4)

Get the square root of an arbitray precision number.

```
string bcsqrt (string operand, int scale)
```

Return the square root of the *operand*. The optional *scale* parameter sets the number of digits after the decimal place in the result.

See also **bcpow()**.

bcsub (PHP3 , PHP4)

Subtract one arbitrary precision number from another.

```
string bcsub (string left operand, string right operand [, int scale])
```

Subtracts the *right operand* from the *left operand* and returns the result in a string. The optional *scale* parameter is used to set the number of digits after the decimal place in the result.

See also **bcadd()**.

III. Array functions

array (unknown)

Create an array

```
array array(...);
```

Returns an array of the parameters. The parameters can be given an index with the => operator.

Note: array() is a language construct used to represent literal arrays, and not a regular function.

The following example demonstrates how to create a two-dimensional array, how to specify keys for associative arrays, and how to skip-and-continue numeric indices in normal arrays.

Example 1. array() example

```
$fruits = array (
    "fruits"  => array("a"=>"orange", "b"=>"banana", "c"=>"apple"),
    "numbers" => array(1, 2, 3, 4, 5, 6),
    "holes"   => array("first", 5 => "second", "third")
);
```

See also: **list()**.

array_count_values (PHP4 >= 4.0b4)

Counts all the values of an array

```
array array_count_values (array input)
```

array_count_values() returns an array using the values of the *input* array as keys and their frequency in *input* as values.

Example 1. array_count_values() example

```
$array = array(1, "hello", 1, "world", "hello");
array_count_values($array); // returns array(1=>2, "hello"=>2, "world"=>1)
```

Note: This function was added in PHP 4.0.

array_flip (PHP4 >= 4.0b4)

Flip all the values of an array

```
array array_flip (array trans)
```

array_flip() returns an array in flip order.

Example 1. array_flip() example

```
$trans = array_flip ($trans);
$original = strtr ($str, $trans);
```

Note: This function was added in PHP 4.0.

array_keys (PHP4)

Return all the keys of an array

```
array array_keys (array input [, mixed search_value])
```

array_keys() returns the keys, numeric and string, from the *input* array.

If the optional *search_value* is specified, then only the keys for that value are returned. Otherwise, all the keys from the *input* are returned.

Example 1. array_keys() example

```
$array = array(0 => 100, "color" => "red");
array_keys ($array);       // returns array (0, "color")

$array = array(1, 100, 2, 100);
array_keys ($array, 100);  //  returns array (0, 2)
```

See also **array_values()**.

Note: This function was added in PHP 4.0.

array_merge (PHP4)

Merge two or more arrays

```
array array_merge (array array1, array array2 [, ...])
```

array_merge() merges the elements of two or more arrays together so that the values of one are appended to the end of the previous one. It returns the resulting array.

If the input arrays had the same string keys, then the later value for that key will overwrite previous one. If, however, the arrays have the same numeric key, this does not happen since the values are appended.

Example 1. array_merge() example

```
$array1 = array ("color" => "red", 2, 4);
$array2 = array ("a", "b", "color" => "green", "shape" => "trapezoid");
array_merge ($array1, $array2);
```

Resulting array will be array("color" => "green", 2, 4, "a", "b", "shape" => "trapezoid").

Note: This function was added in PHP 4.0.

array_pad (PHP4 >= 4.0b4)

Pad array to the specified length with a value

```
array array_pad (array input, int pad_size, mixed pad_value)
```

array_pad() returns a copy of the *input* padded to size specified by *pad_size* with value *pad_value*. If *pad_size* is positive then the array is padded on the right, if it's negative then on the left. If the absolute value of *pad_size* is less than or equal to the length of the *input* then no padding takes place.

Example 1. array_pad() example

```
$input = array (12, 10, 9);

$result = array_pad ($input, 5, 0);
// result is array (12, 10, 9, 0, 0)

$result = array_pad ($input, -7, -1);
// result is array (-1, -1, -1, -1, 12, 10, 9)

$result = array_pad ($input, 2, "noop");
// not padded
```

array_pop (PHP4)

Pop the element off the end of array

```
mixed array_pop (array array)
```

array_pop() pops and returns the last value of the *array*, shortening the *array* by one element.

Example 1. array_pop() example

```
$stack = array ("orange", "apple", "raspberry");
$fruit = array_pop ($stack);
```

After this, $stack has only 2 elements: "orange" and "apple", and $fruit has "raspberry".

See also **array_push()**, **array_shift()**, and **array_unshift()**.

Note: This function was added in PHP 4.0.

array_push (PHP4)

Push one or more elements onto the end of array

```
int array_push (array array, mixed var [, ...])
```

array_push() treats *array* as a stack, and pushes the passed variables onto the end of *array*. The length of *array* increases by the number of variables pushed. Has the same effect as:

```
$array[] = $var;
```

repeated for each *var*.

Returns the new number of elements in the array.

Example 1. array_push() example

```
$stack = array (1, 2);
array_push($stack, "+", 3);
```

This example would result in $stack having 4 elements: 1, 2, "+", and 3.

See also: **array_pop()**, **array_shift()**, and **array_unshift()**.

Note: This function was added in PHP 4.0.

array_reverse (PHP4 >= 4.0b4)

Return an array with elements in reverse order

```
array array_reverse (array array)
```

array_reverse() takes input *array* and returns a new array with the order of the elements reversed.

Example 1. array_reverse() example

```
$input = array ("php", 4.0, array ("green", "red"));
$result = array_reverse ($input);
```

This makes $result have array (array ("green", "red"), 4.0, "php").

Note: This function was added in PHP 4.0 Beta 3.

array_shift (PHP4)

Pop an element off the beginning of array

```
mixed array_shift (array array)
```

array_shift() shifts the first value of the *array* off and returns it, shortening the *array* by one element and moving everything down.

Example 1. array_shift() example

```
$args = array ("-v", "-f");
$opt = array_shift ($args);
```

This would result in $args having one element "-f" left, and $opt being "-v".

See also **array_unshift()**, **array_push()**, and **array_pop()**.

Note: This function was added in PHP 4.0.

array_slice (PHP4)

Extract a slice of the array

```
array array_slice (array array, int offset [, int length])
```

array_slice() returns a sequence of elements from the *array* specified by the *offset* and *length* parameters.

If *offset* is positive, the sequence will start at that offset in the *array*. If *offset* is negative, the sequence will start that far from the end of the *array*.

If *length* is given and is positive, then the sequence will have that many elements in it. If *length* is given and is negative then the sequence will stop that many elements from the end of the array. If it is omitted, then the sequence will have everything from *offset* up until the end of the *array*.

Example 1. array_slice() examples

```
$input = array ("a", "b", "c", "d", "e");

$output = array_slice ($input, 2);      // returns "c", "d", and "e"
$output = array_slice ($input, 2, -1);  // returns "c", "d"
$output = array_slice ($input, -2, 1);  // returns "d"
$output = array_slice ($input, 0, 3);   // returns "a", "b", and "c"
```

See also **array_splice()**.

Note: This function was added in PHP 4.0.

array_splice (PHP4)

Remove a portion of the array and replace it with something else

```
array array_splice (array input, int offset [, int length [, array
replacement]])
```

array_splice() removes the elements designated by *offset* and *length* from the *input* array, and replaces them with the elements of the *replacement* array, if supplied.

If *offset* is positive then the start of removed portion is at that offset from the beginning of the *input* array. If *offset* is negative then it starts that far from the end of the *input* array.

If *length* is omitted, removes everything from *offset* to the end of the array. If *length* is specified and is positive, then that many elements will be removed. If *length* is specified and is negative then the end of the removed portion will be that many elements from the end of the array. Tip: to remove everything from *offset* to the end of the array when *replacement* is also specified, use `count($input)` for *length*.

If *replacement* array is specified, then the removed elements are replaced with elements from this array. If *offset* and *length* are such that nothing is removed, then the elements from the *replacement* array are inserted in the place specified by the *offset*. Tip: if the replacement is just one element it is not necessary to put `array()` around it, unless the element is an array itself.

The following equivalences hold:

```
array_push($input, $x, $y)     array_splice($input, count($input), 0, ar-
ray($x, $y))
array_pop($input)              array_splice($input, -1)
array_shift($input)            array_splice($input, 0, 1)
array_unshift($input, $x, $y)  array_splice($input, 0, 0, array($x, $y))
$a[$x] = $y                    array_splice($input, $x, 1, $y)
```

Returns the array consisting of removed elements.

Example 1. array_splice() examples

```
$input = array("red", "green", "blue", "yellow");
```

```
array_splice($input, 2);      // $input is now array("red", "green")
array_splice($input, 1, -1);  // $input is now array("red", "yellow")
array_splice($input, 1, count($input), "orange");
                              // $input is now array("red", "orange")
array_splice($input, -1, 1, array("black", "maroon"));
                              // $input is now array("red", "green",
                              //          "blue", "black", "maroon")
```

See also **array_slice()**.

Note: This function was added in PHP 4.0.

array_unshift (PHP4)

Push one or more elements onto the beginning of array

```
int array_unshift (array array, mixed var [, ...])
```

array_unshift() prepends passed elements to the front of the *array*. Note that the list of elements is prepended as a whole, so that the prepended elements stay in the same order.

Returns the new number of elements in the *array*.

Example 1. array_unshift() example

```
$queue = array("p1", "p3");
array_unshift($queue, "p4", "p5", "p6");
```

This would result in $queue having 5 elements: "p4", "p5", "p6", "p1", and "p3".

See also **array_shift()**, **array_push()**, and **array_pop()**.

Note: This function was added in PHP 4.0.

array_values (PHP4)

Return all the values of an array

```
array array_values (array input)
```

array_values() returns all the values from the *input* array.

Example 1. array_values() example

```
$array = array("size" => "XL", "color" => "gold");
array_values($array);    // returns array("XL", "gold")
```

Note: This function was added in PHP 4.0.

array_walk (PHP3 >= 3.0.3, PHP4)

Apply a user function to every member of an array.

```
int array_walk (array arr, string func, mixed userdata)
```

Applies the function named by *func* to each element of *arr*. *func* will be passed array value as the first parameter and array key as the second parameter. If *userdata* is supplied, it will be passed as the third parameter to the user function.

If *func* requires more than two or three arguments, depending on *userdata*, a warning will be generated each time **array_walk()** calls *func*. These warnings may be suppressed by prepending the '@' sign to the **array_walk()** call, or by using **error_reporting()**.

Note: If *func* needs to be working with the actual values of the array, specify that the first parameter of *func* should be passed by reference. Then any changes made to those elements will be made in the array itself.

Note: Passing the key and userdata to *func* was added in 4.0.

In PHP 4 **reset()** needs to be called as necessary since **array_walk()** does not reset the array by default.

Example 1. array_walk() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");

function test_alter (&$item1, $key, $prefix) {
   $item1 = "$prefix: $item1";
}

function test_print ($item2, $key) {
   echo "$key. $item2<br>\n";
}

array_walk ($fruits, 'test_print');
reset ($fruits);
array_walk ($fruits, 'test_alter', 'fruit');
reset ($fruits);
array_walk ($fruits, 'test_print');
```

See also **each()** and **list()**.

arsort (PHP3 , PHP4)

Sort an array in reverse order and maintain index association

```
void arsort (array array)
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Example 1. arsort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
arsort ($fruits);
for (reset ($fruits); $key = key ($fruits); next ($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[a] = orange fruits[d] = lemon fruits[b] = banana fruits[c] = apple` The fruits have been sorted in reverse alphabetical order, and the index associated with each element has been maintained.

See also: **asort()**, **rsort()**, **ksort()**, and **sort()**.

asort (PHP3 , PHP4)

Sort an array and maintain index association

```
void asort (array array)
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant.

Example 1. asort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
asort ($fruits);
for (reset ($fruits); $key = key ($fruits); next ($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[c] = apple fruits[b] = banana fruits[d] = lemon fruits[a] = orange` The fruits have been sorted in alphabetical order, and the index associated with each element has been maintained.

See also **arsort()**, **rsort()**, **ksort()**, and **sort()**.

compact (PHP4)

Create array containing variables and their values

```
array compact (string varname | array varnames [, ...])
```

compact() takes a variable number of parameters. Each parameter can be either a string containing the name of the variable, or an array of variable names. The array can contain other arrays of variable names inside it; **compact()** handles it recursively.

For each of these, **compact()** looks for a variable with that name in the current symbol table and adds it to the output array such that the variable name becomes the key and the contents of the variable become the value for that key. In short, it does the opposite of **extract()**. It returns the output array with all the variables added to it.

Example 1. compact() example

```
$city = "San Francisco";
$state = "CA";
$event = "SIGGRAPH";

$location_vars = array ("city", "state");

$result = compact ("event", $location_vars);
```

After this, $result will be array ("event" => "SIGGRAPH", "city" => "San Francisco", "state" => "CA").

See also **extract()**.

Note: This function was added in PHP 4.0.

count (PHP3 , PHP4)

count elements in a variable

```
int count (mixed var)
```

Returns the number of elements in *var*, which is typically an array (since anything else will have one element).

Returns 1 if the variable is not an array.

Returns 0 if the variable is not set.

> **Warning**
>
> **count()** may return 0 for a variable that isn't set, but it may also return 0 for a variable that has been initialized with an empty array. Use **isset()** to test if a variable is set.

See also: **sizeof()**, **isset()**, and **is_array()**.

current (PHP3 , PHP4)

Return the current element in an array

```
mixed current (array array)
```

Every array has an internal pointer to its "current" element, which is initialized to the first element inserted into the array.

The **current()** function simply returns the array element that's currently being pointed by the internal pointer. It does not move the pointer in any way. If the internal pointer points beyond the end of the elements list, **current()** returns false.

> **Warning**
>
> If the array contains empty elements (0 or "", the empty string) then this function will return false for these elements as well. This makes it impossible to determine if you are really at the end of the list in such an array using **current()**. To properly traverse an array that may contain empty elements, use the **each()** function.

See also: **end()**, **next()**, **prev()** and **reset()**.

each (PHP3 , PHP4)

Return the next key and value pair from an array

```
array each (array array)
```

Returns the current key and value pair from the array *array* and advances the array cursor. This pair is returned in a four-element array, with the keys *0*, *1*, *key*, and *value*. Elements *0* and *key* contain the key name of the array element, and *1* and *value* contain the data.

If the internal pointer for the array points past the end of the array contents, **each()** returns false.

Example 1. each() examples

```
$foo = array ("bob", "fred", "jussi", "jouni", "egon", "marliese");
$bar = each ($foo);
```

$bar now contains the following key/value pairs:

- 0 => 0
- 1 => 'bob'
- key => 0
- value => 'bob'

```
$foo = array ("Robert" => "Bob", "Seppo" => "Sepi");
$bar = each ($foo);
```

`$bar` now contains the following key/value pairs:

- 0 => 'Robert'
- 1 => 'Bob'
- key => 'Robert'
- value => 'Bob'

each() is typically used in conjunction with **list()** to traverse an array; for instance, $HTTP_POST_VARS:

Example 2. Traversing $HTTP_POST_VARS with each()

```
echo "Values submitted via POST method:<br>";
reset ($HTTP_POST_VARS);
while (list ($key, $val) = each ($HTTP_POST_VARS)) {
    echo "$key => $val<br>";
}
```

After **each()** has executed, the array cursor will be left on the next element of the array, or on the last element if it hits the end of the array.

See also **key()**, **list()**, **current()**, **reset()**, **next()**, and **prev()**.

end (PHP3 , PHP4)

Set the internal pointer of an array to its last element

```
end (array array)
```

end() advances *array*'s internal pointer to the last element.

See also: **current()**, **each()**, **end()**, **next()**, and **reset()**.

extract (PHP3 >= 3.0.7, PHP4)

Import variables into the symbol table from an array

```
void extract (array var_array [, int extract_type [, string prefix]])
```

This function is used to import variables from an array into the current symbol table. It takes associative array *var_array* and treats keys as variable names and values as variable values. For each key/value pair it will create a variable in the current symbol table, subject to *extract_type* and *prefix* parameters.

extract() checks for colissions with existing variables. The way collisions are treated is determined by *extract_type*. It can be one of the following values:

EXTR_OVERWRITE

If there is a collision, overwrite the existing variable.

EXTR_SKIP

If there is a collision, don't overwrite the existing variable.

EXTR_PREFIX_SAME

If there is a collision, prefix the new variable with *prefix*.

EXTR_PREFIX_ALL

Prefix all variables with *prefix*.

If *extract_type* is not specified, it is assumed to be EXTR_OVERWRITE.

Note that *prefix* is only required if *extract_type* is EXTR_PREFIX_SAME or EXTR_PREFIX_ALL.

extract() checks each key to see if it constitues a valid variable name, and if it does only then does it proceed to import it.

A possible use for extract is to import into symbol table variables contained in an associative array returned by **wddx_deserialize()**.

Example 1. Extract() example

```
<php?

/* Suppose that $var_array is an array returned from
   wddx_deserialize */

$size = "large";
$var_array = array ("color" => "blue",
                    "size"  => "medium",
                    "shape" => "sphere");
extract ($var_array, EXTR_PREFIX_SAME, "wddx");

print "$color, $size, $shape, $wddx_size\n";

?>
```

The above example will produce:

```
blue, large, sphere, medium
```

The $size wasn't overwritten, becaus we specified EXTR_PREFIX_SAME, which resulted in $wddx_size being created. If EXTR_SKIP was specified, then $wddx_size wouldn't even have been created. EXTR_OVERWRITE would have cause $size to have value "medium", and EXTR_PREFIX_ALL would result in new variables being named $wddx_color, $wddx_size, and $wddx_shape.

in_array (PHP4)

Return true if a value exists in an array

```
bool in_array (mixed needle, array haystack)
```

Searches *haystack* for *needle* and returns true if it is found in the array, false otherwise.

Example 1. in_array() example

```
$os = array ("Mac", "NT", "Irix", "Linux");
if (in_array ("Irix", $os))
    print "Got Irix";
```

Note: This function was added in PHP 4.0.

key (PHP3 , PHP4)

Fetch a key from an associative array

```
mixed key (array array)
```

key() returns the index element of the current array position.

See also: **current()**, **next()**

krsort (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sort an array by key in reverse order

```
int krsort (array array)
```

Sorts an array by key in reverse order, maintaining key to data correlations. This is useful mainly for associative arrays.

Example 1. krsort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
krsort ($fruits);
for (reset ($fruits); $key = key ($fruits); next ($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[d] = lemon fruits[c] = apple fruits[b] = banana fruits[a] = orange`

See also **asort()**, **arsort()**, **ksort() sort()**, and **rsort()**.

ksort (PHP3 , PHP4)

Sort an array by key

```
int ksort (array array)
```

Sorts an array by key, maintaining key to data correlations. This is useful mainly for associative arrays.

Example 1. ksort() example

```
$fruits = array ("d"=>"lemon", "a"=>"orange", "b"=>"banana", "c"=>"apple");
ksort ($fruits);
for (reset ($fruits); $key = key ($fruits); next ($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[a] = orange fruits[b] = banana fruits[c] = apple fruits[d] = lemon`

See also **asort()**, **arsort()**, **sort()**, and **rsort()**.

list (unknown)

Assign variables as if they were an array

```
void list(...);
```

Like **array()**, this is not really a function, but a language construct. **list()** is used to assign a list of variables in one operation.

Example 1. list() example

```
<table>
 <tr>
  <th>Employee name</th>
  <th>Salary</th>
 </tr>
```

```
<?php

$result = mysql($conn, "SELECT id, name, salary FROM employees");
while (list($id, $name, $salary) = mysql_fetch_row($result)) {
    print(" <tr>\n".
          "  <td><a href=\"info.php3?id=$id\">$name</a></td>\n".
          "  <td>$salary</td>\n".
          " </tr>\n");
}

?>

</table>
```

See also: **each()**, **array()**.

next (PHP3 , PHP4)

Advance the internal array pointer of an array

```
mixed next (array array)
```

Returns the array element in the next place that's pointed by the internal array pointer, or false if there are no more elements.

next() behaves like **current()**, with one difference. It advances the internal array pointer one place forward before returning the element. That means it returns the next array element and advances the internal array pointer by one. If advancing the internal array pointer results in going beyond the end of the element list, **next()** returns false.

> **Warning**
>
> If the array contains empty elements then this function will return false for these elements as well. To properly traverse an array which may contain empty elements see the **each()** function.

See also: **current()**, **end() prev()** and **reset()**

pos (PHP3 , PHP4)

Get the current element from an array

```
mixed pos (array array)
```

This is an alias for **current()**.

See also: **end()**, **next()**, **prev()** and **reset()**.

prev (PHP3 , PHP4)

Rewind the internal array pointer

```
mixed prev (array array)
```

Returns the array element in the previous place that's pointed by the internal array pointer, or false if there are no more elements.

> **Warning**
>
> If the array contains empty elements then this function will return false for these elements as well. To properly traverse an array which may contain empty elements see the **each()** function.

prev() behaves just like **next()**, except it rewinds the internal array pointer one place instead of advancing it.

See also: **current()**, **end() next()** and **reset()**

range (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Create an array containing a range of integers

```
array range (int low, int high)
```

range() returns an array of integers from *low* to *high*, inclusive.

See **shuffle()** for an example of its use.

reset (PHP3 , PHP4)

Set the internal pointer of an array to its first element

```
mixed reset (array array)
```

reset() rewinds *array*'s internal pointer to the first element.

reset() returns the value of the first array element.

See also: **current()**, **each()**, **next()**, **prev()**, and **reset()**.

rsort (PHP3 , PHP4)

Sort an array in reverse order

```
void rsort (array array)
```

This function sorts an array in reverse order (highest to lowest).

Example 1. rsort() example

```
$fruits = array ("lemon", "orange", "banana", "apple");
rsort ($fruits);
for (reset ($fruits); list ($key, $value) = each ($fruits); ) {
    echo "fruits[$key] = ", $value, "\n";
}
```

This example would display: `fruits[0] = orange fruits[1] = lemon fruits[2] = banana fruits[3] = apple` The fruits have been sorted in reverse alphabetical order.

See also: **arsort()**, **asort()**, **ksort()**, **sort()**, and **usort()**.

shuffle (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Shuffle an array

```
void shuffle (array array)
```

This function shuffles (randomizes the order of the elements in) an array.

Example 1. shuffle() example

```
$numbers = range (1,20);
srand (time());
shuffle ($numbers);
while (list(, $number) = each ($numbers)) {
    echo "$number ";
}
```

See also **arsort()**, **asort()**, **ksort()**, **rsort()**, **sort()** and **usort()**.

sizeof (PHP3 , PHP4)

Get the number of elements in an array

```
int sizeof (array array)
```

Returns the number of elements in the array.

See also: **count()**

sort (PHP3 , PHP4)

Sort an array

```
void sort (array array)
```

This function sorts an array. Elements will be arranged from lowest to highest when this function has completed.

Example 1. sort() example

```
$fruits = array ("lemon", "orange", "banana", "apple");
sort ($fruits);
for (reset ($fruits); $key = key ($fruits); next ($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\n";
}
```

This example would display: `fruits[0] = apple fruits[1] = banana fruits[2] = lemon fruits[3] = orange` The fruits have been sorted in alphabetical order.

See also: **arsort()**, **asort()**, **ksort()**, **rsort()**, and **usort()**.

uasort (PHP3 >= 3.0.4, PHP4)

Sort an array with a user-defined comparison function and maintain index association

```
void uasort (array array, function cmp_function)
```

This function sorts an array such that array indices maintain their correlation with the array elements they are associated with. This is used mainly when sorting associative arrays where the actual element order is significant. The comparison function is user-defined.

uksort (PHP3 >= 3.0.4, PHP4)

Sort an array by keys using a user-defined comparison function

```
void uksort (array array, function cmp_function)
```

This function will sort the keys of an array using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

Example 1. uksort() example

```
function mycompare ($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}
```

```
$a = array (4 => "four", 3 => "three", 20 => "twenty", 10 => "ten");
uksort ($a, mycompare);
while (list ($key, $value) = each ($a)) {
    echo "$key: $value\n";
}
```

This example would display: `20: twenty 10: ten 4: four 3: three`

See also: **arsort()**, **asort()**, **uasort()**, **ksort()**, **rsort()**, and **sort()**.

usort (PHP3 >= 3.0.3, PHP4)

Sort an array by values using a user-defined comparison function

```
void usort (array array, function cmp_function)
```

This function will sort an array by its values using a user-supplied comparison function. If the array you wish to sort needs to be sorted by some non-trivial criteria, you should use this function.

The comparison function must return an integer less than, equal to, or greater than zero if the first argument is considered to be respectively less than, equal to, or greater than the second. If two members compare as equal, their order in the sorted array is undefined.

Example 1. usort() example

```
function cmp ($a, $b) {
    if ($a == $b) return 0;
    return ($a > $b) ? -1 : 1;
}
$a = array (3, 2, 5, 6, 1);
usort ($a, cmp);
while (list ($key, $value) = each ($a)) {
    echo "$key: $value\n";
}
```

This example would display: `0: 6 1: 5 2: 3 3: 2 4: 1`

Note: Obviously in this trivial case the **rsort()** function would be more appropriate.

> **Warning**
>
> The underlying quicksort function in some C libraries (such as on Solaris systems) may cause PHP to crash if the comparison function does not return consistent values.

See also: **arsort()**, **asort()**, **ksort()**, **rsort()** and **sort()**.

IV. Aspell functions

The **aspell()** functions allows you to check the spelling on a word and offer suggestions.

You need the aspell library, available from: http:/metalab.unc.edu/kevina/aspell/.

aspell_new (PHP3 >= 3.0.7, PHP4)

Load a new dictionary

```
int aspell_new (string master, string personal)
```

Aspell_new() opens up a new dictionary and returns the dictionary link identifier for use in other aspell functions.

Example 1. Aspell_new()

```
$aspell_link=aspell_new ("english");
```

aspell_check (PHP3 >= 3.0.7, PHP4)

Check a word

```
boolean aspell_check (int dictionary_link, string word)
```

Aspell_check() checks the spelling of a word and returns true if the spelling is correct, false if not.

Example 1. Aspell_check()

```
$aspell_link=aspell_new ("english");
if (aspell_check ($aspell_link,"testt")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

aspell_check-raw (unknown)

Check a word without changing its case or trying to trim it

```
boolean aspell_check_raw (int dictionary_link, string word)
```

Aspell_check_raw() checks the spelling of a word, without changing its case or trying to trim it in any way and returns true if the spelling is correct, false if not.

Example 1. Aspell_check_raw()

```
$aspell_link=aspell_new ("english");
```

```
if (aspell_check_raw ($aspell_link, "test")) {
    echo "This is a valid spelling";
} else {
    echo "Sorry, wrong spelling";
}
```

aspell_suggest (PHP3 >= 3.0.7, PHP4)

Suggest spellings of a word

```
array aspell_suggest (int dictionary_link, string word)
```

Aspell_suggest() returns an array of possible spellings for the given word.

Example 1. Aspell_suggest()

```
$aspell_link=aspell_new ("english");

if (!aspell_check ($aspell_link, "test")) {
    $suggestions=aspell_suggest ($aspell_link, "test");

    for ($i=0; $i < count ($suggestions); $i++) {
        echo "Possible spelling: " . $suggestions[$i] . "<br>";
    }
}
```

V. Calendar functions

The calendar functions are only available if you have compiled the calendar extension in dl/calendar. Read dl/README for instructions on using it.

The calendar extension presents a series of functions to simplify converting between different calendar formats. The intermediary or standard it is based on is the Julian Day Count. The Julian Day Count is a count of days starting way earlier than any date most people would need to track (somewhere around 4000bc). To convert between calendar systems, you must first convert to Julian Day Count, then to the calendar system of your choice. Julian Day Count is very different from the Julian Calendar! For more information on calendar systems visit http://genealogy.org/~scottlee/cal-overview.html. Excerpts from this page are included in these instructions, and are in quotes.

JDToGregorian (unknown)

Converts Julian Day Count to Gregorian date

```
string jdtogregorian (int julianday)
```

Converts Julian Day Count to a string containing the Gregorian date in the format of "month/day/year".

GregorianToJD (unknown)

Converts a Gregorian date to Julian Day Count

```
int gregoriantojd (int month, int day, int year)
```

Valid Range for Gregorian Calendar 4714 B.C. to 9999 A.D.

Although this software can handle dates all the way back to 4714 B.C., such use may not be meaningful. The Gregorian calendar was not instituted until October 15, 1582 (or October 5, 1582 in the Julian calendar). Some countries did not accept it until much later. For example, Britain converted in 1752, The USSR in 1918 and Greece in 1923. Most European countries used the Julian calendar prior to the Gregorian.

Example 1. Calendar functions

```
<?php
$jd = GregorianToJD (10,11,1970);
echo "$jd\n";
$gregorian = JDToGregorian ($jd);
echo "$gregorian\n";
?>
```

JDToJulian (unknown)

Converts a Julian Calendar date to Julian Day Count

```
string jdtojulian (int julianday)
```

Converts Julian Day Count to a string containing the Julian Calendar Date in the format of "month/day/year".

JulianToJD (unknown)

Converts a Julian Calendar date to Julian Day Count

```
int juliantojd (int month, int day, int year)
```

Valid Range for Julian Calendar 4713 B.C. to 9999 A.D.

Although this software can handle dates all the way back to 4713 B.C., such use may not be meaningful. The calendar was created in 46 B.C., but the details did not stabilize until at least 8 A.D., and perhaps as late at the 4th century. Also, the beginning of a year varied from one culture to another - not all accepted January as the first month.

JDToJewish (unknown)

Converts a Julian Day Count to the Jewish Calendar

```
string jdtojewish (int julianday)
```

Converts a Julian Day Count the the Jewish Calendar.

JewishToJD (unknown)

Converts a date in the Jewish Calendar to Julian Day Count

```
int jewishtojd (int month, int day, int year)
```

Valid Range Although this software can handle dates all the way back to the year 1 (3761 B.C.), such use may not be meaningful.

The Jewish calendar has been in use for several thousand years, but in the early days there was no formula to determine the start of a month. A new month was started when the new moon was first observed.

JDToFrench (unknown)

Converts a Julian Day Count to the French Republican Calendar

```
string jdtofrench (int month, int day, int year)
```

Converts a Julian Day Count to the French Republican Calendar.

FrenchToJD (unknown)

Converts a date from the French Republican Calendar to a Julian Day Count

```
int frenchtojd (int month, int day, int year)
```

Converts a date from the French Republican Calendar to a Julian Day Count.

These routines only convert dates in years 1 through 14 (Gregorian dates 22 September 1792 through 22 September 1806). This more than covers the period when the calendar was in use.

JDMonthName (unknown)

Returns a month name

```
string jdmonthname (int julianday, int mode)
```

Returns a string containing a month name. *mode* tells this function which calendar to convert the Julian Day Count to, and what type of month names are to be returned.

Table 1. Calendar modes

Mode	Meaning
0	Gregorian - abbreviated
1	Gregorian
2	Julian - abbreviated
3	Julian
4	Jewish
5	French Republican

JDDayOfWeek (unknown)

Returns the day of the week

```
mixed jddayofweek (int julianday, int mode)
```

Returns the day of the week. Can return a string or an int depending on the mode.

Table 1. Calendar week modes

Mode	Meaning
0	Returns the day number as an int (0=sunday, 1=monday, etc)
1	Returns string containing the day of week (english-gregorian)
2	Returns a string containing the abbreviated day of week (english-gregorian)

easter_date (PHP3 >= 3.0.9, PHP4 >= 4.0RC2)

Get UNIX timestamp for midnight on Easter of a given year

```
int easter_date (int year)
```

Returns the UNIX timestamp corresponding to midnight on Easter of the given year. If no year is specified, the current year is assumed.

Warning: This function will generate a warning if the year is outside of the range for UNIX timestamps (i.e. before 1970 or after 2037).

Example 1. easter_date() example

```
echo date ("M-d-Y", easter_date(1999));        /* "Apr-04-1999" */
echo date ("M-d-Y", easter_date(2000));        /* "Apr-23-2000" */
echo date ("M-d-Y", easter_date(2001));        /* "Apr-15-2001" */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See **easter_days()** for calculating Easter before 1970 or after 2037.

easter_days (PHP3 >= 3.0.9, PHP4 >= 4.0RC2)

Get number of days after March 21 on which Easter falls for a given year

```
int easter_days (int year)
```

Returns the number of days after March 21 on which Easter falls for a given year. If no year is specified, the current year is assumed.

This function can be used instead of **easter_date()** to calculate Easter for years which fall outside the range of UNIX timestamps (i.e. before 1970 or after 2037).

Example 1. Easter_date() example

```
echo easter_days (1999);        /* 14, i.e. April 4   */
echo easter_days (1492);        /* 32, i.e. April 22  */
echo easter_days (1913);        /*  2, i.e. March 23  */
```

The date of Easter Day was defined by the Council of Nicaea in AD325 as the Sunday after the first full moon which falls on or after the Spring Equinox. The Equinox is assumed to always fall on 21st March, so the calculation reduces to determining the date of the full moon and the date of the following Sunday. The algorithm used here was introduced around the year 532 by Dionysius Exiguus. Under the Julian Calendar (for years before 1753) a simple 19-year cycle is used to track the phases of the Moon. Under the Gregorian Calendar (for years after 1753 - devised by Clavius and Lilius, and introduced by Pope Gregory XIII in October 1582, and into Britain and its then colonies in September 1752) two correction factors are added to make the cycle more accurate.

(The code is based on a C program by Simon Kershaw, <webmaster@ely.anglican.org>)

See also **easter_date()**.

unixtojd (PHP4 >= 4.0RC2)

Convert UNIX timestamp to Julian Day

```
int unixtojd ([int timestamp])
```

Return the Julian Day for a UNIX *timestamp* (seconds since 1.1.1970), or for the current day if no *timestamp* is given.

See also **jdtounix()**.

Note: This function is only available in PHP versions after PHP4RC1.

jdtounix (PHP4 >= 4.0RC2)

Convert Julian Day to UNIX timestamp

```
int jdtounix (int jday)
```

This function will return a UNIX timestamp corresponding to the Julian Day given in *jday* or false if *jday* is not inside the UNIX epoch (Gregorian years between 1970 and 2037 or 2440588 <= *jday* <= 2465342)

See also **jdtounix()**.

Note: This function is only available in PHP versions after PHP4RC1.

VI. COM support functions for Windows

These functions are only available on the Windows version of PHP. These functions have been added in PHP4.

com_load (PHP3 >= 3.0.3, PHP4)

???

```
string com_load (string module name [, string server name])
```

com_invoke (PHP3 >= 3.0.3, PHP4)

???

```
mixed com_invoke (resource object, string function_name [, mixed function
parameters, ...])
```

com_propget (PHP3 >= 3.0.3, PHP4)

???

```
mixed com_propget (resource object, string property)
```

com_get (PHP3 >= 3.0.3, PHP4)

???

```
mixed com_get (resource object, string property)
```

com_propput (PHP3 >= 3.0.3, PHP4)

???

```
void com_propput (resource object, string property, mixed value)
```

com_propset (PHP3 >= 3.0.3, PHP4)

???

```
void com_propset (resource object, string property, mixed value)
```

This function is an alias for **com_propput()**.

com_set (PHP3 >= 3.0.3, PHP4)

???

```
void com_set (resource object, string property, mixed value)
```

This function is an alias for **com_set()**.

VII. Class/Object Functions

get_class_methods (PHP4 CVS only)

Returns an array of class methods' names

```
array get_class_methods (string class_name)
```

This function returns an array of method names defined for the class specified by *class_name*.

get_class_vars (PHP4 CVS only)

Returns an array of default properties of the class

```
array get_class_vars (string class_name)
```

This function will return an array of default properties of the class.

get_object_vars (PHP4 CVS only)

Returns an array of object properties

```
array get_class_vars (object obj)
```

This function returns an array of object properties for the specified object *obj*.

method_exists (PHP4 >= 4.0b2)

Checks if the class method exists

```
bool method_exists (object object, string method_name)
```

This function returns true if the method given by *method_name* has been defined for the given *object*, false otherwise.

VIII. ClibPDF functions

ClibPDF lets you create PDF documents with PHP. It is available at FastIO (http://www.fastio.com) but it isn't free software. You should definitely read the licence before you start playing with ClibPDF. If you cannot fullfil the licence agreement consider using pdflib by Thomas Merz, which is also very powerful. ClibPDF functionality and API is similar to Thomas Merz's pdflib but, according to FastIO, ClibPDF is faster and creates smaller documents. This may have changed with the new version 2.0 of pdflib. A simple benchmark (the pdfclock.c example from pdflib 2.0 turned into a php script) actually shows no difference in speed at all. The file size is also similar if compression is turned off. So, try them both and see which one does the job for you.

This documentation should be read alongside the ClibPDF manual since it explains the library in much greater detail.

Many functions in the native ClibPDF and the PHP module, as well as in pdflib, have the same name. All functions except for **cpdf_open()** take the handle for the document as their first parameter. Currently this handle is not used internally since ClibPDF does not support the creation of several PDF documents at the same time. Actually, you should not even try it, the results are unpredictable. I can't oversee what the consequences in a multi threaded environment are. According to the author of ClibPDF this will change in one of the next releases (current version when this was written is 1.10). If you need this functionality use the pdflib module.

> **Note:** The function **cpdf_set_font()** has changed since PHP3 to support asian fonts. The encoding parameter is no longer an integer but a string.

One big advantage of ClibPDF over pdflib is the possibility to create the pdf document completely in memory without using temporary files. It also provides the ability to pass coordinates in a predefined unit length. This is a handy feature but can be simulated with **pdf_translate()**.

Most of the functions are fairly easy to use. The most difficult part is probably creating a very simple PDF document at all. The following example should help you get started. It creates a document with one page. The page contains the text "Times-Roman" in an outlined 30pt font. The text is underlined.

Example 1. Simple ClibPDF Example

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
cpdf_set_font($cpdf, "Times-Roman", 30, "WinAnsiEncoding");
cpdf_set_text_rendering($cpdf, 1);
cpdf_text($cpdf, "Times Roman outlined", 50, 750);
cpdf_moveto($cpdf, 50, 740);
cpdf_lineto($cpdf, 330, 740);
cpdf_stroke($cpdf);
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

The pdflib distribution contains a more complex example which creates a series of pages with an analog clock. Here is that example converted into PHP using the ClibPDF extension:

Example 2. pdfclock example from pdflib 2.0 distribution

```
<?php
$radius = 200;
$margin = 20;
$pagecount = 40;

$pdf = cpdf_open(0);
cpdf_set_creator($pdf, "pdf_clock.php3");
cpdf_set_title($pdf, "Analog Clock");

while($pagecount- > 0) {
  cpdf_page_init($pdf, $pagecount+1, 0, 2 * ($radius + $margin), 2 * ($ra-
dius + $margin), 1.0);

  cpdf_set_page_animation($pdf, 4, 0.5, 0, 0, 0);  /* wipe */

  cpdf_translate($pdf, $radius + $margin, $radius + $margin);
  cpdf_save($pdf);
  cpdf_setrgbcolor($pdf, 0.0, 0.0, 1.0);

  /* minute strokes */
  cpdf_setlinewidth($pdf, 2.0);
  for ($alpha = 0; $alpha < 360; $alpha += 6)
    {
    cpdf_rotate($pdf, 6.0);
    cpdf_moveto($pdf, $radius, 0.0);
    cpdf_lineto($pdf, $radius-$margin/3, 0.0);
    cpdf_stroke($pdf);
    }

  cpdf_restore($pdf);
  cpdf_save($pdf);

  /* 5 minute strokes */
  cpdf_setlinewidth($pdf, 3.0);
  for ($alpha = 0; $alpha < 360; $alpha += 30)
  {
    cpdf_rotate($pdf, 30.0);
    cpdf_moveto($pdf, $radius, 0.0);
    cpdf_lineto($pdf, $radius-$margin, 0.0);
    cpdf_stroke($pdf);
  }

  $ltime = getdate();

  /* draw hour hand */
  cpdf_save($pdf);
  cpdf_rotate($pdf, -(($ltime['minutes']/60.0) + $ltime['hours'] - 3.0) * 30.0);
  cpdf_moveto($pdf, -$radius/10, -$radius/20);
  cpdf_lineto($pdf, $radius/2, 0.0);
  cpdf_lineto($pdf, -$radius/10, $radius/20);
  cpdf_closepath($pdf);
  cpdf_fill($pdf);
  cpdf_restore($pdf);

  /* draw minute hand */
  cpdf_save($pdf);
```

```
  cpdf_rotate($pdf, -(($ltime['seconds']/60.0) + $ltime['minutes'] -
15.0) * 6.0);
  cpdf_moveto($pdf, -$radius/10, -$radius/20);
  cpdf_lineto($pdf, $radius * 0.8, 0.0);
  cpdf_lineto($pdf, -$radius/10, $radius/20);
  cpdf_closepath($pdf);
  cpdf_fill($pdf);
  cpdf_restore($pdf);

  /* draw second hand */
  cpdf_setrgbcolor($pdf, 1.0, 0.0, 0.0);
  cpdf_setlinewidth($pdf, 2);
  cpdf_save($pdf);
  cpdf_rotate($pdf, -(($ltime['seconds'] - 15.0) * 6.0));
  cpdf_moveto($pdf, -$radius/5, 0.0);
  cpdf_lineto($pdf, $radius, 0.0);
  cpdf_stroke($pdf);
  cpdf_restore($pdf);

  /* draw little circle at center */
  cpdf_circle($pdf, 0, 0, $radius/30);
  cpdf_fill($pdf);

  cpdf_restore($pdf);

  cpdf_finalize_page($pdf, $pagecount+1);
}

cpdf_finalize($pdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($pdf);
cpdf_close($pdf);
?>
```

cpdf_global_set_document_limits (PHP4 >= 4.0b4)

Sets document limits for any pdf document

```
void cpdf_global_set_document_limits (int maxpages, int maxfonts, int maximages,
int maxannotations, int maxobjects)
```

The **cpdf_global_set_document_limits()** function sets several document limits. This function has to be called before **cpdf_open()** to take effect. It sets the limits for any document open afterwards.

See also **cpdf_open()**.

cpdf_set_creator (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the creator field in the pdf document

```
void cpdf_set_creator (string creator)
```

The **cpdf_set_creator()** function sets the creator of a pdf document.

See also **cpdf_set_subject()**, **cpdf_set_title()**, **cpdf_set_keywords()**.

cpdf_set_title (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the title field of the pdf document

```
void cpdf_set_title (string title)
```

The **cpdf_set_title()** function sets the title of a pdf document.

See also **cpdf_set_subject()**, **cpdf_set_creator()**, **cpdf_set_keywords()**.

cpdf_set_subject (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the subject field of the pdf document

```
void cpdf_set_subject (string subject)
```

The **cpdf_set_subject()** function sets the subject of a pdf document.

See also **cpdf_set_title()**, **cpdf_set_creator()**, **cpdf_set_keywords()**.

cpdf_set_keywords (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the keywords field of the pdf document

```
void cpdf_set_keywords (string keywords)
```

The **cpdf_set_keywords()** function sets the keywords of a pdf document.

See also **cpdf_set_title()**, **cpdf_set_creator()**, **cpdf_set_subject()**.

cpdf_open (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Opens a new pdf document

```
int cpdf_open (int compression, string filename)
```

The **cpdf_open()** function opens a new pdf document. The first parameter turns document compression on if it is unequal to 0. The second optional parameter sets the file in which the document is written. If it is omitted the document is created in memory and can either be written into a file with the **cpdf_save_to_file()** or written to standard output with **cpdf_output_buffer()**.

Note: The return value will be needed in futher versions of ClibPDF as the first parameter in all other functions which are writing to the pdf document.

The ClibPDF library takes the filename "-" as a synonym for stdout. If PHP is compiled as an apache module this will not work because the way ClibPDF outputs to stdout does not work with apache. You can solve this problem by skipping the filename and using **cpdf_output_buffer()** to output the pdf document.

See also **cpdf_close()**, **cpdf_output_buffer()**.

cpdf_close (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Closes the pdf document

```
void cpdf_close (int pdf document)
```

The **cpdf_close()** function closes the pdf document. This should be the last function even after **cpdf_finalize()**, **cpdf_output_buffer()** and **cpdf_save_to_file()**.

See also **cpdf_open()**.

cpdf_page_init (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Starts new page

```
void cpdf_page_init (int pdf document, int page number, int orientation, double
height, double width, double unit)
```

The **cpdf_page_init()** function starts a new page with height *height* and width *width*. The page has number *page number* and orientation *orientation*. *orientation* can be 0 for portrait and 1 for landscape. The last optional parameter *unit* sets the unit for the koordinate system. The value should be the number of postscript points per unit. Since one inch is equal to 72 points, a value of 72 would set the unit to one inch. The default is also 72.

See also **cpdf_set_current_page()**.

cpdf_finalize_page (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Ends page

```
void cpdf_finalize_page (int pdf document, int page number)
```

The **cpdf_finalize_page()** function ends the page with page number *page number*. This function is only for saving memory. A finalized page takes less memory but cannot be modified anymore.

See also **cpdf_page_init()**.

cpdf_finalize (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Ends document

```
void cpdf_finalize (int pdf document)
```

The **cpdf_finalize()** function ends the document. You still have to call **cpdf_close()**.

See also **cpdf_close()**.

cpdf_output_buffer (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Outputs the pdf document in memory buffer

```
void cpdf_output_buffer (int pdf document)
```

The **cpdf_output_buffer()** function outputs the pdf document to stdout. The document has to be created in memory which is the case if **cpdf_open()** has been called with no filename parameter.

See also **cpdf_open()**.

cpdf_save_to_file (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Writes the pdf document into a file

```
void cpdf_save_to_file (int pdf document, string filename)
```

The **cpdf_save_to_file()** function outputs the pdf document into a file if it has been created in memory. This function is not needed if the pdf document has been open by specifying a filename as a parameter of **cpdf_open()**.

See also **cpdf_output_buffer()**, **cpdf_open()**.

cpdf_set_current_page (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Sets current page

```
void cpdf_set_current_page (int pdf document, int page number)
```

The **cpdf_set_current_page()** function set the page on which all operations are performed. One can switch between pages until a page is finished with **cpdf_finalize_page()**.

See also **cpdf_finalize_page()**.

cpdf_begin_text (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Starts text section

```
void cpdf_begin_text (int pdf document)
```

The **cpdf_begin_text()** function starts a text section. It must be ended with **cpdf_end_text()**.

Example 1. Text output

```
<?php cpdf_begin_text($pdf);
cpdf_set_font($pdf, 16, "Helvetica", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf) ?>
```

See also **cpdf_end_text()**.

cpdf_end_text (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Ends text section

```
void cpdf_end_text (int pdf document)
```

The **cpdf_end_text()** function ends a text section which was started with **cpdf_begin_text()**.

Example 1. Text output

```
<?php cpdf_begin_text($pdf);
cpdf_set_font($pdf, 16, "Helvetica", "WinAnsiEncoding");
cpdf_text($pdf, 100, 100, "Some text");
cpdf_end_text($pdf) ?>
```

See also **cpdf_begin_text()**.

cpdf_show (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Output text at current position

```
void cpdf_show (int pdf document, string text)
```

The **cpdf_show()** function outputs the string in *text* at the current position.

See also **cpdf_text()**, **cpdf_begin_text()**, **cpdf_end_text()**.

cpdf_show_xy (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Output text at position

```
void cpdf_show_xy (int pdf document, string text, double x-koor, double y-koor,
int mode)
```

The **cpdf_show_xy()** function outputs the string *text* at position with coordinates (*x-koor*, *y-koor*). The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

Note: The function **cpdf_show_xy()** is identical to **cpdf_text()** without the optional parameters.

See also **cpdf_text()**.

cpdf_text (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Output text with parameters

```
void cpdf_text (int pdf document, string text, double x-koor, double y-koor, int
mode, double orientation, int alignmode)
```

The **cpdf_text()** function outputs the string *text* at position with coordinates (*x-koor*, *y-koor*). The optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit. The optional parameter *orientation* is the rotation of the text in degree. The optional parameter *alignmode* determines how the text is align. See the ClibPDF documentation for possible values.

See also **cpdf_show_xy()**.

cpdf_set_font (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Select the current font face and size

```
void cpdf_set_font (int pdf document, string font name, double size, string
encoding)
```

The **cpdf_set_font()** function sets the the current font face, font size and encoding. Currently only the standard postscript fonts are supported. The last parameter *encoding* can take the following values: "MacRomanEncoding", "MacExpertEncoding", "WinAnsiEncoding", and "NULL". "NULL" stands for the font's built-in encoding. See the ClibPDF Manual for more information, especially how to support asian fonts.

cpdf_set_leading (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets distance between text lines

```
void cpdf_set leading (int pdf document, double distance)
```

The **cpdf_set_leading()** function sets the distance between text lines. This will be used if text is output by **cpdf_continue_text()**.

See also **cpdf_continue_text()**.

cpdf_set_text_rendering (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Determines how text is rendered

```
void cpdf_set_text_rendering (int pdf document, int mode)
```

The **cpdf_set_text_rendering()** function determines how text is rendered. The possible values for *mode* are 0=fill text, 1=stroke text, 2=fill and stroke text, 3=invisible, 4=fill text and add it to 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.

cpdf_set_horiz_scaling (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets horizontal scaling of text

```
void cpdf_set_horiz_scaling (int pdf document, double scale)
```

The **cpdf_set_horiz_scaling()** function sets the horizontal scaling to *scale* percent.

cpdf_set_text_rise (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the text rise

```
void cpdf_set_text_rise (int pdf document, double value)
```

The **cpdf_set_text_rise()** function sets the text rising to *value* units.

cpdf_set_text_matrix (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets the text matrix

```
void cpdf_set_text_matrix (int pdf document, array matrix)
```

The **cpdf_set_text_matrix()** function sets a matrix which describes a transformation applied on the current text font.

cpdf_set_text_pos (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets text position

```
void cpdf_set_text_pos (int pdf document, double x-koor, double y-koor, int
mode)
```

The **cpdf_set_text_pos()** function sets the position of text for the next **cpdf_show()** function call.

The last optional parameter *mode* determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_show()**, **cpdf_text()**.

cpdf_set_char_spacing (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets character spacing

```
void cpdf_set_char_spacing (int pdf document, double space)
```

The **cpdf_set_char_spacing()** function sets the spacing between characters.
See also **cpdf_set_word_spacing()**, **cpdf_set_leading()**.

cpdf_set_word_spacing (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets spacing between words

```
void cpdf_set_word_spacing (int pdf document, double space)
```

The **cpdf_set_word_spacing()** function sets the spacing between words.
See also **cpdf_set_char_spacing()**, **cpdf_set_leading()**.

cpdf_continue_text (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Output text in next line

```
void cpdf_continue_text (int pdf document, string text)
```

The **cpdf_continue_text()** function outputs the string in *text* in the next line.
See also **cpdf_show_xy()**, **cpdf_text()**, **cpdf_set_leading()**, **cpdf_set_text_pos()**.

cpdf_stringwidth (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Returns width of text in current font

```
double cpdf_stringwidth (int pdf document, string text)
```

The **cpdf_stringwidth()** function returns the width of the string in *text*. It requires a font to be set before.
See also **cpdf_set_font()**.

cpdf_save (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Saves current enviroment

```
void cpdf_save (int pdf document)
```

The **cpdf_save()** function saves the current enviroment. It works like the postscript command gsave. Very useful if you want to translate or rotate an object without effecting other objects.

See also **cpdf_restore()**.

cpdf_restore (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Restores formerly saved enviroment

```
void cpdf_restore (int pdf document)
```

The **cpdf_restore()** function restores the enviroment saved with **cpdf_save()**. It works like the postscript command grestore. Very useful if you want to translate or rotate an object without effecting other objects.

Example 1. Save/Restore

```
<?php cpdf_save($pdf);
// do all kinds of rotations, transformations, ...
cpdf_restore($pdf) ?>
```

See also **cpdf_save()**.

cpdf_translate (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets origin of coordinate system

```
void cpdf_translate (int pdf document, double x-koor, double y-koor, int mode)
```

The **cpdf_translate()** function set the origin of coordinate system to the point (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

cpdf_scale (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets scaling

```
void cpdf_scale (int pdf document, double x-scale, double y-scale)
```

The **cpdf_scale()** function set the scaling factor in both directions.

cpdf_rotate (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets rotation

```
void cpdf_rotate (int pdf document, double angle)
```

The **cpdf_rotate()** function set the rotation in degress to *angle*.

cpdf_setflat (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets flatness

```
void cpdf_setflat (int pdf document, double value)
```

The **cpdf_setflat()** function set the flatness to a value between 0 and 100.

cpdf_setlinejoin (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets linejoin parameter

```
void cpdf_setlinejoin (int pdf document, long value)
```

The **cpdf_setlinejoin()** function set the linejoin parameter between a value of 0 and 2. 0 = miter, 1 = round, 2 = bevel.

cpdf_setlinecap (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets linecap aparameter

```
void cpdf_setlinecap (int pdf document, int value)
```

The **cpdf_setlinecap()** function set the linecap parameter between a value of 0 and 2. 0 = butt end, 1 = round, 2 = projecting square.

cpdf_setmiterlimit (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets miter limit

```
void cpdf_setmiterlimit (int pdf document, double value)
```

The **cpdf_setmiterlimit()** function set the miter limit to a value greater or equal than 1.

cpdf_setlinewidth (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets line width

```
void cpdf_setlinewidth (int pdf document, double width)
```

The **cpdf_setlinewidth()** function set the line width to *width*.

cpdf_setdash (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets dash pattern

```
void cpdf_setdash (int pdf document, double white, double black)
```

The **cpdf_setdash()** function set the dash pattern *white* white units and *black* black units. If both are 0 a solid line is set.

cpdf_moveto (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets current point

```
void cpdf_moveto (int pdf document, double x-koor, double y-koor, int mode)
```

The **cpdf_moveto()** function set the current point to the coordinates *x-koor* and *y-koor*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

cpdf_rmoveto (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Sets current point

```
void cpdf_rmoveto (int pdf document, double x-koor, double y-koor, int mode)
```

The **cpdf_rmoveto()** function set the current point relative to the coordinates *x-koor* and *y-koor*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**.

cpdf_curveto (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draws a curve

```
void cpdf_curveto (int pdf document, double x1, double y1, double x2, double y2,
double x3, double y3, int mode)
```

The **cpdf_curveto()** function draws a Bezier curve from the current point to the point (*x3*, *y3*) using (*x1*, *y1*) and (*x2*, *y2*) as control points.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_rlineto()**, **cpdf_lineto()**.

cpdf_lineto (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draws a line

```
void cpdf_lineto (int pdf document, double x-koor, double y-koor, int mode)
```

The **cpdf_lineto()** function draws a line from the current point to the point with coordinates (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_rlineto (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Draws a line

```
void cpdf_rlineto (int pdf document, double x-koor, double y-koor, int mode)
```

The **cpdf_rlineto()** function draws a line from the current point to the relative point with coordinates (*x-koor*, *y-koor*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_moveto()**, **cpdf_rmoveto()**, **cpdf_curveto()**.

cpdf_circle (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draw a circle

```
void cpdf_circle (int pdf document, double x-koor, double y-koor, double radius,
int mode)
```

The **cpdf_circle()** function draws a circle with center at point (*x-koor*, *y-koor*) and radius *radius*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_arc()**.

cpdf_arc (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draws an arc

```
void cpdf_arc (int pdf document, double x-koor, double y-koor, double radius,
double start, double end, int mode)
```

The **cpdf_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*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_circle()**.

cpdf_rect (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draw a rectangle

```
void cpdf_rect (int pdf document, double x-koor, double y-koor, double width,
double height, int mode)
```

The **cpdf_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*.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

cpdf_closepath (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Close path

```
void cpdf_closepath (int pdf document)
```

The **cpdf_closepath()** function closes the current path.

cpdf_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Draw line along path

```
void cpdf_stroke (int pdf document)
```

The **cpdf_stroke()** function draws a line along current path.

See also **cpdf_closepath()**, **cpdf_closepath_stroke()**.

cpdf_closepath_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Close path and draw line along path

```
void cpdf_closepath_stroke (int pdf document)
```

The **cpdf_closepath_stroke()** function is a combination of **cpdf_closepath()** and **cpdf_stroke()**. Than clears the path.

See also **cpdf_closepath()**, **cpdf_stroke()**.

cpdf_fill (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Fill current path

```
void cpdf_fill (int pdf document)
```

The **cpdf_fill()** function fills the interior of the current path with the current fill color.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_fill_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Fill and stroke current path

```
void cpdf_fill_stroke (int pdf document)
```

The **cpdf_fill_stroke()** function fills the interior of the current path with the current fill color and draws current path.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_fill()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_closepath_fill_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Close, fill and stroke current path

```
void cpdf_closepath_fill_stroke (int pdf document)
```

The **cpdf_closepath_fill_stroke()** function closes, fills the interior of the current path with the current fill color and draws current path.

See also **cpdf_closepath()**, **cpdf_stroke()**, **cpdf_fill()**, **cpdf_setgray_fill()**, **cpdf_setgray()**, **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_clip (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Clips to current path

```
void cpdf_clip (int pdf document)
```

The **cpdf_clip()** function clips all drawing to the current path.

cpdf_setgray_fill (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets filling color to gray value

```
void cpdf_setgray_fill (int pdf document, double value)
```

The **cpdf_setgray_fill()** function sets the current gray value to fill a path.

See also **cpdf_setrgbcolor_fill()**.

cpdf_setgray_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets drawing color to gray value

```
void cpdf_setgray_stroke (int pdf document, double gray value)
```

The **cpdf_setgray_stroke()** function sets the current drawing color to the given gray value.

See also **cpdf_setrgbcolor_stroke()**.

cpdf_setgray (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets drawing and filling color to gray value

```
void cpdf_setgray (int pdf document, double gray value)
```

The **cpdf_setgray_stroke()** function sets the current drawing and filling color to the given gray value.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor_fill()**.

cpdf_setrgbcolor_fill (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets filling color to rgb color value

```
void cpdf_setrgbcolor_fill (int pdf document, double red value, double green value, double blue value)
```

The **cpdf_setrgbcolor_fill()** function sets the current rgb color value to fill a path.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor()**.

cpdf_setrgbcolor_stroke (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets drawing color to rgb color value

```
void cpdf_setrgbcolor_stroke (int pdf document, double red value, double green value, double blue value)
```

The **cpdf_setrgbcolor_stroke()** function sets the current drawing color to the given rgb color value.

See also **cpdf_setrgbcolor_fill()**, **cpdf_setrgbcolor()**.

cpdf_setrgbcolor (PHP3 >= 3.0.8, PHP4 >= 4.0b4)

Sets drawing and filling color to rgb color value

```
void cpdf_setrgbcolor (int pdf document, double red value, double green value, double blue value)
```

The **cpdf_setrgbcolor_stroke()** function sets the current drawing and filling color to the given rgb color value.

See also **cpdf_setrgbcolor_stroke()**, **cpdf_setrgbcolor_fill()**.

cpdf_add_outline (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Adds bookmark for current page

```
void cpdf_add_outline (int pdf document, string text)
```

The **cpdf_add_outline()** function adds a bookmark with text *text* that points to the current page.

Example 1. Adding a page outline

```
<?php
$cpdf = cpdf_open(0);
cpdf_page_init($cpdf, 1, 0, 595, 842);
cpdf_add_outline($cpdf, 0, 0, 0, 1, "Page 1");
// ...
// some drawing
// ...
cpdf_finalize($cpdf);
Header("Content-type: application/pdf");
cpdf_output_buffer($cpdf);
cpdf_close($cpdf);
?>
```

cpdf_set_page_animation (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Sets duration between pages

```
void cpdf_set_page_animation (int pdf document, int transition, double duration)
```

The **cpdf_set_page_animation()** function set the transition between following pages.

The value of *transition* can be

0 for none,
1 for two lines sweeping across the screen reveal the page,
2 for multiple lines sweeping across the screen reveal the page,
3 for a box reveals the page,
4 for a single line sweeping across the screen reveals the page,
5 for the old page dissolves to reveal the page,
6 for the dissolve effect moves from one screen edge to another,
7 for the old page is simply replaced by the new page (default)

The value of *duration* is the number of seconds between page flipping.

cpdf_import_jpeg (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Opens a JPEG image

```
int cpdf_open_jpeg (int pdf document, string file name, double x-koor, double
y-koor, double angle, double width, double height, double x-scale, double
y-scale, int mode)
```

The **cpdf_import_jpeg()** function opens an image stored in the file with the name *file name*. The format of the image has to be jpeg. The image is placed on the current page at position (*x-koor*, *y-koor*). The image is rotated by *angle* degres.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_place_inline_image()**,

cpdf_place_inline_image (PHP3 >= 3.0.9, PHP4 >= 4.0b4)

Places an image on the page

```
void cpdf_place_inline_image (int pdf document, int image, double x-koor, double
y-koor, double angle, double width, double height, int mode)
```

The **cpdf_place_inline_image()** function places an image created with the php image functions on the page at postion (*x-koor*, *y-koor*). The image can be scaled at the same time.

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

See also **cpdf_import_jpeg()**,

cpdf_add_annotation (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

Adds annotation

```
void cpdf_add_annotation (int pdf document, double llx, double lly, double urx,
double ury, string title, string content, int mode)
```

The **cpdf_add_annotation()** adds a note with the lower left corner at (*llx*, *lly*) and the upper right corner at (*urx*, *ury*).

The last optional parameter determines the unit length. If is 0 or omitted the default unit as specified for the page is used. Otherwise the koodinates are measured in postscript points disregarding the current unit.

IX. Cybercash payment functions

These functions are only available if the interpreter has been compiled with the `-with-cybercash=[DIR]`. These functions have been added in PHP4.

cybercash_encr (PHP4 >= 4.0b4)

???

```
array cybercash_encr (string wmk, string sk, string inbuff)
```

The function returns an associative array with the elements "errcode" and, if "errcode" is false, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_decr (PHP4 >= 4.0b4)

???

```
array cybercash_decr (string wmk, string sk, string inbuff)
```

The function returns an associative array with the elements "errcode" and, if "errcode" is false, "outbuff" (string), "outLth" (long) and "macbuff" (string).

cybercash_base64_encode (PHP4 >= 4.0b4)

???

```
string cybercash_base64_encode  (string inbuff)
```

cybercash_base64_decode (PHP4 >= 4.0b4)

```
string cybercash_base64_decode  (string inbuff)
```

X. DOM XML functions

These functions are only available if PHP was configured with `-with-dom=[DIR]`, using the GNOME xml library. These functions have been added in PHP4.

This module defines the following constants:

Table 1. XML constants

Constant	Value	Description
XML_ELEMENT_NODE	1	
XML_ATTRIBUTE_NODE	2	
XML_TEXT_NODE	3	
XML_CDATA_SECTION_NODE	4	
XML_ENTITY_REF_NODE	5	
XML_ENTITY_NODE	6	
XML_PI_NODE	7	
XML_COMMENT_NODE	8	
XML_DOCUMENT_NODE	9	
XML_DOCUMENT_TYPE_NODE	10	
XML_DOCUMENT_FRAG_NODE	11	
XML_NOTATION_NODE	12	
XML_GLOBAL_NAMESPACE	1	
XML_LOCAL_NAMESPACE	2	

This module defines a number of classes. The DOM XML functions return a parsed tree of the XML document with each node being an object belonging to one of these classes.

xmldoc (PHP4 >= 4.0b4)

Creates a DOM object of an XML document

```
object xmldoc (string str)
```

The function parses the XML document in *str* and returns an object of class "Dom document", having the properties "doc" (resource), "version" (string) and "type" (long).

xmldocfile (PHP4 >= 4.0b4)

Creates a DOM object from XML file

```
object xmldocfile (string filename)
```

The function parses the XML document in the file named *filename* and returns an object of class "Dom document", having the properties "doc" (resource), "version" (string).

xmltree (PHP4 >= 4.0b4)

Creates a tree of php objects from XML document

```
object xmltree (string str)
```

The function parses the XML document in *str* and returns a tree PHP objects as the parsed document.

XI. 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 (PHP3 , PHP4)

close an open gz-file pointer

```
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 (PHP3 , PHP4)

test for end-of-file on a gz-file pointer

```
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 (PHP3 , PHP4)

read entire gz-file into an array

```
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 (PHP3 , PHP4)

get character from gz-file pointer

```
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 (PHP3 , PHP4)

get line from file pointer

```
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 (PHP3 , PHP4)

get line from gz-file pointer and strip HTML tags

```
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 (PHP3 , PHP4)

open gz-file

```
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 (PHP3 , PHP4)

output all remaining data on a gz-file pointer

```
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 (PHP3 , PHP4)

write to a gz-file pointer

```
int gzputs (int zp, string str [, int length])
```

gzputs() is an alias to **gzwrite()**, and is identical in every way.

gzread (PHP3 , PHP4)

Binary-safe gz-file read

```
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 (PHP3 , PHP4)

rewind the position of a gz-file pointer

```
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 (PHP3 , PHP4)

seek on a gz-file pointer

```
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 (PHP3 , PHP4)

tell gz-file pointer read/write position

```
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 (PHP3 , PHP4)

Binary-safe gz-file write

```
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 (PHP3 , PHP4)

output a gz-file

```
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()**.

XII. Database (dbm-style) abstraction layer functions

These functions build the foundation for accessing Berkeley DB style databases.

This is a general abstraction layer for several file-based databases. As such, functionality is limited to a subset of features modern databases such as Sleepycat Software's DB2 () support. (This is not to be confused with IBM's DB2 software, which is supported through the ODBC functions.)

The behaviour of various aspects depend on the implementation of the underlying database. Functions such as **dba_optimize()** and **dba_sync()** will do what they promise for one database and will do nothing for others.

The following handlers are supported:

- dbm is the oldest (original) type of Berkeley DB style databases. You should avoid it, if possible. We do not support the compatibility functions built into DB2 and gdbm, because they are only compatible on the source code level, but cannot handle the original dbm format.
- ndbm is a newer type and more flexible than dbm. It still has most of the arbitrary limits of dbm (therefore it is deprecated).
- gdbm is the GNU database manager ().
- db2 is Sleepycat Software's DB2 (). It is described as "a programmatic toolkit that provides high-performance built-in database support for both standalone and client/server applications."
- cdb is "a fast, reliable, lightweight package for creating and reading constant databases." It is from the author of qmail and can be found here (). Since it is constant, we support only reading operations.

Example 1. DBA example

```
<?php

$id = dba_open("/tmp/test.db", "n", "db2");

if(!$id) {
    echo "dba_open failed\n";
    exit;
}

dba_replace("key", "This is an example!", $id);

if(dba_exists("key", $id)) {
    echo dba_fetch("key", $id);
    dba_delete("key", $id);
}

dba_close($id);
?>
```

DBA is binary safe and does not have any arbitrary limits. It inherits all limits set by the underlying database implementation.

All file-based databases must provide a way of setting the file mode of a new created database, if that is possible at all. The file mode is commonly passed as the fourth argument to **dba_open()** or **dba_popen()**.

You can access all entries of a database in a linear way by using the **dba_firstkey()** and **dba_nextkey()** functions. You may not change the database while traversing it.

Example 2. Traversing a database

```
<?php

# ...open database...

$key = dba_firstkey($id);

while($key != false) {
    if(...) { # remember the key to perform some action later
        $handle_later[] = $key;
    }
    $key = dba_nextkey($id);
}

for($i = 0; $i < count($handle_later); $i++)
    dba_delete($handle_later[$i], $id);

?>
```

dba_close (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Close database

```
void dba_close (int handle)
```

dba_close() closes the established database and frees all resources specified by *handle*.
handle is a database handle returned by **dba_open()**.
dba_close() does not return any value.
See also: **dba_open() dba_popen()**

dba_delete (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Delete entry specified by key

```
string dba_delete (string key, int handle)
```

dba_delete() deletes the entry specified by *key* from the database specified with *handle*.
key is the key of the entry which is deleted.
handle is a database handle returned by **dba_open()**.
dba_delete() returns true or false, if the entry is deleted or not deleted, respectively.
See also: **dba_exists() dba_fetch() dba_insert() dba_replace()**

dba_exists (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Check whether key exists

```
bool dba_exists (string key, int handle)
```

dba_exists() checks whether the specified *key* exists in the database specified by *handle*.
key is the key the check is performed for.
handle is a database handle returned by **dba_open()**.
dba_exists() returns true or false, if the key is found or not found, respectively.
See also: **dba_fetch() dba_delete() dba_insert() dba_replace()**

dba_fetch (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Fetch data specified by key

```
string dba_fetch (string key, int handle)
```

dba_fetch() fetches the data specified by *key* from the database specified with *handle*.

key is the key the data is specified by.

handle is a database handle returned by **dba_open()**.

dba_fetch() returns the associated string or false, if the key/data pair is found or not found, respectively.

See also: **dba_exists() dba_delete() dba_insert() dba_replace()**

dba_firstkey (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Fetch first key

```
string dba_firstkey (int handle)
```

dba_firstkey() returns the first key of the database specified by *handle* and resets the internal key pointer. This permits a linear search through the whole database.

handle is a database handle returned by **dba_open()**.

dba_firstkey() returns the key or false depending on whether it succeeds or fails, respectively.

See also: **dba_nextkey()**

dba_insert (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Insert entry

```
bool dba_insert (string key, string value, int handle)
```

dba_insert() inserts the entry described with *key* and *value* into the database specified by *handle*. It fails, if an entry with the same *key* already exists.

key is the key of the entry to be inserted.

value is the value to be inserted.

handle is a database handle returned by **dba_open()**.

dba_insert() returns true or false, depending on whether it succeeds of fails, respectively.

See also: **dba_exists() dba_delete() dba_fetch() dba_replace()**

dba_nextkey (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Fetch next key

```
string dba_nextkey (int handle)
```

dba_nextkey() returns the next key of the database specified by *handle* and increments the internal key pointer.

handle is a database handle returned by **dba_open()**.

dba_nextkey() returns the key or false depending on whether it succeeds or fails, respectively.

See also: **dba_firstkey()**

dba_popen (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Open database persistently

```
int dba_popen (string path, string mode, string handler [, ...])
```

dba_popen() establishes a persistent database instance for *path* with *mode* using *handler*.

path is commonly a regular path in your filesystem.

mode is "r" for read access, "w" for read/write access to an already existing database, "c" for read/write access and database creation if it doesn't currently exist, and "n" for create, truncate and read/write access.

handler is the name of the handler which shall be used for accessing *path*. It is passed all optional parameters given to **dba_popen()** and can act on behalf of them.

dba_popen() returns a positive handler id or false, in the case the open is successful or fails, respectively.

See also: **dba_open() dba_close()**

dba_open (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Open database

```
int dba_open (string path, string mode, string handler [, ...])
```

dba_open() establishes a database instance for *path* with *mode* using *handler*.

path is commonly a regular path in your filesystem.

mode is "r" for read access, "w" for read/write access to an already existing database, "c" for read/write access and database creation if it doesn't currently exist, and "n" for create, truncate and read/write access.

handler is the name of the handler which shall be used for accessing *path*. It is passed all optional parameters given to **dba_open()** and can act on behalf of them.

dba_open() returns a positive handler id or false, in the case the open is successful or fails, respectively.

See also: **dba_popen() dba_close()**

dba_optimize (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Optimize database

```
bool dba_optimize (int handle)
```

dba_optimize() optimizes the underlying database specified by *handle*.

handle is a database handle returned by **dba_open()**.

dba_optimize() returns true or false, if the optimization succeeds or fails, respectively.

See also: **dba_sync()**

dba_replace (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Replace or insert entry

```
bool dba_replace (string key, string value, int handle)
```

dba_replace() replaces or inserts the entry described with *key* and *value* into the database specified by *handle*.

key is the key of the entry to be inserted.

value is the value to be inserted.

handle is a database handle returned by **dba_open()**.

dba_replace() returns true or false, depending on whether it succeeds of fails, respectively.

See also: **dba_exists() dba_delete() dba_fetch() dba_insert()**

dba_sync (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Synchronize database

```
bool dba_sync (int handle)
```

dba_sync() synchronizes the database specified by *handle*. This will probably trigger a physical write to disk, if supported.

handle is a database handle returned by **dba_open()**.

dba_sync() returns true or false, if the synchronization succeeds or fails, respectively.

See also: **dba_optimize()**

XIII. Date and Time functions

checkdate (PHP3 , PHP4)

Validate a date/time

```
int checkdate (int month, int day, int year)
```

Returns true if the date given is valid; otherwise returns false. Checks the validity of the date formed by the arguments. A date is considered valid if:

- year is between 0 and 32767 inclusive
- month is between 1 and 12 inclusive
- *Day* is within the allowed number of days for the given *month*. Leap *year*s are taken into consideration.

date (PHP3 , PHP4)

Format a local time/date

```
string date (string format [, int timestamp])
```

Returns a string formatted according to the given format string using the given *timestamp* or the current local time if no timestamp is given.

The following characters are recognized in the format string:

- a - "am" or "pm"
- A - "AM" or "PM"
- d - day of the month, 2 digits with leading zeros; i.e. "01" to "31"
- D - day of the week, textual, 3 letters; i.e. "Fri"
- F - month, textual, long; i.e. "January"
- h - hour, 12-hour format; i.e. "01" to "12"
- H - hour, 24-hour format; i.e. "00" to "23"
- g - hour, 12-hour format without leading zeros; i.e. "1" to "12"
- G - hour, 24-hour format without leading zeros; i.e. "0" to "23"
- i - minutes; i.e. "00" to "59"
- j - day of the month without leading zeros; i.e. "1" to "31"
- l (lowercase 'L') - day of the week, textual, long; i.e. "Friday"
- L - boolean for whether it is a leap year; i.e. "0" or "1"
- m - month; i.e. "01" to "12"
- n - month without leading zeros; i.e. "1" to "12"
- M - month, textual, 3 letters; i.e. "Jan"
- s - seconds; i.e. "00" to "59"

- S - English ordinal suffix, textual, 2 characters; i.e. "th", "nd"
- t - number of days in the given month; i.e. "28" to "31"
- U - seconds since the epoch
- w - day of the week, numeric, i.e. "0" (Sunday) to "6" (Saturday)
- Y - year, 4 digits; i.e. "1999"
- y - year, 2 digits; i.e. "99"
- z - day of the year; i.e. "0" to "365"
- Z - timezone offset in seconds (i.e. "-43200" to "43200")

Unrecognized characters in the format string will be printed as-is. The "Z" format will always return "0" when using **gmdate()**.

Example 1. Date() example

```
print (date ("l dS of F Y h:i:s A"));
print ("July 1, 2000 is on a " . date ("l", mktime(0,0,0,7,1,2000)));
```

It is possible to use **date()** and **mktime()** together to find dates in the future or the past.

Example 2. Date() and mktime() example

```
$tomorrow  = mktime (0,0,0,date("m")  ,date("d")+1,date("Y"));
$lastmonth = mktime (0,0,0,date("m")-1,date("d"),  date("Y"));
$nextyear  = mktime (0,0,0,date("m"),  date("d"),  date("Y")+1);
```

To format dates in other languages, you should use the **setlocale()** and **strftime()** functions.

See also **gmdate()** and **mktime()**.

getdate (PHP3 , PHP4)

Get date/time information

```
array getdate (int timestamp)
```

Returns an associative array containing the date information of the *timestamp* as the following array elements:

- "seconds" - seconds
- "minutes" - minutes
- "hours" - hours
- "mday" - day of the month
- "wday" - day of the week, numeric
- "mon" - month, numeric

- "year" - year, numeric
- "yday" - day of the year, numeric; i.e. "299"
- "weekday" - day of the week, textual, full; i.e. "Friday"
- "month" - month, textual, full; i.e. "January"

gettimeofday (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Get current time

```
array gettimeofday (void)
```

This is an interface to gettimeofday(2). It returns an associative array containing the data returned from the system call.

- "sec" - seconds
- "usec" - microseconds
- "minuteswest" - minutes west of Greenwich
- "dsttime" - type of dst correction

gmdate (PHP3 , PHP4)

Format a GMT/CUT date/time

```
string gmdate (string format, int timestamp)
```

Identical to the **date()** function except that the time returned is Greenwich Mean Time (GMT). For example, when run in Finland (GMT +0200), the first line below prints "Jan 01 1998 00:00:00", while the second prints "Dec 31 1997 22:00:00".

Example 1. Gmdate() example

```
echo date ("M d Y H:i:s", mktime (0,0,0,1,1,1998));
echo gmdate ("M d Y H:i:s", mktime (0,0,0,1,1,1998));
```

See also **date()**, **mktime()**, and **gmmktime()**.

gmmktime (PHP3 , PHP4)

Get UNIX timestamp for a GMT date

```
int gmmktime (int hour, int minute, int second, int month, int day, int year [,
int is_dst])
```

Identical to **mktime()** except the passed parameters represents a GMT date.

gmstrftime (PHP3 >= 3.0.12, PHP4 >= 4.0RC2)

Format a GMT/CUT time/date according to locale settings

```
string gmstrftime (string format, int timestamp)
```

Behaves the same as **strftime()** except that the time returned is Greenwich Mean Time (GMT). For example, when run in Eastern Standard Time (GMT -0500), the first line below prints "Dec 31 1998 20:00:00", while the second prints "Jan 01 1999 01:00:00".

Example 1. Gmstrftime() example

```
setlocale ('LC_TIME', 'en_US');
echo strftime ("%b %d %Y %H:%M:%S", mktime (20,0,0,12,31,98))."\n";
echo gmstrftime ("%b %d %Y %H:%M:%S", mktime (20,0,0,12,31,98))."\n";
```

See also **strftime()**.

localtime (PHP4 >= 4.0RC2)

Get the local time

```
array localtime ([int timestamp [, bool is_associative]])
```

The **localtime()** function returns an array identical to that of the structure returned by the C function call. The first argument to **localtime()** is the timestamp, if this is not given the current time is used. The second argument to the **localtime()** is the *is_associative*, if this is set to 0 or not supplied than the array is returned as a regular, numerically indexed array. If the argument is set to 1 then **localtime()** is an associative array containing all the different elements of the structure returned by the C function call to localtime. The names of the different keys of the associative array are as follows:

- "tm_sec" - seconds
- "tm_min" - minutes
- "tm_hour" - hour
- "tm_mday" - day of the month
- "tm_mon" - month of the year
- "tm_year" - Year, not y2k compliant
- "tm_wday" - Day of the week
- "tm_yday" - Day of the year

- "tm_isdst" - Is daylight savings time in effect

microtime (PHP3 , PHP4)

Return current UNIX timestamp with microseconds

```
string microtime(void);
```

Returns the string "msec sec" where sec is the current time measured in the number of seconds since the Unix Epoch (0:00:00 January 1, 1970 GMT), and msec is the microseconds part. This function is only available on operating systems that support the gettimeofday() system call.

See also **time()**.

mktime (PHP3 , PHP4)

Get UNIX timestamp for a date

```
int mktime (int hour, int minute, int second, int month, int day, int year [,
int is_dst])
```

Warning: Note the strange order of arguments, which differs from the order of arguments in a regular UNIX mktime() call and which does not lend itself well to leaving out parameters from right to left (see below). It is a common error to mix these values up in a script.

Returns the Unix timestamp corresponding to the arguments given. This timestamp is a long integer containing the number of seconds between the Unix Epoch (January 1 1970) and the time specified.

Arguments may be left out in order from right to left; any arguments thus omitted will be set to the current value according to the local date and time.

Is_dst can be set to 1 if the time is during daylight savings time, 0 if it is not, or -1 (the default) if it is unknown whether the time is within daylight savings time or not.

> **Note:** *Is_dst* was added in 3.0.10.

Mktime() is useful for doing date arithmetic and validation, as it will automatically calculate the correct value for out-of-range input. For example, each of the following lines produces the string "Jan-01-1998".

Example 1. Mktime() example

```
echo date ("M-d-Y", mktime (0,0,0,12,32,1997));
echo date ("M-d-Y", mktime (0,0,0,13,1,1997));
echo date ("M-d-Y", mktime (0,0,0,1,1,1998));
echo date ("M-d-Y", mktime (0,0,0,1,1,98));
```

Year may be a two or four digit value, with values between 0-69 mapping to 2000-2069 and 70-99 to 1970-1999 (on systems where time_t is a 32bit signed integer, as most common today, the valid range for *year* is somewhere between 1902 and 2037).

The last day of any given month can be expressed as the "0" day of the next month, not the -1 day. Both of the following examples will produce the string "The last day in Feb 2000 is: 29".

Example 2. Last day of next month

```
$lastday = mktime (0,0,0,3,0,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);

$lastday = mktime (0,0,0,4,-31,2000);
echo strftime ("Last day in Feb 2000 is: %d", $lastday);
```

See also **date()** and **time()**.

strftime (PHP3 , PHP4)

Format a local time/date according to locale settings

```
string strftime (string format, int timestamp)
```

Returns a string formatted according to the given format string using the given *timestamp* or the current local time if no timestamp is given. Month and weekday names and other language dependent strings respect the current locale set with **setlocale()**.

The following conversion specifiers are recognized in the format string:

- %a - abbreviated weekday name according to the current locale
- %A - full weekday name according to the current locale
- %b - abbreviated month name according to the current locale
- %B - full month name according to the current locale
- %c - preferred date and time representation for the current locale
- %C - century number (the year divided by 100 and truncated to an integer, range 00 to 99)
- %d - day of the month as a decimal number (range 00 to 31)
- %D - same as %m/%d/%y
- %e - day of the month as a decimal number, a single digit is preceded by a space (range ’ 1’ to ’31’)
- %h - same as %b
- %H - hour as a decimal number using a 24-hour clock (range 00 to 23)
- %I - hour as a decimal number using a 12-hour clock (range 01 to 12)
- %j - day of the year as a decimal number (range 001 to 366)
- %m - month as a decimal number (range 1 to 12)
- %M - minute as a decimal number

- %n - newline character
- %p - either ‘am’ or ‘pm’ according to the given time value, or the corresponding strings for the current locale
- %r - time in a.m. and p.m. notation
- %R - time in 24 hour notation
- %S - second as a decimal number
- %t - tab character
- %T - current time, equal to %H:%M:%S
- %u - weekday as a decimal number [1,7], with 1 representing Monday
- %U - week number of the current year as a decimal number, starting with the first Sunday as the first day of the first week
- %V - The ISO 8601:1988 week number of the current year as a decimal number, range 01 to 53, where week 1 is the first week that has at least 4 days in the current year, and with Monday as the first day of the week.
- %W - week number of the current year as a decimal number, starting with the first Monday as the first day of the first week
- %w - day of the week as a decimal, Sunday being 0
- %x - preferred date representation for the current locale without the time
- %X - preferred time representation for the current locale without the date
- %y - year as a decimal number without a century (range 00 to 99)
- %Y - year as a decimal number including the century
- %Z - time zone or name or abbreviation
- %% - a literal ‘%’ character

Example 1. Strftime() example

```
setlocale ("LC_TIME", "C");
print (strftime ("%A in Finnish is "));
setlocale ("LC_TIME", "fi_FI");
print (strftime ("%A, in French "));
setlocale ("LC_TIME", "fr_CA");
print (strftime ("%A and in German "));
setlocale ("LC_TIME", "de_DE");
print (strftime ("%A.\n"));
```

This example works if you have the respective locales installed in your system.

See also **setlocale()** and **mktime()** and the Open Group specification of **strftime()()** (http://www.opengroup.org/onlinepubs/7908799/xsh/strftime.html).

time (PHP3 , PHP4)

Return current UNIX timestamp

```
int time(void);
```

Returns the current time measured in the number of seconds since the Unix Epoch (January 1 1970 00:00:00 GMT).

See also **date()**.

strtotime (PHP3 >= 3.0.12, PHP4 >= 4.0b2)

Parse about any english textual datetime description into a UNIX timestamp

```
int strtotime (string time [, int now])
```

The function expects to be given a string containing an english date format and will try to parse that format into a UNIX timestamp.

Example 1. Strtotime() example

```
echo strtotime ("10 march 2000") . "\n";
```

XIV. dBase functions

These functions allow you to access records stored in dBase-format (dbf) databases.

There is no support for indexes or memo fields. There is no support for locking, too. Two concurrent webserver processes modifying the same dBase file will very likely ruin your database.

Unlike SQL databases, dBase "databases" cannot change the database definition afterwards. Once the file is created, the database definition is fixed. There are no indexes that speed searching or otherwise organize your data. dBase files are simple sequential files of fixed length records. Records are appended to the end of the file and delete records are kept until you call **dbase_pack()()**.

We recommend that you do not use dBase files as your production database. Choose any real SQL server instead; MySQL or Postgres are common choices with PHP. dBase support is here to allow you to import and export data to and from your web database, since the file format is commonly understood with Windows spreadsheets and organizers. Import and export of data is about all that dBase support is good for.

dbase_create (PHP3 , PHP4)

Creates a dBase database

```
int dbase_create (string filename, array fields)
```

The *fields* parameter is an array of arrays, each array describing the format of one field in the database. Each field consists of a name, a character indicating the field type, a length, and a precision.

The types of fields available are:

L

Boolean. These do not have a length or precision.

M

Memo. (Note that these aren't supported by PHP.) These do not have a length or precision.

D

Date (stored as YYYYMMDD). These do not have a length or precision.

N

Number. These have both a length and a precision (the number of digits after the decimal point).

C

String.

If the database is successfully created, a dbase_identifier is returned, otherwise false is returned.

Example 1. Creating a dBase database file

```
// "database" name
$dbname = "/tmp/test.dbf";

// database "definition"
$def =
    array(
        array("date",     "D"),
        array("name",     "C",  50),
        array("age",      "N",   3, 0),
        array("email",    "C", 128),
        array("ismember", "L")
    );

// creation
if (!dbase_create($dbname, $def))
    print "<strong>Error!</strong>";
```

dbase_open (PHP3 , PHP4)

Opens a dBase database

```
int dbase_open (string filename, int flags)
```

The flags correspond to those for the open() system call. (Typically 0 means read-only, 1 means write-only, and 2 means read and write.)

Returns a dbase_identifier for the opened database, or false if the database couldn't be opened.

dbase_close (PHP3 , PHP4)

Close a dBase database

```
bool dbase_close (int dbase_identifier)
```

Closes the database associated with *dbase_identifier*.

dbase_pack (PHP3 , PHP4)

Packs a dBase database

```
bool dbase_pack (int dbase_identifier)
```

Packs the specified database (permanently deleting all records marked for deletion using **dbase_delete_record()**.

dbase_add_record (PHP3 , PHP4)

Add a record to a dBase database

```
bool dbase_add_record (int dbase_identifier, array record)
```

Adds the data in the *record* to the database. If the number of items in the supplied record isn't equal to the number of fields in the database, the operation will fail and false will be returned.

dbase_replace_record (PHP3 >= 3.0.11, PHP4)

Replace a record in a dBase database

```
bool dbase_replace_record (int dbase_identifier, array record, int
dbase_record_number)
```

Replaces the data associated with the record *record_number* with the data in the *record* in the database. If the number of items in the supplied record is not equal to the number of fields in the database, the operation will fail and false will be returned.

dbase_record_number is an integer which spans from 1 to the number of records in the database (as returned by **dbase_numrecords()**).

dbase_delete_record (PHP3 , PHP4)

Deletes a record from a dBase database

```
bool dbase_delete_record (int dbase_identifier, int record)
```

Marks *record* to be deleted from the database. To actually remove the record from the database, you must also call **dbase_pack()**.

dbase_get_record (PHP3 , PHP4)

Gets a record from a dBase database

```
array dbase_get_record (int dbase_identifier, int record)
```

Returns the data from *record* in an array. The array is indexed starting at 0, and includes an associative member named 'deleted' which is set to 1 if the record has been marked for deletion (see **dbase_delete_record()**.

Each field is converted to the appropriate PHP type. (Dates are left as strings.)

dbase_get_record_with_names (PHP3 >= 3.0.4, PHP4)

Gets a record from a dBase database as an associative array

```
array dbase_get_record_with_names  (int dbase_identifier, int record)
```

Returns the data from *record* in an associative array. The array also includes an associative member named 'deleted' which is set to 1 if the record has been marked for deletion (see **dbase_delete_record()**.

Each field is converted to the appropriate PHP type. (Dates are left as strings.)

dbase_numfields (PHP3 , PHP4)

Find out how many fields are in a dBase database

```
int dbase_numfields (int dbase_identifier)
```

Returns the number of fields (columns) in the specified database. Field numbers are between 0 and dbase_numfields($db)-1, while record numbers are between 1 and dbase_numrecords($db).

Example 1. Using dbase_numfields()

```
$rec = dbase_get_record($db, $recno);
$nf  = dbase_numfields($db);
for ($i=0; $i < $nf; $i++) {
    print $rec[$i]."<br>\n";
}
```

dbase_numrecords (PHP3 , PHP4)

Find out how many records are in a dBase database

```
int dbase_numrecords (int dbase_identifier)
```

Returns the number of records (rows) in the specified database. Record numbers are between 1 and dbase_numrecords($db), while field numbers are between 0 and dbase_numfields($db)-1.

XV. dbm functions

These functions allow you to store records stored in a dbm-style database. This type of database (supported by the Berkeley db, gdbm, and some system libraries, as well as a built-in flatfile library) stores key/value pairs (as opposed to the full-blown records supported by relational databases).

Example 1. dbm example

```
$dbm = dbmopen("lastseen", "w");
if (dbmexists($dbm, $userid)) {
  $last_seen = dbmfetch($dbm, $userid);
} else {
  dbminsert($dbm, $userid, time());
}
do_stuff();
dbmreplace($dbm, $userid, time());
dbmclose($dbm);
```

dbmopen (PHP3 , PHP4)

opens a dbm database

```
int dbmopen (string filename, string flags)
```

The first argument is the full-path filename of the dbm file to be opened and the second is the file open mode which is one of "r", "n", "c" or "w" for read-only, new (implies read-write, and most likely will truncate an already-existing database of the same name), create (implies read-write, and will not truncate an already-existing database of the same name) and read-write respectively.

Returns an identifer to be passed to the other dbm functions on success, or false on failure.

If ndbm support is used, ndbm will actually create filename.dir and filename.pag files. gdbm only uses one file, as does the internal flat-file support, and Berkeley db creates a filename.db file. Note that PHP does its own file locking in addition to any file locking that may be done by the dbm library itself. PHP does not delete the .lck files it creates. It uses these files simply as fixed inodes on which to do the file locking. For more information on dbm files, see your Unix man pages, or obtain GNU's gdbm from `ftp://prep.ai.mit.edu/pub/gnu`.

dbmclose (PHP3 , PHP4)

closes a dbm database

```
bool dbmclose (int dbm_identifier)
```

Unlocks and closes the specified database.

dbmexists (PHP3 , PHP4)

tells if a value exists for a key in a dbm database

```
bool dbmexists (int dbm_identifier, string key)
```

Returns true if there is a value associated with the *key*.

dbmfetch (PHP3 , PHP4)

fetches a value for a key from a dbm database

```
string dbmfetch (int dbm_identifier, string key)
```

Returns the value associated with *key*.

dbminsert (PHP3 , PHP4)

inserts a value for a key in a dbm database

```
int dbminsert (int dbm_identifier, string key, string value)
```

Adds the value to the database with the specified key.

Returns -1 if the database was opened read-only, 0 if the insert was successful, and 1 if the specified key already exists. (To replace the value, use **dbmreplace()**.)

dbmreplace (PHP3 , PHP4)

replaces the value for a key in a dbm database

```
bool dbmreplace (int dbm_identifier, string key, string value)
```

Replaces the value for the specified key in the database.

This will also add the key to the database if it didn't already exist.

dbmdelete (PHP3 , PHP4)

deletes the value for a key from a dbm database

```
bool dbmdelete (int dbm_identifier, string key)
```

Deletes the value for *key* in the database.

Returns false if the key didn't exist in the database.

dbmfirstkey (PHP3 , PHP4)

retrieves the first key from a dbm database

```
string dbmfirstkey (int dbm_identifier)
```

Returns the first key in the database. Note that no particular order is guaranteed since the database may be built using a hash-table, which doesn't guarantee any ordering.

dbmnextkey (PHP3 , PHP4)

retrieves the next key from a dbm database

```
string dbmnextkey (int dbm_identifier, string key)
```

Returns the next key after *key*. By calling **dbmfirstkey()** followed by successive calls to **dbmnextkey()** it is possible to visit every key/value pair in the dbm database. For example:

Example 1. Visiting every key/value pair in a dbm database.

```
$key = dbmfirstkey($dbm_id);
while ($key) {
    echo "$key = " . dbmfetch($dbm_id, $key) . "\n";
    $key = dbmnextkey($dbm_id, $key);
}
```

dblist (PHP3 , PHP4)

describes the dbm-compatible library being used

```
string dblist (void)
```

XVI. Directory functions

chdir (PHP3 , PHP4)

change directory

```
int chdir (string directory)
```

Changes PHP's current directory to *directory*. Returns FALSE if unable to change directory, TRUE otherwise.

dir (PHP3 , PHP4)

directory class

```
new dir (string directory)
```

A pseudo-object oriented mechanism for reading a directory. The given *directory* is opened. Two properties are available once directory has been opened. The handle property can be used with other directory functions such as **readdir()**, **rewinddir()** and **closedir()**. The path property is set to path the directory that was opened. Three methods are available: read, rewind and close.

Example 1. Dir() Example

```
$d = dir("/etc");
echo "Handle: ".$d->handle."<br>\n";
echo "Path: ".$d->path."<br>\n";
while($entry=$d->read()) {
    echo $entry."<br>\n";
}
$d->close();
```

closedir (PHP3 , PHP4)

close directory handle

```
void closedir (int dir_handle)
```

Closes the directory stream indicated by *dir_handle*. The stream must have previously been opened by **opendir()**.

opendir (PHP3 , PHP4)

open directory handle

```
int opendir (string path)
```

Returns a directory handle to be used in subsequent **closedir()**, **readdir()**, and **rewinddir()** calls.

readdir (PHP3 , PHP4)

read entry from directory handle

```
string readdir (int dir_handle)
```

Returns the filename of the next file from the directory. The filenames are not returned in any particular order.

Example 1. List all files in the current directory

```
<?php
$handle=opendir('.');
echo "Directory handle: $handle\n";
echo "Files:\n";
while ($file = readdir($handle)) {
    echo "$file\n";
}
closedir($handle);
?>
```

Note that **readdir()** will return the . and .. entries. If you don't want these, simply strip them out:

Example 2. List all files in the current directory and strip out . and ..

```
<?php
$handle=opendir('.');
while ($file = readdir($handle)) {
    if ($file != "." && $file != "..") {
        echo "$file\n";
    }
}
closedir($handle);
?>
```

rewinddir (PHP3 , PHP4)

rewind directory handle

```
void rewinddir (int dir_handle)
```

Resets the directory stream indicated by *dir_handle* to the beginning of the directory.

XVII. Dynamic Loading functions

dl (PHP3 , PHP4)

load a PHP extension at runtime

```
int dl (string library)
```

Loads the PHP extension defined in *library*. See also the extension_dir configuration directive.

XVIII. 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 (PHP3 >= 3.0.8, PHP4)

Get the name of the specified cipher

```
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 (PHP3 >= 3.0.8, PHP4)

Get the block size of the specified cipher

```
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 (PHP3 >= 3.0.8, PHP4)

Get the key size of the specified cipher

```
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 (PHP3 >= 3.0.8, PHP4)

Create an initialization vector (IV) from a random source

```
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 (PHP3 >= 3.0.8, PHP4)

Encrypt/decrypt data in CBC mode

```
string 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()**, and **mcrypt_ofb()**.

mcrypt_cfb (PHP3 >= 3.0.8, PHP4)

Encrypt/decrypt data in CFB mode

```
string 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()**, and **mcrypt_ofb()**.

mcrypt_ecb (PHP3 >= 3.0.8, PHP4)

Encrypt/decrypt data in ECB mode

```
string 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()**, and **mcrypt_ofb()**.

mcrypt_ofb (PHP3 >= 3.0.8, PHP4)

Encrypt/decrypt data in OFB mode

```
string 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()**, and **mcrypt_ecb()**.

XIX. filePro functions

These functions allow read-only access to data stored in filePro databases.

filePro is a registered trademark of Fiserv, Inc. You can find more information about filePro at http://www.fileproplus.com/.

filepro (PHP3 , PHP4)

read and verify the map file

```
bool filepro (string directory)
```

This reads and verifies the map file, storing the field count and info.
No locking is done, so you should avoid modifying your filePro database while it may be opened in PHP.

filepro_fieldname (PHP3 , PHP4)

gets the name of a field

```
string filepro_fieldname (int field_number)
```

Returns the name of the field corresponding to *field_number*.

filepro_fieldtype (PHP3 , PHP4)

gets the type of a field

```
string filepro_fieldtype (int field_number)
```

Returns the edit type of the field corresponding to *field_number*.

filepro_fieldwidth (PHP3 , PHP4)

gets the width of a field

```
int filepro_fieldwidth (int field_number)
```

Returns the width of the field corresponding to *field_number*.

filepro_retrieve (PHP3 , PHP4)

retrieves data from a filePro database

```
string filepro_retrieve (int row_number, int field_number)
```

Returns the data from the specified location in the database.

filepro_fieldcount (PHP3 , PHP4)

find out how many fields are in a filePro database

```
int filepro_fieldcount(void);
```

Returns the number of fields (columns) in the opened filePro database.
See also **filepro()**.

filepro_rowcount (PHP3 , PHP4)

find out how many rows are in a filePro database

```
int filepro_rowcount(void);
```

Returns the number of rows in the opened filePro database.
See also **filepro()**.

XX. Filesystem functions

basename (PHP3 , PHP4)

Return filename component of path

```
string basename (string path)
```

Given a string containing a path to a file, this function will return the base name of the file.

On Windows, both slash (/) and backslash (\) are used as path separator character. In other environments, it is the forward slash (/).

Example 1. basename() example

```
$path = "/home/httpd/html/index.php3";
$file = basename($path); // $file is set to "index.php3"
```

See also: **dirname()**

chgrp (PHP3 , PHP4)

Change file group

```
int chgrp (string filename, mixed group)
```

Attempts to change the group of the file filename to group. Only the superuser may change the group of a file arbitrarily; other users may change the group of a file to any group of which that user is a member.

Returns true on success; otherwise returns false.

See also **chown()** and **chmod()**.

> **Note:** This function does not work on Windows systems

chmod (PHP3 , PHP4)

Change file mode

```
int chmod (string filename, int mode)
```

Attempts to change the mode of the file specified by *filename* to that given in *mode*.

Note that *mode* is not automatically assumed to be an octal value. To ensure the expected operation, you need to prefix *mode* with a zero (0):

```
chmod( "/somedir/somefile", 755 );   // decimal; probably incorrect
chmod( "/somedir/somefile", 0755 );  // octal; correct value of mode
```

Returns true on success and false otherwise.

See also **chown()** and **chgrp()**.

> **Note:** This function does not work on Windows systems

chown (PHP3 , PHP4)

change file owner

```
int chown (string filename, mixed user)
```

Attempts to change the owner of the file filename to user user. Only the superuser may change the owner of a file.

Returns true on success; otherwise returns false.

> **Note:** On Windows, does nothing and returns true.

See also **chown()** and **chmod()**.

> **Note:** This function does not work on Windows systems

clearstatcache (PHP3 , PHP4)

Clear file stat cache

```
void clearstatcache(void);
```

Invoking the stat or lstat system call on most systems is quite expensive. Therefore, the result of the last call to any of the status functions (listed below) is stored for use on the next such call using the same filename. If you wish to force a new status check, for instance if the file is being checked many times and may change or disappear, use this function to clear the results of the last call from memory.

This value is only cached for the lifetime of a single request.

Affected functions include **stat()**, **lstat()**, **file_exists()**, **is_writeable()**, **is_readable()**, **is_executable()**, **is_file()**, **is_dir()**, **is_link()**, **filectime()**, **fileatime()**, **filemtime()**, **fileinode()**, **filegroup()**, **fileowner()**, **filesize()**, **filetype()**, and **fileperms()**.

copy (PHP3 , PHP4)

Copy file

```
int copy (string source, string dest)
```

Makes a copy of a file. Returns true if the copy succeeded, false otherwise.

Example 1. copy() example

```
if ( !copy($file, $file.'.bak') ) {
    print("failed to copy $file...<br>\n");
}
```

See also: **rename()**.

delete (unknown)

A dummy manual entry

```
void delete (string file)
```

This is a dummy manual entry to satisfy those people who are looking for **unlink()** or **unset()** in the wrong place.

See also: **unlink()** to delete files, **unset()** to delete variables.

dirname (PHP3 , PHP4)

Return directory name component of path

```
string dirname (string path)
```

Given a string containing a path to a file, this function will return the name of the directory.

On Windows, both slash (/) and backslash (\) are used as path separator character. In other environments, it is the forward slash (/).

Example 1. dirname() example

```
$path = "/etc/passwd";
$file = dirname($path); // $file is set to "/etc"
```

See also: **basename()**

diskfreespace (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Return available space in directory

```
float diskfreespace (string directory)
```

Given a string containing a directory, this function will return the number of bytes available on the corresponding disk.

Example 1. diskfreespace() example

```
$df = diskfreespace("/"); // $df contains the number of bytes
                          // available on "/"
```

fclose (PHP3 , PHP4)

Close an open file pointer

```
int fclose (int fp)
```

The file pointed to by fp is closed.

Returns true on success and false on failure.

The file pointer must be valid, and must point to a file successfully opened by **fopen()** or **fsockopen()**.

feof (PHP3 , PHP4)

Test for end-of-file on a file pointer

```
int feof (int fp)
```

Returns true if the file pointer is at EOF or an error occurs; otherwise returns false.

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsockopen()**.

fgetc (PHP3 , PHP4)

Get character from file pointer

```
string fgetc (int fp)
```

Returns a string containing a single character read from the file pointed to by fp. Returns FALSE on EOF (as does **feof()**).

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsockopen()**.

See also **fread()**, **fopen()**, **popen()**, **fsockopen()**, and **fgets()**.

fgetcsv (PHP3 >= 3.0.8, PHP4)

Get line from file pointer and parse for CSV fields

```
array fgetcsv (int fp, int length [, string delimiter])
```

Similar to fgets() except that fgetcsv() parses the line it reads for fields in CSV format and returns an array containing the fields read. The field delimiter is a comma, unless you specifiy another delimiter with the optional third parameter.

fp must be a valid file pointer to a file successfully opened by **fopen()**, **popen()**, or **fsockopen()**

length must be greater than the longest line to be found in the CSV file (allowing for trailing line-end characters).

fgetcsv() returns false on error, including end of file.

NB A blank line in a CSV file will be returned as an array comprising just one single null field, and will not be treated as an error.

Example 1. Fgetcsv() example - Read and print entire contents of a CSV file

```
$row = 1;
$fp = fopen ("test.csv","r");
while ($data = fgetcsv ($fp, 1000, ",")) {
    $num = count ($data);
    print "<p> $num fields in line $row: <br>";
    $row++;
    for ($c=0; $c<$num; $c++) {
        print $data[$c] . "<br>";
    }
}
fclose ($fp);
```

fgets (PHP3 , PHP4)

Get line from file pointer

```
string fgets (int fp, int length)
```

Returns a 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 (which is included in the return value), or on EOF (whichever comes first).

If an error occurs, returns false.

Common Pitfalls:

People used to the 'C' semantics of fgets should note the difference in how EOF is returned.

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsockopen()**.

A simple example follows:

Example 1. Reading a file line by line

```
$fd = fopen ("/tmp/inputfile.txt", "r");
while (!feof($fd)) {
    $buffer = fgets($fd, 4096);
    echo $buffer;
}
fclose ($fd);
```

See also **fread()**, **fopen()**, **popen()**, **fgetc()**, and **fsockopen()**.

fgetss (PHP3 , PHP4)

Get line from file pointer and strip HTML tags

```
string fgetss (int fp, int length [, string allowable_tags])
```

Identical to **fgets()**, except that fgetss 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 **fgets()**, **fopen()**, **fsockopen()**, **popen()**, and **strip_tags()**.

file (PHP3 , PHP4)

read entire file into an array

```
array file (string filename [, int use_include_path])
```

Identical to **readfile()**, except that **file()** returns the file in an array. Each element of the array corresponds to a line in the file, with the newline still attached.

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.

```
<?php
// get a web page into an array and print it out
$fcontents = file( 'http://www.php.net' );
while ( list( $line_num, $line ) = each( $fcontents ) ) {
   echo "<b>Line $line_num:</b> " . htmlspecialchars( $line ) . "<br>\n";
}

// get a web page into a string
$fcontents = join( '', file( 'http://www.php.net' ) );
?>
```

See also **readfile()**, **fopen()**, and **popen()**.

file_exists (PHP3 , PHP4)

Check whether a file exists

```
int file_exists (string filename)
```

Returns true if the file specified by *filename* exists; false otherwise.

file_exists() will not work on remote files; the file to be examined must be accessible via the server's filesystem.

The results of this function are cached. See **clearstatcache()** for more details.

fileatime (PHP3 , PHP4)

Get last access time of file

```
int fileatime (string filename)
```

Returns the time the file was last accessed, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

filectime (PHP3 , PHP4)

Get inode change time of file

```
int filectime (string filename)
```

Returns the time the file was last changed, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

filegroup (PHP3 , PHP4)

Get file group

```
int filegroup (string filename)
```

Returns the group ID of the owner of the file, or false in case of an error. The group ID is returned in numerical format, use **posix_getgrgid()** to resolve it to a group name.

The results of this function are cached. See **clearstatcache()** for more details.

> **Note:** This function does not work on Windows systems

fileinode (PHP3 , PHP4)

Get file inode

```
int fileinode (string filename)
```

Returns the inode number of the file, or false in case of an error.

The results of this function are cached. See **clearstatcache()** for more details.

> **Note:** This function does not work on Windows systems

filemtime (PHP3 , PHP4)

Get file modification time

```
int filemtime (string filename)
```

Returns the time the file was last modified, or false in case of an error. The time is returned as a Unix timestamp.

The results of this function are cached. See **clearstatcache()** for more details.

fileowner (PHP3 , PHP4)

Get file owner

```
int fileowner (string filename)
```

Returns the user ID of the owner of the file, or false in case of an error. The user ID is returned in numerical format, use **posix_getpwuid()** to resolve it to a username.

The results of this function are cached. See **clearstatcache()** for more details.

> **Note:** This function does not work on Windows systems

fileperms (PHP3 , PHP4)

Get file permissions

```
int fileperms (string filename)
```

Returns the permissions on the file, or false in case of an error.

The results of this function are cached. See **clearstatcache()** for more details.

filesize (PHP3 , PHP4)

Get file size

```
int filesize (string filename)
```

Returns the size of the file, or false in case of an error.

The results of this function are cached. See **clearstatcache()** for more details.

filetype (PHP3 , PHP4)

Get file type

```
string filetype (string filename)
```

Returns the type of the file. Possible values are fifo, char, dir, block, link, file, and unknown.

Returns false if an error occurs.

The results of this function are cached. See **clearstatcache()** for more details.

flock (PHP3 >= 3.0.7, PHP4)

Portable advisory file locking

```
bool flock (int fp, int operation)
```

PHP supports a portable way of locking complete files in an advisory way (which means all accessing programs have to use the same way of locking or it will not work).

flock() operates on *fp* which must be an open file pointer. *operation* is one of the following values:

- To acquire a shared lock (reader), set *operation* to 1.
- To acquire an exclusive lock (writer), set *operation* to 2.
- To release a lock (shared or exclusive), set *operation* to 3.
- If you don't want **flock()** to block while locking, add 4 to *operation*.

flock() allows you to perform a simple reader/writer model which can be used on virtually every platform (including most Unices and even Windows).

flock() returns true on success and false on error (e.g. when a lock could not be acquired).

fopen (PHP3 , PHP4)

Open file or URL

```
int fopen (string filename, string mode [, int use_include_path])
```

If *filename* begins with "http://" (not case sensitive), an HTTP 1.0 connection is opened to the specified server and a file pointer is returned to the beginning of the text of the response.

Does not handle HTTP redirects, so you must include trailing slashes on directories.

If *filename* begins with "ftp://" (not case sensitive), an ftp connection to the specified server is opened and a pointer to the requested file is returned. If the server does not support passive mode ftp, this will fail. You can open files for either reading and writing via ftp (but not both simultaneously).

If *filename* is one of "php://stdin", "php://stdout", or "php://stderr", the corresponding stdio stream will be opened. (This was introduced in PHP 3.0.13; in earlier versions, a filename such as "/dev/stdin" or "/dev/fd/0" must be used to access the stdio streams.)

If *filename* begins with anything else, the file will be opened from the filesystem, and a file pointer to the file opened is returned.

If the open fails, the function returns false.

mode may be any of the following:

- 'r' - Open for reading only; place the file pointer at the beginning of the file.
- 'r+' - Open for reading and writing; place the file pointer at the beginning of the file.
- 'w' - Open for writing only; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.

- 'w+' - Open for reading and writing; place the file pointer at the beginning of the file and truncate the file to zero length. If the file does not exist, attempt to create it.
- 'a' - Open for writing only; place the file pointer at the end of the file. If the file does not exist, attempt to create it.
- 'a+' - Open for reading and writing; place the file pointer at the end of the file. If the file does not exist, attempt to create it.

As well, *mode* may contain the letter 'b'. This is useful only on systems which differentiate between binary and text files (i.e., it's useless on Unix). If not needed, this will be ignored.

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. fopen() example

```
$fp = fopen("/home/rasmus/file.txt", "r");
$fp = fopen("http://www.php.net/", "r");
$fp = fopen("ftp://user:password@example.com/", "w");
```

If you are experiencing problems with reading and writing to files and you're using the server module version of PHP, remember to make sure that the files and directories you're using are accessible to the server process.

On the Windows platform, be careful to escape any backslashes used in the path to the file, or use forward slashes.

```
$fp = fopen("c:\\data\\info.txt", "r");
```

See also **fclose()**, **fsockopen()**, and **popen()**.

fpassthru (PHP3 , PHP4)

Output all remaining data on a file pointer

```
int fpassthru (int fp)
```

Reads to EOF on the given file pointer and writes the results to standard output.

If an error occurs, **fpassthru()** returns false.

The file pointer must be valid, and must point to a file successfully opened by **fopen()**, **popen()**, or **fsockopen()**. The file is closed when **fpassthru()** is done reading it (leaving *fp* useless).

If you just want to dump the contents of a file to stdout you may want to use the **readfile()**, which saves you the **fopen()** call.

See also **readfile()**, **fopen()**, **popen()**, and **fsockopen()**

fputs (PHP3 , PHP4)

Write to a file pointer

```
int fputs (int fp, string str [, int length])
```

fputs() is an alias to **fwrite()**, and is identical in every way. Note that the *length* parameter is optional and if not specified the entire string will be written.

fread (PHP3 , PHP4)

Binary-safe file read

```
string fread (int fp, int length)
```

fread() reads up to *length* bytes from the file pointer referenced by *fp*. Reading stops when *length* bytes have been read or EOF is reached, whichever comes first.

```
// get contents of a file into a string
$filename = "/usr/local/something.txt";
$fd = fopen ($filename, "r");
$contents = fread ($fd, filesize ($filename));
fclose ($fd);
```

See also **fwrite()**, **fopen()**, **fsockopen()**, **popen()**, **fgets()**, **fgetss()**, **file()**, and **fpassthru()**.

fseek (PHP3 , PHP4)

Seek on a file pointer

```
int fseek (int fp, int offset [, int whence])
```

Sets the file position indicator for the file referenced by *fp*.The new position, measured in bytes from the beginning of the file, is obtained by adding *offset* to the position specified by *whence*, whose values are defined as follows:

SEEK_SET - Set position equal to *offset bytes.*
SEEK_CUR - Set position to current location plus *offset.*
SEEK_END - Set position to end-of-file plus *offset.*

If *whence is not specified, it is assumed to be SEEK_SET.*

Upon success, returns 0; otherwise, returns -1. Note that seeking past EOF is not considered an error.

May not be used on file pointers returned by **fopen()** if they use the "http://" or "ftp://" formats.

Note: The *whence* argument was added after PHP 4.0 RC1.

See also **ftell()** and **rewind()**.

ftell (PHP3 , PHP4)

Tell file pointer read/write position

```
int ftell (int fp)
```

Returns the position of the file pointer referenced by fp; 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 **fopen()** or **popen()**.

See also **fopen()**, **popen()**, **fseek()** and **rewind()**.

ftruncate (PHP4 >= 4.0RC1)

Truncate a file to a given length.

```
int ftruncate (int fp, int size)
```

Takes the filepointer, fp, and truncates the file to length, size. This function returns true on success and false on failure.

fwrite (PHP3 , PHP4)

Binary-safe file write

```
int fwrite (int fp, string string [, int length])
```

fwrite() writes the contents of *string* to the file stream pointed to by *fp*. If the *length* argument is given, writing will stop after *length* 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 **fread()**, **fopen()**, **fsockopen()**, **popen()**, and **fputs()**.

set_file_buffer (PHP3 >= 3.0.8)

Sets file buffering on the given file pointer

```
int fwrite (int fp, int buffer)
```

set_file_buffer() sets the buffering for write operations on the given filepointer *fp* to *buffer* bytes. If *buffer* is 0 then write operations are unbuffered.

The function returns 0 on success, or EOF if the request cannot be honored.

Note that the default for any fopen with calling set_file_buffer is 8K.

See also **fopen()**.

is_dir (PHP3 , PHP4)

Tells whether the filename is a directory

```
bool is_dir (string filename)
```

Returns true if the filename exists and is a directory.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_file()** and **is_link()**.

is_executable (PHP3 , PHP4)

Tells whether the filename is executable

```
bool is_executable (string filename)
```

Returns true if the filename exists and is executable.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_file()** and **is_link()**.

is_file (PHP3 , PHP4)

Tells whether the filename is a regular file

```
bool is_file (string filename)
```

Returns true if the filename exists and is a regular file.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_dir()** and **is_link()**.

is_link (PHP3 , PHP4)

Tells whether the filename is a symbolic link

```
bool is_link (string filename)
```

Returns true if the filename exists and is a symbolic link.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_dir()** and **is_file()**.

> **Note:** This function does not work on Windows systems

is_readable (PHP3 , PHP4)

Tells whether the filename is readable

```
bool is_readable (string filename)
```

Returns true if the filename exists and is readable.

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_writeable()**.

is_writeable (PHP3 , PHP4)

Tells whether the filename is writeable

```
bool is_writeable (string filename)
```

Returns true if the filename exists and is writeable. The filename argument may be a directory name allowing you to check if a directory is writeable.

Keep in mind that PHP may be accessing the file as the user id that the web server runs as (often 'nobody'). Safe mode limitations are not taken into account.

The results of this function are cached. See **clearstatcache()** for more details.

See also **is_readable()**.

link (PHP3 , PHP4)

Create a hard link

```
int link (string target, string link)
```

Link() creates a hard link.

See also the **symlink()** to create soft links, and **readlink()** along with **linkinfo()**.

> **Note:** This function does not work on Windows systems

linkinfo (PHP3 , PHP4)

Get information about a link

```
int linkinfo (string path)
```

Linkinfo() returns the st_dev field of the UNIX C stat structure returned by the lstat system call. This function is used to verify if a link (pointed to by *path*) really exists (using the same method as the S_ISLNK macro defined in stat.h). Returns 0 or FALSE in case of error.

See also **symlink()**, **link()**, and **readlink()**.

> **Note:** This function does not work on Windows systems

mkdir (PHP3 , PHP4)

Make directory

```
int mkdir (string pathname, int mode)
```

Attempts to create the directory specified by pathname.

Note that you probably want to specify the mode as an octal number, which means it should have a leading zero.

```
mkdir ("/path/to/my/dir", 0700);
```

Returns true on success and false on failure.

See also **rmdir()**.

pclose (PHP3 , PHP4)

Close process file pointer

```
int pclose (int fp)
```

Closes a file pointer to a pipe opened by **popen()**.

The file pointer must be valid, and must have been returned by a successful call to **popen()**.

Returns the termination status of the process that was run.

See also **popen()**.

popen (PHP3 , PHP4)

Open process file pointer

```
int popen (string command, string mode)
```

Opens a pipe to a process executed by forking the command given by command.

Returns a file pointer identical to that returned by **fopen()**, except that it is unidirectional (may only be used for reading or writing) and must be closed with **pclose()**. This pointer may be used with **fgets()**, **fgetss()**, and **fputs()**.

If an error occurs, returns false.

```
$fp = popen ("/bin/ls", "r");
```

See also **pclose()**.

readfile (PHP3 , PHP4)

Output a file

```
int readfile (string filename [, int use_include_path])
```

Reads a file and writes it to standard output.

Returns the number of bytes read from the file. If an error occurs, false is returned and unless the function was called as @readfile, an error message is printed.

If *filename* begins with "http://" (not case sensitive), an HTTP 1.0 connection is opened to the specified server and the text of the response is written to standard output.

Does not handle HTTP redirects, so you must include trailing slashes on directories.

If *filename* begins with "ftp://" (not case sensitive), an ftp connection to the specified server is opened and the requested file is written to standard output. If the server does not support passive mode ftp, this will fail.

If *filename* begins with neither of these strings, the file 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 **fpassthru()**, **file()**, **fopen()**, **include()**, **require()**, and **virtual()**.

readlink (PHP3 , PHP4)

Return the target of a symbolic link

```
string readlink (string path)
```

Readlink() does the same as the readlink C function and returns the contents of the symbolic link path or 0 in case of error.

See also **symlink()**, **readlink()** and **linkinfo()**.

Note: This function does not work on Windows systems

rename (PHP3 , PHP4)

Rename a file

```
int rename (string oldname, string newname)
```

Attempts to rename *oldname* to *newname*.

Returns true on success and false on failure.

rewind (PHP3 , PHP4)

Rewind the position of a file pointer

```
int rewind (int fp)
```

Sets the file position indicator for fp 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 **fopen()**.

See also **fseek()** and **ftell()**.

rmdir (PHP3 , PHP4)

Remove directory

```
int rmdir (string dirname)
```

Attempts to remove the directory named by pathname. The directory must be empty, and the relevant permissions must permit. this.

If an error occurs, returns 0.

See also **mkdir()**.

stat (PHP3 , PHP4)

Give information about a file

```
array stat (string filename)
```

Gathers the statistics of the file named by filename.

Returns an array with the statistics of the file with the following elements:

1. device
2. inode
3. inode protection mode
4. number of links
5. user id of owner
6. group id owner
7. device type if inode device *
8. size in bytes
9. time of last access
10. time of last modification
11. time of last change
12. blocksize for filesystem I/O *
13. number of blocks allocated

* - only valid on systems supporting the st_blksize type–other systems (i.e. Windows) return -1

The results of this function are cached. See **clearstatcache()** for more details.

lstat (PHP3 >= 3.0.4, PHP4)

Give information about a file or symbolic link

```
array lstat (string filename)
```

Gathers the statistics of the file or symbolic link named by filename. This function is identical to the **stat()** function except that if the *filename* parameter is a symbolic link, the status of the symbolic link is returned, not the status of the file pointed to by the symbolic link.

Returns an array with the statistics of the file with the following elements:

1. device
2. inode
3. number of links
4. user id of owner
5. group id owner
6. device type if inode device *
7. size in bytes
8. time of last access
9. time of last modification
10. time of last change
11. blocksize for filesystem I/O *
12. number of blocks allocated

* - only valid on systems supporting the st_blksize type–other systems (i.e. Windows) return -1

The results of this function are cached. See **clearstatcache()** for more details.

symlink (PHP3 , PHP4)

Create a symbolic link

```
int symlink (string target, string link)
```

symlink() creates a symbolic link from the existing *target* with the specified name *link*.

See also **link()** to create hard links, and **readlink()** along with **linkinfo()**.

> **Note:** This function does not work on Windows systems

tempnam (PHP3 , PHP4)

create unique file name

```
string tempnam (string dir, string prefix)
```

Creates a unique temporary filename in the specified directory. If the directory does not exist, **tempnam()** may generate a filename in the system's temporary directory.

The behaviour of the **tempnam()** function is system dependent. On Windows the TMP environment variable will override the *dir* parameter, on Linux the TMPDIR environment variable has precedence, while SVR4 will always use your *dir* parameter if the directory it points to exists. Consult your system documentation on the tempnam(3) function if in doubt.

Returns the new temporary filename, or the null string on failure.

Example 1. Tempnam() example

```
$tmpfname = tempnam ("/tmp", "FOO");
```

touch (PHP3 , PHP4)

Set modification time of file

```
int touch (string filename, int time)
```

Attempts to set the modification time of the file named by filename to the value given by time. If the option time is not given, uses the present time.

If the file does not exist, it is created.

Returns true on success and false otherwise.

Example 1. touch() example

```
if ( touch($FileName) ) {
  print "$FileName modification time has been changed to todays date and time";
} else {
  print "Sorry Could Not change modification time of $FileName";
}
```

umask (PHP3 , PHP4)

Changes the current umask

```
int umask (int mask)
```

Umask() sets PHP's umask to mask & 0777 and returns the old umask. When PHP is being used as a server module, the umask is restored when each request is finished.

Umask() without arguments simply returns the current umask.

unlink (PHP3 , PHP4)

Delete a file

```
int unlink (string filename)
```

Deletes *filename*. Similar to the Unix C unlink() function.

Returns 0 or FALSE on an error.

See also **rmdir()** for removing directories.

> **Note:** This function may not work on Windows systems.

XXI. Forms Data Format functions

Forms Data Format (FDF) is a format for handling forms within PDF documents. You should read the documentation at http://partners.adobe.com/asn/developer/acrosdk/forms.html for more information on what FDF is and how it is used in general.

> **Note:** Currently Adobe only provides a libc5 compatible version for Linux. Tests with glibc2 resulted in a segmentation fault. If somebody is able to make it work, please comment on this page.

> **Note:** If you run into problems configuring php with fdftk support, check whether the header file FdfTk.h and the library libFdfTk.so are at the right place. They should be in fdftk-dir/include and fdftk-dir/lib. This will not be the case if you just unpack the FdfTk distribution.

The general idea of FDF is similar to HTML forms. The diffence is basically the format how filled in data is transmitted to the server when the submit button is pressed (this is actually the Form Data Format) and the format of the form itself (which is the Portable Document Format, PDF). Processing the FDF data is one of the features provided by the fdf functions. But there is more. One may as well take an existing PDF form and populated the input fields with data without modifying the form itself. In such a case one would create a FDF document (**fdf_create()**) set the values of each input field (**fdf_set_value()**) and associate it with a PDF form (**fdf_set_file()**). Finally it has to be sent to the browser with MimeType application/vnd.fdf. The Acrobat reader plugin of your browser recognizes the MimeType, reads the associated PDF form and fills in the data from the FDF document.

The following examples shows just the evaluation of form data.

Example 1. Evaluating a FDF document

```
<?php
// Save the FDF data into a temp file
$fdffp = fopen("test.fdf", "w");
fwrite($fdffp, $HTTP_FDF_DATA, strlen($HTTP_FDF_DATA));
fclose($fdffp);

// Open temp file and evaluate data
// The pdf form contained several input text fields with the names
// volume, date, comment, publisher, preparer, and two checkboxes
// show_publisher and show_preparer.
$fdf = fdf_open("test.fdf");
$volume = fdf_get_value($fdf, "volume");
echo "The volume field has the value '<B>$volume</B>'<BR>";

$date = fdf_get_value($fdf, "date");
echo "The date field has the value '<B>$date</B>'<BR>";

$comment = fdf_get_value($fdf, "comment");
echo "The comment field has the value '<B>$comment</B>'<BR>";

if(fdf_get_value($fdf, "show_publisher") == "On") {
  $publisher = fdf_get_value($fdf, "publisher");
  echo "The publisher field has the value '<B>$publisher</B>'<BR>";
} else
  echo "Publisher shall not be shown.<BR>";

if(fdf_get_value($fdf, "show_preparer") == "On") {
  $preparer = fdf_get_value($fdf, "preparer");
```

```
  echo "The preparer field has the value '<B>$preparer</B>'<BR>";
} else
  echo "Preparer shall not be shown.<BR>";
fdf_close($fdf);
?>
```

fdf_open (PHP3 >= 3.0.6, PHP4)

Open a FDF document

```
int fdf_open (string filename)
```

The **fdf_open()** function opens a file with form data. This file must contain the data as returned from a PDF form. Currently, the file has to be created 'manually' by using **fopen()** and writing the content of HTTP_FDF_DATA with **fwrite()** into it. A mechanism like for HTML form data where for each input field a variable is created does not exist.

Example 1. Accessing the form data

```
<?php
// Save the FDF data into a temp file
$fdffp = fopen("test.fdf", "w");
fwrite($fdffp, $HTTP_FDF_DATA, strlen($HTTP_FDF_DATA));
fclose($fdffp);

// Open temp file and evaluate data
$fdf = fdf_open("test.fdf");
...
fdf_close($fdf);
?>
```

See also **fdf_close()**.

fdf_close (PHP3 >= 3.0.6, PHP4)

Close an FDF document

```
void fdf_close (int fdf_document)
```

The **fdf_close()** function closes the FDF document.

See also **fdf_open()**.

fdf_create (PHP3 >= 3.0.6, PHP4)

Create a new FDF document

```
int fdf_create (void )
```

The **fdf_create()** creates a new FDF document. This function is needed if one would like to populate input fields in a PDF document with data.

Example 1. Populating a PDF document

```
<?php
$outfdf = fdf_create();
fdf_set_value($outfdf, "volume", $volume, 0);

fdf_set_file($outfdf, "http:/testfdf/resultlabel.pdf");
fdf_save($outfdf, "outtest.fdf");
fdf_close($outfdf);
Header("Content-type: application/vnd.fdf");
$fp = fopen("outtest.fdf", "r");
fpassthru($fp);
unlink("outtest.fdf");
?>
```

See also **fdf_close()**, **fdf_save()**, **fdf_open()**.

fdf_save (PHP3 >= 3.0.6, PHP4)

Save a FDF document

```
int fdf_save (string filename)
```

The **fdf_save()** function saves a FDF document. The FDF Toolkit provides a way to output the document to stdout if the parameter *filename* is '.'. This does not work if PHP is used as an apache module. In such a case one will have to write to a file and use e.g. **fpassthru()**. to output it.

See also **fdf_close()** and example for **fdf_create()**.

fdf_get_value (PHP3 >= 3.0.6, PHP4)

Get the value of a field

```
string fdf_get_value (int fdf_document, string fieldname)
```

The **fdf_get_value()** function returns the value of a field.

See also **fdf_set_value()**.

fdf_set_value (PHP3 >= 3.0.6, PHP4)

Set the value of a field

```
void fdf_set_value (int fdf_document, string fieldname, string value, int
isName)
```

The **fdf_set_value()** function sets the value of a field. The last parameter determines if the field value is to be converted to a PDF Name (*isName* = 1) or set to a PDF String (*isName* = 0).

See also **fdf_get_value()**.

fdf_next_field_name (PHP3 >= 3.0.6, PHP4)

Get the next field name

```
string fdf_next_field_name (int fdf_document, string fieldname)
```

The **fdf_next_field_name()** function returns the name of the field after the field in *fieldname* or the field name of the first field if the second paramter is NULL.

See also **fdf_set_field()**, **fdf_get_field()**.

fdf_set_ap (PHP3 >= 3.0.6, PHP4)

Set the appearance of a field

```
void fdf_set_ap (int fdf_document, string field_name, int face, string filename,
int page_number)
```

The **fdf_set_ap()** function sets the appearance of a field (i.e. the value of the /AP key). The possible values of *face* are 1=FDFNormalAP, 2=FDFRolloverAP, 3=FDFDownAP.

fdf_set_status (PHP3 >= 3.0.6, PHP4)

Set the value of the /STATUS key

```
void fdf_set_status (int fdf_document, string status)
```

The **fdf_set_status()** sets the value of the /STATUS key.

See also **fdf_get_status()**.

fdf_get_status (PHP3 >= 3.0.6, PHP4)

Get the value of the /STATUS key

```
string fdf_get_status (int fdf_document)
```

The **fdf_get_status()** returns the value of the /STATUS key.

See also **fdf_set_status()**.

fdf_set_file (PHP3 >= 3.0.6, PHP4)

Set the value of the /F key

```
void fdf_set_file (int fdf_document, string filename)
```

The **fdf_set_file()** sets the value of the /F key. The /F key is just a reference to a PDF form which is to be populated with data. In a web environment it is a URL (e.g. http:/testfdf/resultlabel.pdf).

See also **fdf_get_file()** and example for **fdf_create()**.

fdf_get_file (PHP3 >= 3.0.6, PHP4)

Get the value of the /F key

```
string fdf_get_file (int fdf_document)
```

The **fdf_set_file()** returns the value of the /F key.

See also **fdf_set_file()**.

XXII. FTP functions

FTP stands for File Transfer Protocol.

The following constants are defined when using the FTP module: FTP_ASCII, and FTP_BINARY.

ftp_connect (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Opens up an FTP connection

```
int ftp_connect (string host [, int port])
```

Returns a FTP stream on success, false on error.

ftp_connect() opens up a FTP connection to the specified *host*. The *port* parameter specifies an alternate port to connect to. If it is omitted or zero, then the default FTP port, 21, will be used.

ftp_login (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Logs in an FTP connection

```
int ftp_login (int ftp_stream, string username, string password)
```

Returns true on success, false on error.

Logs in the given FTP stream.

ftp_pwd (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the current directory name

```
int ftp_pwd (int ftp_stream)
```

Returns the current directory, or false on error.

ftp_cdup (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Changes to the parent directory

```
int ftp_cdup (int ftp_stream)
```

Returns true on success, false on error.

Changes to the parent directory.

ftp_chdir (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Changes directories on a FTP server

```
int ftp_chdir (int ftp_stream, string directory)
```

Returns true on success, false on error.

Changes to the specified *directory*.

ftp_mkdir (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Creates a directory

```
string ftp_mkdir (int ftp_stream, string directory)
```

Returns the newly created directory name on success, false on error.

Creates the specified *directory*.

ftp_rmdir (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Removes a directory

```
int ftp_rmdir (int ftp_stream, string directory)
```

Returns true on success, false on error.

Removes the specified *directory*.

ftp_nlist (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns a list of files in the given directory.

```
int ftp_nlist (int ftp_stream, string directory)
```

Returns an array of filenames on success, false on error.

ftp_rawlist (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns a detailed list of files in the given directory.

```
int ftp_rawlist (int ftp_stream, string directory)
```

ftp_rawlist() executes the FTP LIST command, and returns the result as an array. Each array element corresponds to one line of text. The output is not parsed in any way. The system type identifier returned by **ftp_systype()** can be used to determine how the results should be interpreted.

ftp_systype (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the system type identifier of the remote FTP server.

```
int ftp_systype (int ftp_stream)
```

Returns the remote system type, or false on error.

ftp_pasv (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Turns passive mode on or off.

```
int ftp_pasv (int ftp_stream, int pasv)
```

Returns true on success, false on error.

ftp_pasv() turns on passive mode if the *pasv* parameter is true (it turns off passive mode if *pasv* is false.) In passive mode, data connections are initiated by the client, rather than by the server.

ftp_get (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Downloads a file from the FTP server.

```
int ftp_get (int ftp_stream, string local_file, string remote_file, int mode)
```

Returns true on success, false on error.

ftp_get() retrieves *remote_file* from the FTP server, and saves it to *local_file* locally. The transfer *mode* specified must be either FTP_ASCII or FTP_BINARY.

ftp_fget (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Downloads a file from the FTP server and saves to an open file.

```
int ftp_fget (int ftp_stream, int fp, string remote_file, int mode)
```

Returns true on success, false on error.

ftp_fget() retrieves *remote_file* from the FTP server, and writes it to the given file pointer, *fp*. The transfer *mode* specified must be either FTP_ASCII or FTP_BINARY.

ftp_put (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Uploads a file to the FTP server.

```
int ftp_put (int ftp_stream, string remote_file, string local_file, int mode)
```

Returns true on success, false on error.

ftp_put() stores *local_file* on the FTP server, as *remote_file*. The transfer *mode* specified must be either FTP_ASCII or FTP_BINARY.

ftp_fput (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Uploads from an open file to the FTP server.

```
int ftp_fput (int ftp_stream, string remote_file, int fp, int mode)
```

Returns true on success, false on error.

ftp_fput() uploads the data from the file pointer *fp* until end of file. The results are stored in *remote_file* on the FTP server. The transfer *mode* specified must be either FTP_ASCII or FTP_BINARY.

ftp_size (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the size of the given file.

```
int ftp_size (int ftp_stream, string remote_file)
```

Returns the file size on success, or -1 on error.

ftp_size() returns the size of a file. If an error occurs, of if the file does not exist, -1 is returned. Not all servers support this feature.

ftp_mdtm (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the last modified time of the given file.

```
int ftp_mdtm (int ftp_stream, string remote_file)
```

Returns a UNIX timestamp on success, or -1 on error.

ftp_mdtm() checks the last-modified time for a file, and returns it as a UNIX timestamp. If an error occurs, or the file does not exist, -1 is returned. Note that not all servers support this feature.

ftp_rename (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Renames a file on the ftp server.

```
int ftp_rename (int ftp_stream, string from, string to)
```

Returns true on success, false on error.

ftp_rename() renames the file specified by *from* to the new name *to*

ftp_delete (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Deletes a file on the ftp server.

```
int ftp_delete (int ftp_stream, string path)
```

Returns true on success, false on error.

ftp_delete() deletes the file specified by *path* from the FTP server.

ftp_site (PHP3 >= 3.0.15, PHP4 >= 4.0RC1)

Sends a SITE command to the server.

```
int ftp_site (int ftp_stream, string cmd)
```

Returns true on success, false on error.

ftp_site() sends the command specified by *cmd* to the FTP server. SITE commands are not standardized, and vary from server to server. They are useful for handling such things as file permissions and group membership.

ftp_quit (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Closes an FTP connection

```
int ftp_quit (int ftp_stream)
```

ftp_connect() closes *`ftp_stream`*.

XXIII. GNU Gettext

bindtextdomain (PHP3 >= 3.0.7, PHP4)

Sets the path for a domain

```
string bindtextdomain (string domain, string directory)
```

The **bindtextdomain()** function sets the path for a domain.

dcgettext (PHP3 >= 3.0.7, PHP4)

Overrides the domain for a single lookup

```
string dcgettext (string domain, string message, int category)
```

This function allows you to override the current domain for a single message lookup. It also allows you to specify a category.

dgettext (PHP3 >= 3.0.7, PHP4)

Override the current domain

```
string dgettext (string domain, string message)
```

The **dgettext()** function allows you to override the current domain for a single message lookup.

gettext (PHP3 >= 3.0.7, PHP4)

Lookup a message in the current domain

```
string gettext (string message)
```

This function returns a translated string if one is found in the translation table, or the submitted message if not found. You may use an underscore character as an alias to this function.

Example 1. Gettext()-check

```
<?php
// Set language to German
putenv ("LANG=de");

// Specify location of translation tables
bindtextdomain ("myPHPApp", "./locale");
```

```
// Choose domain
textdomain ("myPHPApp");

// Print a test message
print (gettext ("Welcome to My PHP Application"));
?>
```

textdomain (PHP3 >= 3.0.7, PHP4)

Sets the default domain

```
int textdomain ([string library])
```

This function sets the domain to search within when calls are made to **gettext()**, usually the named after an application. The previous default domain is returned. Call it with no parameters to get the current setting without changing it.

XXIV. 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 (PHP3 >= 3.0.9, PHP4)

Get the name of the specified hash

```
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 (PHP3 >= 3.0.9, PHP4)

Get the block size of the specified hash

```
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 (PHP3 >= 3.0.9, PHP4)

Get the highest available hash id

```
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 (PHP3 >= 3.0.9, PHP4)

Compute hash

```
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).

XXV. HTTP functions

These functions let you manipulate the output sent back to the remote browser right down to the HTTP protocol level.

header (PHP3 , PHP4)

Send a raw HTTP header

```
int header (string string)
```

The **Header()** function is used at the top of an HTML file to send raw HTTP header strings. See the HTTP 1.1 Specification (http://www.w3.org/Protocols/rfc2616/rfc2616) for more information on raw http headers. *Note:* Remember that the **Header()** function must be called before any actual output is sent either by normal HTML tags or from PHP. It is a very common error to read code with **include()** or with auto_prepend and have spaces or empty lines in this code that force output before **header()** is called.

There are two special-case header calls. The first is the "Location" header. Not only does it send this header back to the browser, it also returns a REDIRECT status code to Apache. From a script writer's point of view this should not be important, but for people who understand Apache internals it is important to understand.

```
header ("Location: http://www.php.net"); /* Redirect browser
                                            to PHP web site */
exit;                 /* Make sure that code below does
                         not get executed when we redirect. */
```

The second special-case is any header that starts with the string, "HTTP/" (case is not significant). For example, if you have your ErrorDocument 404 Apache directive pointed to a PHP script, it would be a good idea to make sure that your PHP script is actually generating a 404. The first thing you do in your script should then be:

```
header ("http/1.0 404 Not Found");
```

PHP scripts often generate dynamic HTML that must not be cached by the client browser or any proxy caches between the server and the client browser. Many proxies and clients can be forced to disable caching with

```
header ("Expires: Mon, 26 Jul 1997 05:00:00 GMT");    // Date in the past
header ("Last-Modified: " . gmdate("D, d M Y H:i:s") . " GMT");
                                                      // always modified
header ("Cache-Control: no-cache, must-revalidate");  // HTTP/1.1
header ("Pragma: no-cache");                          // HTTP/1.0
```

setcookie (PHP3 , PHP4)

Send a cookie

```
int setcookie (string name, string value, int expire, string path, string
domain, int secure)
```

setcookie() defines a cookie to be sent along with the rest of the header information. Cookies must be sent *before* any other headers are sent (this is a restriction of cookies, not PHP). This requires you to place calls to this function before any `<html>` or `<head>` tags.

All the arguments except the *`name`* argument are optional. If only the name argument is present, the cookie by that name will be deleted from the remote client. You may also replace any argument with an empty string (*""*) in order to skip that argument. The *`expire`* and *`secure`* arguments are integers and cannot be skipped with an empty string. Use a zero (*0*) instead. The *`expire`* argument is a regular Unix time integer as returned by the **time()** or **mktime()** functions. The *`secure`* indicates that the cookie should only be transmitted over a secure HTTPS connection.

Common Pitfalls:

Cookies will not become visible until the next loading of a page that the cookie should be visible for.

Multiple calls to **setcookie()** in the same script will be performed in reverse order. If you are trying to delete one cookie before inserting another you should put the insert before the delete.

Some examples follow:

Example 1. setcookie() examples

```
setcookie ("TestCookie", "Test Value");
setcookie ("TestCookie", $value,time()+3600);  /* expire in 1 hour */
setcookie ("TestCookie", $value,time()+3600, "/~rasmus/", ".utoronto.ca", 1);
```

Note that the value portion of the cookie will automatically be urlencoded when you send the cookie, and when it is received, it is automatically decoded and assigned to a variable by the same name as the cookie name. To see the contents of our test cookie in a script, simply use one of the following examples:

```
echo $TestCookie;
echo $HTTP_COOKIE_VARS["TestCookie"];
```

You may also set array cookies by using array notation in the cookie name. This has the effect of setting as many cookies as you have array elements, but when the cookie is received by your script, the values are all placed in an array with the cookie's name:

```
setcookie ("cookie[three]", "cookiethree");
setcookie ("cookie[two]", "cookietwo");
setcookie ("cookie[one]", "cookieone");
if (isset ($cookie)) {
    while (list ($name, $value) = each ($cookie)) {
        echo "$name == $value<br>\n";
    }
}
```

For more information on cookies, see Netscape's cookie specification at http://www.netscape.com/newsref/std/cookie_spec.html.

Microsoft Internet Explorer 4 with Service Pack 1 applied does not correctly deal with cookies that have their path parameter set.

Netscape Communicator 4.05 and Microsoft Internet Explorer 3.x appear to handle cookies incorrectly when the path and time are not set.

XXVI. Hyperwave functions

Introduction

Hyperwave has been developed at IICM (http://www.iicm.edu) in Graz. It started with the name Hyper-G and changed to Hyperwave when it was commercialised (If I remember properly it was in 1996).

Hyperwave is not free software. The current version, 4.1, is available at www.hyperwave.com (http://www.hyperwave.com/). A time limited version can be ordered for free (30 days).

Hyperwave is an information system similar to a database (HIS, Hyperwave Information Server). Its focus is the storage and management of documents. A document can be any possible piece of data that may as well be stored in file. Each document is accompanied by its object record. The object record contains meta data for the document. The meta data is a list of attributes which can be extended by the user. Certain attributes are always set by the Hyperwave server, other may be modified by the user. An attribute is a name/value pair of the form name=value. The complete object record contains as many of those pairs as the user likes. The name of an attribute does not have to be unique, e.g. a title may appear several times within an object record. This makes sense if you want to specify a title in several languages. In such a case there is a convention, that each title value is preceded by the two letter language abbreviation followed by a colon, e.g. 'en:Title in English' or 'ge:Titel in deutsch'. Other attributes like a description or keywords are potential candidates. You may also replace the language abbreviation by any other string as long as it separated by colon from the rest of the attribute value.

Each object record has native a string representation with each name/value pair separated by a newline. The Hyperwave extension also knows a second representation which is an associated array with the attribute name being the key. Multilingual attribute values itself form another associated array with the key being the language abbreviation. Actually any multiple attribute forms an associated array with the string left to the colon in the attribute value being the key. (This is not fully implemented. Only the attributes Title, Description and Keyword are treated properly yet.)

Besides the documents, all hyper links contained in a document are stored as object records as well. Hyper links which are in a document will be removed from it and stored as individual objects, when the document is inserted into the database. The object record of the link contains information about where it starts and where it ends. In order to gain the original document you will have to retrieve the plain document without the links and the list of links and reinsert them (The functions **hw_pipedocument()** and **hw_gettext()** do this for you. The advantage of separating links from the document is obvious. Once a document to which a link is pointing to changes its name, the link can easily be modified accordingly. The document containing the link is not affected at all. You may even add a link to a document without modifying the document itself.

Saying that **hw_pipedocument()** and **hw_gettext()** do the link insertion automatically is not as simple as it sounds. Inserting links implies a certain hierarchy of the documents. On a web server this is given by the file system, but Hyperwave has its own hierarchy and names do not reflect the position of an object in that hierarchy. Therefore creation of links first of all requires a mapping from the Hyperwave hierarchy and namespace into a web hierarchy respective web namespace. The fundamental difference between Hyperwave and the web is the clear distinction between names and hierarchy in Hyperwave. The name does not contain any information about the objects position in the hierarchy. In the web the name also contains the information on where the object is located in the hierarchy. This leads to two possibles ways of mapping. Either the Hyperwave hierarchy and name of the Hyperwave object is reflected in the URL or the name only. To make things simple the second approach is used. Hyperwave object with name 'my_object' is mapped to 'http://host/my_object' disregarding where it resides in the Hyperwave hierarchy. An object with name 'parent/my_object' could be the child of 'my_object' in the Hyperwave hierarchy, though in a web namespace it appears to be just the opposite and the user might get confused. This can only be prevented by selecting reasonable object names.

Having made this decision a second problem arises. How do you involve PHP? The URL http://host/my_object will not call any PHP script unless you tell your web server to rewrite it to e.g. 'http://host/php3_script/my_object' and the script 'php3_script' evaluates the $PATH_INFO variable and retrieves the object with name 'my_object' from the Hyperwave server. Their is just one little drawback which can be fixed easily. Rewriting any URL would not allow any access to other document on the web server. A PHP script for searching in the Hyperwave server would be impossible. Therefore you will need at least a second rewriting rule to exclude certain URLS like all e.g. starting with http://host/Hyperwave. This is basically sharing of a namespace by the web and Hyperwave server.

Based on the above mechanism links are insert into documents.

It gets more complicated if PHP is not run as a server module or CGI script but as a standalone application e.g. to dump the content of the Hyperwave server on a CD-ROM. In such a case it makes sense to retain the Hyperwave hierarchy and map in onto the file system. This conflicts with the object names if they reflect its own hierarchy (e.g. by choosing names including '/'). Therefore '/' has to be replaced by another character, e.g. '_'. to be continued.

The network protocol to communicate with the Hyperwave server is called HG-CSP (http://www.hyperwave.de/7.17-hg-prot) (Hyper-G Client/Server Protocol). It is based on messages to initiate certain actions, e.g. get object record. In early versions of the Hyperwave Server two native clients (Harmony, Amadeus) were provided for communication with the server. Those two disappeared when Hyperwave was commercialised. As a replacement a so called wavemaster was provided. The wavemaster is like a protocol converter from HTTP to HG-CSP. The idea is to do all the administration of the database and visualisation of documents by a web interface. The wavemaster implements a set of placeholders for certain actions to customise the interface. This set of placeholders is called the PLACE Language. PLACE lacks a lot of features of a real programming language and any extension to it only enlarges the list of placeholders. This has led to the use of JavaScript which IMO does not make life easier.

Adding Hyperwave support to PHP should fill in the gap of a missing programming language for interface customisation. It implements all the messages as defined by the HG-CSP but also provides more powerful commands to e.g. retrieve complete documents.

Hyperwave has its own terminology to name certain pieces of information. This has widely been taken over and extended. Almost all functions operate on one of the following data types.

- object ID: An unique integer value for each object in the Hyperwave server. It is also one of the attributes of the object record (ObjectID). Object ids are often used as an input parameter to specify an object.
- object record: A string with attribute-value pairs of the form attribute=value. The pairs are separated by a carriage return from each other. An object record can easily be converted into an object array with **hw_object2array()**. Several functions return object records. The names of those functions end with obj.
- object array: An associated array with all attributes of an object. The key is the attribute name. If an attribute occurs more than once in an object record it will result in another indexed or associated array. Attributes which are language depended (like the title, keyword, description) will form an associated array with the key set to the language abbreviation. All other multiple attributes will form an indexed array. PHP functions never return object arrays.
- hw_document: This is a complete new data type which holds the actual document, e.g. HTML, PDF etc. It is somewhat optimised for HTML documents but may be used for any format.

Several functions which return an array of object records do also return an associated array with statistical information about them. The array is the last element of the object record array. The statistical array contains the following entries:

Hidden

Number of object records with attribute PresentationHints set to Hidden.

CollectionHead

Number of object records with attribute PresentationHints set to CollectionHead.

FullCollectionHead

Number of object records with attribute PresentationHints set to FullCollectionHead.

CollectionHeadNr

Index in array of object records with attribute PresentationHints set to CollectionHead.

FullCollectionHeadNr

Index in array of object records with attribute PresentationHints set to FullCollectionHead.

Total

Total: Number of object records.

Integration with Apache

The Hyperwave extension is best used when PHP is compiled as an Apache module. In such a case the underlying Hyperwave server can be hidden from users almost completely if Apache uses its rewriting engine. The following instructions will explain this.

Since PHP with Hyperwave support built into Apache is intended to replace the native Hyperwave solution based on Wavemaster I will assume that the Apache server will only serve as a Hyperwave web interface. This is not necessary but it simplifies the configuration. The concept is quite simple. First of all you need a PHP script which evaluates the PATH_INFO variable and treats its value as the name of a Hyperwave object. Let's call this script 'Hyperwave'. The URL http://your.hostname/Hyperwave/name_of_object would than return the Hyperwave object with the name 'name_of_object'. Depending on the type of the object the script has to react accordingly. If it is a collection, it will probably return a list of children. If it is a document it will return the mime type and the content. A slight improvement can be achieved if the Apache rewriting engine is used. From the users point of view it would be more straight forward if the URL http://your.hostname/name_of_object would return the object. The rewriting rule is quite easy:

```
RewriteRule ^/(.*) /usr/local/apache/htdocs/HyperWave/$1 [L]
```

Now every URL relates to an object in the Hyperwave server. This causes a simple to solve problem. There is no way to execute a different script, e.g. for searching, than the 'Hyperwave' script. This can be fixed with another rewriting rule like the following:

```
RewriteRule ^/hw/(.*) /usr/local/apache/htdocs/hw/$1 [L]
```

This will reserve the directory `/usr/local/apache/htdocs/hw` for additional scripts and other files. Just make sure this rule is evaluated before the one above. There is just a little drawback: all Hyperwave objects whose name starts with 'hw/' will be shadowed. So, make sure you don't use such names. If you need more

directories, e.g. for images just add more rules or place them all in one directory. Finally, don't forget to turn on the rewriting engine with

```
RewriteEngine on
```

My experiences have shown that you will need the following scripts:

- to return the object itself
- to allow searching
- to identify yourself
- to set your profile
- one for each additional function like to show the object attributes, to show information about users, to show the status of the server, etc.

Todo

There are still some things todo:

- The hw_InsertDocument has to be split into **hw_InsertObject()** and **hw_PutDocument()**.
- The names of several functions are not fixed, yet.
- Most functions require the current connection as its first parameter. This leads to a lot of typing, which is quite often not necessary if there is just one open connection. A default connection will improve this.
- Conversion form object record into object array needs to handle any multiple attribute.

hw_Array2Objrec (unknown)

convert attributes from object array to object record

```
strin hw_array2objrec (array object_array)
```

Converts an *object_array* into an object record. Multiple attributes like 'Title' in different languages are treated properly.

See also **hw_objrec2array()**.

hw_Children (unknown)

object ids of children

```
array hw_children (int connection, int objectID)
```

Returns an array of object ids. Each id belongs to a child of the collection with ID *objectID*. The array contains all children both documents and collections.

hw_ChildrenObj (unknown)

object records of children

```
array hw_childrenobj (int connection, int objectID)
```

Returns an array of object records. Each object record belongs to a child of the collection with ID *objectID*. The array contains all children both documents and collections.

hw_Close (unknown)

closes the Hyperwave connection

```
int hw_close (int connection)
```

Returns false if connection is not a valid connection index, otherwise true. Closes down the connection to a Hyperwave server with the given connection index.

hw_Connect (unknown)

opens a connection

```
int hw_connect (string host, int port, string username, string password)
```

Opens a connection to a Hyperwave server and returns a connection index on success, or false if the connection could not be made. Each of the arguments should be a quoted string, except for the port number. The *username* and *password* arguments are optional and can be left out. In such a case no identification with the server will be done. It is similar to identify as user anonymous. This function returns a connection index that is needed by other Hyperwave functions. You can have multiple connections open at once. Keep in mind, that the password is not encrypted.

See also **hw_pConnect()**.

hw_Cp (unknown)

copies objects

```
int hw_cp (int connection, array object_id_array, int destination id)
```

Copies the objects with object ids as specified in the second parameter to the collection with the id *destination id*.

The value return is the number of copied objects.

See also **hw_mv()**.

hw_Deleteobject (unknown)

deletes object

```
int hw_deleteobject (int connection, int object_to_delete)
```

Deletes the object with the given object id in the second parameter. It will delete all instances of the object.

Returns TRUE if no error occurs otherwise FALSE.

See also **hw_mv()**.

hw_DocByAnchor (unknown)

object id object belonging to anchor

```
int hw_docbyanchor (int connection, int anchorID)
```

Returns an th object id of the document to which *anchorID* belongs.

hw_DocByAnchorObj (unknown)

object record object belonging to anchor

```
string hw_docbyanchorobj (int connection, int anchorID)
```

Returns an th object record of the document to which *anchorID* belongs.

hw_DocumentAttributes (unknown)

object record of hw_document

```
string hw_documentattributes (int hw_document)
```

Returns the object record of the document.

See also **hw_DocumentBodyTag()**, **hw_DocumentSize()**.

hw_DocumentBodyTag (unknown)

body tag of hw_document

```
string hw_documentbodytag (int hw_document)
```

Returns the BODY tag of the document. If the document is an HTML document the BODY tag should be printed before the document.

See also **hw_DocumentAttributes()**, **hw_DocumentSize()**.

hw_DocumentContent (unknown)

returns content of hw_document

```
string hw_documentcontent (int hw_document)
```

Returns the content of the document. If the document is an HTML document the content is everything after the BODY tag. Information from the HEAD and BODY tag is in the stored in the object record.

See also **hw_DocumentAttributes()**, **hw_DocumentSize()**, **hw_DocumentSetContent()**.

hw_DocumentSetContent (unknown)

sets/replaces content of hw_document

```
string hw_documentsetcontent (int hw_document, string content)
```

Sets or replaces the content of the document. If the document is an HTML document the content is everything after the BODY tag. Information from the HEAD and BODY tag is in the stored in the object record. If you provide this information in the content of the document too, the Hyperwave server will change the object record accordingly when the document is inserted. Probably not a very good idea. If this functions fails the document will retain its old content.

See also **hw_DocumentAttributes()**, **hw_DocumentSize()**, **hw_DocumentContent()**.

hw_DocumentSize (unknown)

size of hw_document

```
int hw_documentsize (int hw_document)
```

Returns the size in bytes of the document.

See also **hw_DocumentBodyTag()**, **hw_DocumentAttributes()**.

hw_ErrorMsg (unknown)

returns error message

```
string hw_errormsg (int connection)
```

Returns a string containing the last error message or 'No Error'. If false is returned, this function failed. The message relates to the last command.

hw_EditText (unknown)

retrieve text document

```
int hw_edittext (int connection, int hw_document)
```

Uploads the text document to the server. The object record of the document may not be modified while the document is edited. This function will only works for pure text documents. It will not open a special data connection and therefore blocks the control connection during the transfer.

See also **hw_PipeDocument()**, **hw_FreeDocument()**, **hw_DocumentBodyTag()**, **hw_DocumentSize()**, **hw_OutputDocument()**, **hw_GetText()**.

hw_Error (unknown)

error number

```
int hw_error (int connection)
```

Returns the last error number. If the return value is 0 no error has occurred. The error relates to the last command.

hw_Free_Document (unknown)

frees hw_document

```
int hw_free_document (int hw_document)
```

Frees the memory occupied by the Hyperwave document.

hw_GetParents (unknown)

object ids of parents

```
array hw_getparentsobj (int connection, int objectID)
```

Returns an indexed array of object ids. Each object id belongs to a parent of the object with ID *objectID*.

hw_GetParentsObj (unknown)

object records of parents

```
array hw_getparentsobj (int connection, int objectID)
```

Returns an indexed array of object records plus an associated array with statistical information about the object records. The associated array is the last entry of the returned array. Each object record belongs to a parent of the object with ID *objectID*.

hw_GetChildColl (unknown)

object ids of child collections

```
array hw_getchildcoll (int connection, int objectID)
```

Returns an array of object ids. Each object ID belongs to a child collection of the collection with ID *objectID*. The function will not return child documents.

See also **hw_GetChildren()**, **hw_GetChildDocColl()**.

hw_GetChildCollObj (unknown)

object records of child collections

```
array hw_getchildcollobj (int connection, int objectID)
```

Returns an array of object records. Each object records belongs to a child collection of the collection with ID *objectID*. The function will not return child documents.

See also **hw_ChildrenObj()**, **hw_GetChildDocCollObj()**.

hw_GetRemote (unknown)

Gets a remote document

```
int hw_getremote (int connection, int objectID)
```

Returns a remote document. Remote documents in Hyperwave notation are documents retrieved from an external source. Common remote documents are for example external web pages or queries in a database. In order to be able to access external sources throught remote documents Hyperwave introduces the HGI (Hyperwave Gateway Interface) which is similar to the CGI. Currently, only ftp, http-servers and some databases can be accessed by the HGI. Calling **hw_GetRemote()** returns the document from the external source. If you want to use this function you should be very familiar with HGIs. You should also consider to use PHP instead of Hyperwave to access external sources. Adding database support by a Hyperwave gateway should be more difficult than doing it in PHP.

See also **hw_GetRemoteChildren()**.

hw_GetRemoteChildren (unknown)

Gets children of remote document

```
int hw_getremotechildren (int connection, string object record)
```

Returns the children of a remote document. Children of a remote document are remote documents itself. This makes sense if a database query has to be narrowed and is explained in Hyperwave Programmers' Guide. If the number of children is 1 the function will return the document itself formated by the Hyperwave Gateway Interface (HGI). If the number of children is greater than 1 it will return an array of object record with each maybe the input value for another call to **hw_GetRemoteChildren()**. Those object records are virtual and do not exist in the Hyperwave server, therefore they do not have a valid object ID. How exactely

such an object record looks like is up to the HGI. If you want to use this function you should be very familiar with HGIs. You should also consider to use PHP instead of Hyperwave to access external sources. Adding database support by a Hyperwave gateway should be more difficult than doing it in PHP.

See also **hw_GetRemote()**.

hw_GetSrcByDestObj (unknown)

Returns anchors pointing at object

```
array hw_getsrcbydestobj (int connection, int objectID)
```

Returns the object records of all anchors pointing to the object with ID *objectID*. The object can either be a document or an anchor of type destination.

See also **hw_GetAnchors()**.

hw_GetObject (unknown)

object record

```
array hw_getobject (int connection, [int|array] objectID, string query)
```

Returns the object record for the object with ID *objectID* if the second parameter is an integer. If the second parameter is an array of integer the function will return an array of object records. In such a case the last parameter is also evaluated which is a query string.

The query string has the following syntax:

<expr> ::= "(" <expr> ")" |

"!" <expr> | /* NOT */

<expr> "||" <expr> | /* OR */

<expr> "&&" <expr> | /* AND */

<attribute> <operator> <value>

<attribute> ::= /* any attribute name (Title, Author, DocumentType ...) */

<operator> ::= "=" | /* equal */

"<" | /* less than (string compare) */

">" | /* greater than (string compare) */

"~" /* regular expression matching */

The query allows to further select certain objects from the list of given objects. Unlike the other query functions, this query may use not indexed attributes. How many object records are returned depends on the query and if access to the object is allowed.

See also **hw_GetAndLock()**, **hw_GetObjectByQuery()**.

hw_GetAndLock (unknown)

return bject record and lock object

```
string hw_getandlock (int connection, int objectID)
```

Returns the object record for the object with ID *objectID*. It will also lock the object, so other users cannot access it until it is unlocked.

See also **hw_Unlock()**, **hw_GetObject()**.

hw_GetText (unknown)

retrieve text document

```
int hw_gettext (int connection, int objectID [, mixed rootID/prefix])
```

Returns the document with object ID *objectID*. If the document has anchors which can be inserted, they will be inserted already. The optional parameter *rootID/prefix* can be a string or an integer. If it is an integer it determines how links are inserted into the document. The default is 0 and will result in links that are constructed from the name of the link's destination object. This is useful for web applications. If a link points to an object with name 'internet_movie' the HTML link will be <A HREF="/internet_movie">. The actual location of the source and destination object in the document hierachy is disregarded. You will have to set up your web browser, to rewrite that URL to for example '/my_script.php3/internet_movie'. 'my_script.php3' will have to evaluate $PATH_INFO and retrieve the document. All links will have the prefix '/my_script.php3/'. If you do not want this you can set the optional parameter *rootID/prefix* to any prefix which is used instead. Is this case it has to be a string.

If *rootID/prefix* is an integer and unequal to 0 the link is constructed from all the names starting at the object with the id *rootID/prefix* separated by a slash relative to the current object. If for example the above document 'internet_movie' is located at 'a-b-c-internet_movie' with '-' being the seperator between hierachy levels on the Hyperwave server and the source document is located at 'a-b-d-source' the resulting HTML link would be: <A HREF="../c/internet_movie">. This is useful if you want to download the whole server content onto disk and map the document hierachy onto the file system.

This function will only work for pure text documents. It will not open a special data connection and therefore blocks the control connection during the transfer.

See also **hw_PipeDocument()**, **hw_FreeDocument()**, **hw_DocumentBodyTag()**, **hw_DocumentSize()**, **hw_OutputDocument()**.

hw_GetObjectByQuery (unknown)

search object

```
array hw_getobjectbyquery (int connection, string query, int max_hits)
```

Searches for objects on the whole server and returns an array of object ids. The maximum number of matches is limited to *max_hits*. If *max_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also **hw_GetObjectByQueryObj()**.

hw_GetObjectByQueryObj (unknown)

search object

```
array hw_getobjectbyqueryobj (int connection, string query, int max_hits)
```

Searches for objects on the whole server and returns an array of object records. The maximum number of matches is limited to *max_hits*. If *max_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also **hw_GetObjectByQuery()**.

hw_GetObjectByQueryColl (unknown)

search object in collection

```
array hw_getobjectbyquerycoll (int connection, int objectID, string query, int
max_hits)
```

Searches for objects in collection with ID *objectID* and returns an array of object ids. The maximum number of matches is limited to *max_hits*. If *max_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also **hw_GetObjectByQueryCollObj()**.

hw_GetObjectByQueryCollObj (unknown)

search object in collection

```
array hw_getobjectbyquerycollobj (int connection, int objectID, string query,
int max_hits)
```

Searches for objects in collection with ID *objectID* and returns an array of object records. The maximum number of matches is limited to *max_hits*. If *max_hits* is set to -1 the maximum number of matches is unlimited.

The query will only work with indexed attributes.

See also **hw_GetObjectByQueryColl()**.

hw_GetChildDocColl (unknown)

object ids of child documents of collection

```
array hw_getchilddoccoll (int connection, int objectID)
```

Returns array of object ids for child documents of a collection.
See also **hw_GetChildren()**, **hw_GetChildColl()**.

hw_GetChildDocCollObj (unknown)

object records of child documents of collection

```
array hw_getchilddoccollobj (int connection, int objectID)
```

Returns an array of object records for child documents of a collection.
See also **hw_ChildrenObj()**, **hw_GetChildCollObj()**.

hw_GetAnchors (unknown)

object ids of anchors of document

```
array hw_getanchors (int connection, int objectID)
```

Returns an array of object ids with anchors of the document with object ID *objectID*.

hw_GetAnchorsObj (unknown)

object records of anchors of document

```
array hw_getanchorsobj (int connection, int objectID)
```

Returns an array of object records with anchors of the document with object ID *objectID*.

hw_Mv (unknown)

moves objects

```
int hw_mv (int connection, array object id array, int source id, int destination id)
```

Moves the objects with object ids as specified in the second parameter from the collection with id *source id* to the collection with the id *destination id*. If the destination id is 0 the objects will be unlinked from the source collection. If this is the last instance of that object it will be deleted. If you want to delete all instances at once, use **hw_deleteobject()**.

The value return is the number of moved objects.

See also **hw_cp()**, **hw_deleteobject()**.

hw_Identify (unknown)

identifies as user

```
int hw_identify (string username, string password)
```

Identifies as user with *username* and *password*. Identification is only valid for the current session. I do not thing this function will be needed very often. In most cases it will be easier to identify with the opening of the connection.

See also **hw_Connect()**.

hw_InCollections (unknown)

check if object ids in collections

```
array hw_incollections (int connection, array object_id_array, array collection_id_array, int return_collections)
```

Checks whether a set of objects (documents or collections) specified by the *object_id_array* is part of the collections listed in *collection_id_array*. When the fourth parameter *return_collections* is 0, the subset of object ids that is part of the collections (i.e., the documents or collections that are children of one or more collections of collection ids or their subcollections, recursively) is returned as an array. When the fourth parameter is 1, however, the set of collections that have one or more objects of this subset as children are returned as an array. This option allows a client to, e.g., highlight the part of the collection hierarchy that contains the matches of a previous query, in a graphical overview.

hw_Info (unknown)

info about connection

```
string hw_info (int connection)
```

Returns information about the current connection. The returned string has the following format: <Serverstring>, <Host>, <Port>, <Username>, <Port of Client>, <Byte swapping>

hw_InsColl (unknown)

insert collection

```
int hw_inscoll (int connection, int objectID, array object_array)
```

Inserts a new collection with attributes as in *object_array* into collection with object ID *objectID*.

hw_InsDoc (unknown)

insert document

```
int hw_insdoc (int connection, int parentID, string object_record, string text)
```

Inserts a new document with attributes as in *object_record* into collection with object ID *parentID*. This function inserts either an object record only or an object record and a pure ascii text in *text* if *text* is given. If you want to insert a general document of any kind use **hw_insertdocument()** instead.

See also **hw_InsertDocument()**, **hw_InsColl()**.

hw_InsertDocument (unknown)

upload any document

```
int hw_insertdocument (int connection, int parent_id, int hw_document)
```

Uploads a document into the collection with *parent_id*. The document has to be created before with **hw_NewDocument()**. Make sure that the object record of the new document contains at least the attributes: Type, DocumentType, Title and Name. Possibly you also want to set the MimeType. The functions returns the object id of the new document or false.

See also **hw_PipeDocument()**.

hw_InsertObject (unknown)

inserts an object record

```
int hw_insertobject (int connection, string object rec, string parameter)
```

Inserts an object into the server. The object can be any valid hyperwave object. See the HG-CSP documentation for a detailed information on how the parameters have to be.

Note: If you want to insert an Anchor, the attribute Position has always been set either to a start/end value or to 'invisible'. Invisible positions are needed if the annotation has no correspondig link in the annotation text.

See also **hw_PipeDocument()**, **hw_InsertDocument()**, **hw_InsDoc()**, **hw_InsColl()**.

hw_mapid (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Maps global id on virtual local id

```
int hw_mapid (int connection, int server id, int object id)
```

Maps a global object id on any hyperwave server, even those you did not connect to with **hw_connect()**, onto a virtual object id. This virtual object id can then be used as any other object id, e.g. to obtain the object record with **hw_getobject()**. The server id is the first part of the global object id (GOid) of the object which is actually the IP number as an integer.

Note: In order to use this function you will have to set the F_DISTRIBUTED flag, which can currently only be set at compile time in hg_comm.c. It is not set by default. Read the comment at the beginning of hg_comm.c

hw_Modifyobject (unknown)

modifies object record

```
int hw_modifyobject (int connection, int object_to_change, array remove, array
add, int mode)
```

This command allows to remove, add, or modify individual attributes of an object record. The object is specified by the Object ID *object_to_change*. The first array *remove* is a list of attributes to remove. The second array *add* is a list of attributes to add. In order to modify an attribute one will have to remove the old one and add a new one. **hw_modifyobject()** will always remove the attributes before it adds attributes unless the value of the attribute to remove is not a string or array.

The last parameter determines if the modification is performed recursively. 1 means recurive modification. If some of the objects cannot be modified they will be skiped without notice. **hw_error()** may not indicate an error though some of the objects could not be modified.

The keys of both arrays are the attributes name. The value of each array element can either be an array, a string or anything else. If it is an array each attribute value is constructed by the key of each element plus a colon and the value of each element. If it is a string it is taken as the attribute value. An empty string will result in a complete removal of that attribute. If the value is neither a string nor an array but something else, e.g. an integer, no operation at all will be performed on the attribute. This is neccessary if you want to to add a completely new attribute not just a new value for an existing attribute. If the remove array contained an empty string for that attribute, the attribute would be tried to be removed which would fail since it doesn't exist. The following addition of a new value for that attribute would also fail. Setting the value for that attribute to e.g. 0 would not even try to remove it and the addition will work.

If you would like to change the attribute ’Name’ with the current value ’books’ into ’articles’ you will have to create two arrays and call **hw_modifyobject()**.

Example 1. modifying an attribute

```
// $connect is an existing connection to the Hyperwave server
// $objid is the ID of the object to modify
$remarr = array("Name" => "books");
$addarr = array("Name" => "articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

In order to delete/add a name=value pair from/to the object record just pass the remove/add array and set the last/third parameter to an empty array. If the attribute is the first one with that name to add, set attribute value in the remove array to an integer.

Example 2. adding a completely new attribute

```
// $connect is an existing connection to the Hyperwave server
// $objid is the ID of the object to modify
$remarr = array("Name" => 0);
$addarr = array("Name" => "articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

Note: Multilingual attributes, e.g. ’Title’, can be modified in two ways. Either by providing the attributes value in its native form ’language’:’title’ or by providing an array with elements for each language as described above. The above example would than be:

Example 3. modifying Title attribute

```
$remarr = array("Title" => "en:Books");
$addarr = array("Title" => "en:Articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

or

Example 4. modifying Title attribute

```
$remarr = array("Title" => array("en" => "Books"));
$addarr = array("Title" => array("en" => "Articles", "ge"=>"Artikel"));
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

This removes the english title ’Books’ and adds the english title ’Articles’ and the german title ’Artikel’.

Example 5. removing attribute

```
$remarr = array("Title" => "");
$addarr = array("Title" => "en:Articles");
$hw_modifyobject($connect, $objid, $remarr, $addarr);
```

Note: This will remove all attributes with the name 'Title' and adds a new 'Title' attribute. This comes in handy if you want to remove attributes recursively.

Note: If you need to delete all attributes with a certain name you will have to pass an empty string as the attribute value.

Note: Only the attributes 'Title', 'Description' and 'Keyword' will properly handle the language prefix. If those attributes don't carry a language prefix, the prefix 'xx' will be assigned.

Note: The 'Name' attribute is somewhat special. In some cases it cannot be complete removed. You will get an error message 'Change of base attribute' (not clear when this happens). Therefore you will always have to add a new Name first and than remove the old one.

Note: You may not suround this function by calls to **hw_getandlock()** and **hw_unlock()**. **hw_modifyobject()** does this internally.

Returns TRUE if no error occurs otherwise FALSE.

hw_New_Document (unknown)

create new document

```
int hw_new_document (string object_record, string document_data, int
document_size)
```

Returns a new Hyperwave document with document data set to *document_data* and object record set to *object_record*. The length of the *document_data* has to passed in *document_size*This function does not insert the document into the Hyperwave server.

See also **hw_FreeDocument()**, **hw_DocumentSize()**, **hw_DocumentBodyTag()**, **hw_OutputDocument()**, **hw_InsertDocument()**.

hw_Objrec2Array (unknown)

convert attributes from object record to object array

```
array hw_objrec2array (string object_record)
```

Converts an *object_record* into an object array. The keys of the resulting array are the attributes names. Multiple attributes like 'Title' in different languages form its own array. The keys of this array are the left part to the colon of the attribute value. Currently only the attributes 'Title', 'Description' and 'Keyword' are treated properly. Other multiple attributes form an index array. Currently only the attribute 'Group' is handled properly.

See also **hw_array2objrec()**.

hw_OutputDocument (unknown)

prints hw_document

```
int hw_outputdocument (int hw_document)
```

Prints the document without the BODY tag.

hw_pConnect (unknown)

make a persistent database connection

```
int hw_pconnect (string host, int port, string username, string password)
```

Returns a connection index on success, or false if the connection could not be made. Opens a persistent connection to a Hyperwave server. Each of the arguments should be a quoted string, except for the port number. The *username* and *password* arguments are optional and can be left out. In such a case no identification with the server will be done. It is similar to identify as user anonymous. This function returns a connection index that is needed by other Hyperwave functions. You can have multiple persistent connections open at once.

See also **hw_Connect()**.

hw_PipeDocument (unknown)

retrieve any document

```
int hw_pipedocument (int connection, int objectID)
```

Returns the Hyperwave document with object ID *objectID*. If the document has anchors which can be inserted, they will have been inserted already. The document will be transfered via a special data connection which does not block the control connection.

See also **hw_GetText()** for more on link insertion, **hw_FreeDocument()**, **hw_DocumentSize()**, **hw_DocumentBodyTag()**, **hw_OutputDocument()**.

hw_Root (unknown)

root object id

```
int hw_root ()
```

Returns the object ID of the hyperroot collection. Currently this is always 0. The child collection of the hyperroot is the root collection of the connected server.

hw_Unlock (unknown)

unlock object

```
int hw_unlock (int connection, int objectID)
```

Unlocks a document, so other users regain access.

See also **hw_GetAndLock()**.

hw_Who (unknown)

List of currently logged in users

```
int hw_who (int connection)
```

Returns an array of users currently logged into the Hyperwave server. Each entry in this array is an array itself containing the elements id, name, system, onSinceDate, onSinceTime, TotalTime and self. 'self' is 1 if this entry belongs to the user who initianted the request.

hw_Username (unknown)

name of currently logged in user

```
string hw_getusername (int connection)
```

Returns the username of the connection.

XXVII. Image functions

You can use the image functions in PHP to get the size of JPEG, GIF, and PNG images, and if you have the GD library (available at http://www.boutell.com/gd/) you will also be able to create and manipulate images.

GetImageSize (unknown)

Get the size of a GIF, JPEG or PNG image

```
array getimagesize (string filename [, array imageinfo])
```

The **GetImageSize()** function will determine the size of any GIF, JPG or PNG image file and return the dimensions along with the file type and a height/width text string to be used inside a normal HTML IMG tag.

Returns an array with 4 elements. Index 0 contains the width of the image in pixels. Index 1 contains the height. Index 2 a flag indicating the type of the image. 1 = GIF, 2 = JPG, 3 = PNG. Index 3 is a text string with the correct "height=xxx width=xxx" string that can be used directly in an IMG tag.

Example 1. GetImageSize

```
<?php $size = GetImageSize ("img/flag.jpg"); ?>
<IMG SRC="img/flag.jpg" <?php echo $size[3]; ?>
```

The optional *imageinfo* parameter allows you to extract some extended information from the image file. Currently this will return the diffrent JPG APP markers in an associative Array. Some Programs use these APP markers to embedd text information in images. A very common one in to embed IPTC http://www.xe.net/iptc/ information in the APP13 marker. You can use the **iptcparse()** function to parse the binary APP13 marker into something readable.

Example 2. GetImageSize returning IPTC

```
<?php
    $size = GetImageSize ("testimg.jpg",&$info);
    if (isset ($info["APP13"])) {
        $iptc = iptcparse ($info["APP13"]);
        var_dump ($iptc);
    }
?>
```

Note: This function does not require the GD image library.

ImageArc (unknown)

Draw a partial ellipse

```
int imagearc (int im, int cx, int cy, int w, int h, int s, int e, int col)
```

ImageArc() draws a partial ellipse centered at *cx*, *cy* (top left is 0, 0) in the image represented by im. *w* and *h* specifies the ellipse's width and height respectively while the start and end points are specified in degrees indicated by the *s* and *e*. arguments.

ImageChar (unknown)

Draw a character horizontally

```
int imagechar (int im, int font, int x, int y, string c, int col)
```

ImageChar() draws the first character of *c* in the image identified by *id* with its upper-left at *x*,*y* (top left is 0, 0) with the color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used (with higher numbers corresponding to larger fonts).

See also **imageloadfont()**.

ImageCharUp (unknown)

Draw a character vertically

```
int imagecharup (int im, int font, int x, int y, string c, int col)
```

ImageCharUp() draws the character *c* vertically in the image identified by *im* at coordinates *x*, *y* (top left is 0, 0) with the color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used.

See also **imageloadfont()**.

ImageColorAllocate (unknown)

Allocate a color for an image

```
int imagecolorallocate (int im, int red, int green, int blue)
```

ImageColorAllocate() returns a color identifier representing the color composed of the given RGB components. The *im* argument is the return from the **imagecreate()** function. **ImageColorAllocate()** must be called to create each color that is to be used in the image represented by *im*.

```
$white = ImageColorAllocate ($im, 255, 255, 255);
$black = ImageColorAllocate ($im, 0, 0, 0);
```

ImageColorAt (unknown)

Get the index of the color of a pixel

```
int imagecolorat (int im, int x, int y)
```

Returns the index of the color of the pixel at the specified location in the image.

See also **imagecolorset()** and **imagecolorsforindex()**.

ImageColorClosest (unknown)

Get the index of the closest color to the specified color

```
int imagecolorclosest (int im, int red, int green, int blue)
```

Returns the index of the color in the palette of the image which is "closest" to the specified RGB value.

The "distance" between the desired color and each color in the palette is calculated as if the RGB values represented points in three-dimensional space.

See also **imagecolorexact()**.

ImageColorExact (unknown)

Get the index of the specified color

```
int imagecolorexact (int im, int red, int green, int blue)
```

Returns the index of the specified color in the palette of the image.

If the color does not exist in the image's palette, -1 is returned.

See also **imagecolorclosest()**.

ImageColorResolve (unknown)

Get the index of the specified color or its closest possible alternative

```
int imagecolorresolve (int im, int red, int green, int blue)
```

This function is guaranteed to return a color index for a requested color, either the exact color or the closest possible alternative.

See also **imagecolorclosest()**.

ImageColorSet (unknown)

Set the color for the specified palette index

```
bool imagecolorset (int im, int index, int red, int green, int blue)
```

This sets the specified index in the palette to the specified color. This is useful for creating flood-fill-like effects in paletted images without the overhead of performing the actual flood-fill.

See also **imagecolorat()**.

ImageColorsForIndex (unknown)

Get the colors for an index

```
array imagecolorsforindex (int im, int index)
```

This returns an associative array with red, green, and blue keys that contain the appropriate values for the specified color index.

See also **imagecolorat()** and **imagecolorexact()**.

ImageColorsTotal (unknown)

Find out the number of colors in an image's palette

```
int imagecolorstotal (int im)
```

This returns the number of colors in the specified image's palette.

See also **imagecolorat()** and **imagecolorsforindex()**.

ImageColorTransparent (unknown)

Define a color as transparent

```
int imagecolortransparent (int im [, int col])
```

ImageColorTransparent() sets the transparent color in the *im* image to *col*. *Im* is the image identifier returned by **ImageCreate()** and *col* is a color identifier returned by **ImageColorAllocate()**.

The identifier of the new (or current, if none is specified) transparent color is returned.

ImageCopyResized (unknown)

Copy and resize part of an image

```
int imagecopyresized (int dst_im, int src_im, int dstX, int dstY, int srcX, int
srcY, int dstW, int dstH, int srcW, int srcH)
```

ImageCopyResized() copies a rectangular portion of one image to another image. *dst_im* is the destination image, *src_im* is the source image identifier. If the source and destination coordinates and width and heights differ, appropriate stretching or shrinking of the image fragment will be performed. The coordinates refer to the upper left corner. This function can be used to copy regions within the same image (if *dst_im* is the same as *src_im*) but if the regions overlap the results will be unpredictable.

ImageCreate (unknown)

Create a new image

```
int imagecreate (int x_size, int y_size)
```

ImageCreate() returns an image identifier representing a blank image of size *x_size* by *y_size*.

ImageCreateFromGif (unknown)

Create a new image from file or URL

```
int imagecreatefromgif (string filename)
```

ImageCreateFromGif() returns an image identifier representing the image obtained from the given filename.

ImageCreateFromGif() returns an empty string on failure. It also outputs an error message, which unfortunately displays as a broken link in a browser. To ease debugging the following example will produce an error GIF:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```
function LoadGif ($imgname) {
    $im = @imagecreatefromgif ($imgname); /* Attempt to open */
    if ($im == "") { /* See if it failed */
        $im = ImageCreate (150, 30); /* Create a blank image */
        $bgc = ImageColorAllocate ($im, 255, 255, 255);
        $tc  = ImageColorAllocate ($im, 0, 0, 0);
        ImageFilledRectangle ($im, 0, 0, 150, 30, $bgc);
        /* Output an errmsg */
        ImageString($im, 1, 5, 5, "Error loading $imgname", $tc);
    }
    return $im;
}
```

Note: Since all GIF support was removed from the GD library in version 1.6, this function is not available if you are using that version of the GD library.

ImageCreateFromJPEG (unknown)

Create a new image from file or URL

```
int imagecreatefromjpeg (string filename)
```

ImageCreateFromJPEG() returns an image identifier representing the image obtained from the given filename.

ImagecreateFromJPEG() returns an empty string on failure. It also outputs an error message, which unfortunately displays as a broken link in a browser. To ease debugging the following example will produce an error JPEG:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```
function LoadJpeg ($imgname) {
    $im = @imagecreatefromjpeg ($imgname); /* Attempt to open */
    if ($im == "") { /* See if it failed */
        $im = ImageCreate (150, 30); /* Create a blank image */
        $bgc = ImageColorAllocate ($im, 255, 255, 255);
        $tc  = ImageColorAllocate ($im, 0, 0, 0);
        ImageFilledRectangle ($im, 0, 0, 150, 30, $bgc);
        /* Output an errmsg */
        ImageString ($im, 1, 5, 5, "Error loading $imgname", $tc);
    }
    return $im;
}
```

ImageCreateFromPNG (unknown)

Create a new image from file or URL

```
int imagecreatefrompng (string filename)
```

ImageCreateFromPNG() returns an image identifier representing the image obtained from the given filename.

ImageCreateFromPNG() returns an empty string on failure. It also outputs an error message, which unfortunately displays as a broken link in a browser. To ease debugging the following example will produce an error PNG:

Example 1. Example to handle an error during creation (courtesy vic@zymsys.com)

```
function LoadPNG ($imgname) {
    $im = @imagecreatefrompng ($imgname); /* Attempt to open */
    if ($im == "") { /* See if it failed */
        $im = ImageCreate (150, 30); /* Create a blank image */
        $bgc = ImageColorAllocate ($im, 255, 255, 255);
        $tc  = ImageColorAllocate ($im, 0, 0, 0);
        ImageFilledRectangle ($im, 0, 0, 150, 30, $bgc);
```

```
        /* Output an errmsg */
        ImageString ($im, 1, 5, 5, "Error loading $imgname", $tc);
    }
    return $im;
}
```

ImageDashedLine (unknown)

Draw a dashed line

```
int imagedashedline (int im, int x1, int y1, int x2, int y2, int col)
```

ImageDashedLine() draws a dashed line from *x1*, *y1* to *x2*, *y2* (top left is 0, 0) in image *im* of color *col*.

See also **ImageLine()**.

ImageDestroy (unknown)

Destroy an image

```
int imagedestroy (int im)
```

ImageDestroy() frees any memory associated with image *im*. *Im* is the image identifier returned by the **ImageCreate()** function.

ImageFill (unknown)

Flood fill

```
int imagefill (int im, int x, int y, int col)
```

ImageFill() performs a flood fill starting at coordinate *x*, *y* (top left is 0, 0) with color *col* in the image *im*.

ImageFilledPolygon (unknown)

Draw a filled polygon

```
int imagefilledpolygon (int im, array points, int num_points, int col)
```

ImageFilledPolygon() creates a filled polygon in image *im*. *Points* is a PHP array containing the polygon's vertices, ie. points[0] = x0, points[1] = y0, points[2] = x1, points[3] = y1, etc. *Num_points* is the total number of vertices.

ImageFilledRectangle (unknown)

Draw a filled rectangle

```
int imagefilledrectangle (int im, int x1, int y1, int x2, int y2, int col)
```

ImageFilledRectangle() creates a filled rectangle of color **col()** in image *im* starting at upper left coordinates *x1*, *y1* and ending at bottom right coordinates *x2*, *y2*. 0, 0 is the top left corner of the image.

ImageFillToBorder (unknown)

Flood fill to specific color

```
int imagefilltoborder (int im, int x, int y, int border, int col)
```

ImageFillToBorder() performs a flood fill whose border color is defined by *border*. The starting point for the fill is *x*, *y* (top left is 0, 0) and the region is filled with color *col*.

ImageFontHeight (unknown)

Get font height

```
int imagefontheight (int font)
```

Returns the pixel height of a character in the specified font.

See also **ImageFontWidth()** and **ImageLoadFont()**.

ImageFontWidth (unknown)

Get font width

```
int imagefontwidth (int font)
```

Returns the pixel width of a character in font.

See also **ImageFontHeight()** and **ImageLoadFont()**.

ImageGIF (unknown)

Output image to browser or file

```
int imagegif (int im [, string filename])
```

ImageGIF() creates the GIF file in filename from the image *im*. The *im* argument is the return from the **imagecreate()** function.

The image format will be GIF87a unless the image has been made transparent with **ImageColorTransparent()**, in which case the image format will be GIF89a.

The filename argument is optional, and if left off, the raw image stream will be output directly. By sending an image/gif content-type using **header()**, you can create a PHP script that outputs GIF images directly.

Note: Since all GIF support was removed from the GD library in version 1.6, this function is not available if you are using that version of the GD library.

ImageJPEG (unknown)

Output image to browser or file

```
int imagejpeg (int im [, string filename [, int quality]])
```

ImageJPEG() creates the JPEG file in filename from the image *im*. The *im* argument is the return from the **ImageCreate()** function.

The filename argument is optional, and if left off, the raw image stream will be output directly. To skip the filename argument in order to provide a quality argument just use an empty string (''). By sending an image/jpg content-type using **header()**, you can create a PHP script that outputs JPEG images directly.

Note: JPEG support is only available in PHP if PHP was compiled against GD-1.8 or later.

ImageInterlace (unknown)

Enable or disable interlace

```
int imageinterlace (int im [, int interlace])
```

ImageInterlace() turns the interlace bit on or off. If interlace is 1 the im image will be interlaced, and if interlace is 0 the interlace bit is turned off.

This functions returns whether the interlace bit is set for the image.

ImageLine (unknown)

Draw a line

```
int imageline (int im, int x1, int y1, int x2, int y2, int col)
```

ImageLine() draws a line from *x1*, *y1* to *x2*, *y2* (top left is 0, 0) in image im of color *col*.

See also **ImageCreate()** and **ImageColorAllocate()**.

ImageLoadFont (unknown)

Load a new font

```
int imageloadfont (string file)
```

ImageLoadFont() loads a user-defined bitmap font and returns an identifier for the font (that is always greater than 5, so it will not conflict with the built-in fonts).

The font file format is currently binary and architecture dependent. This means you should generate the font files on the same type of CPU as the machine you are running PHP on.

Table 1. Font file format

byte position	C data type	description
byte 0-3	int	number of characters in the font
byte 4-7	int	value of first character in the font (often 32 for space)
byte 8-11	int	pixel width of each character
byte 12-15	int	pixel height of each character
byte 16-	char	array with character data, one byte per pixel in each character, for a total of (nchars*width*height) bytes.

See also **ImageFontWidth()** and **ImageFontHeight()**.

ImagePolygon (unknown)

Draw a polygon

```
int imagepolygon (int im, array points, int num_points, int col)
```

ImagePolygon() creates a polygon in image id. *Points* is a PHP array containing the polygon's vertices, ie. points[0] = x0, points[1] = y0, points[2] = x1, points[3] = y1, etc. *Num_points* is the total number of

vertices.

See also **imagecreate()**.

ImagePSBBox (unknown)

Give the bounding box of a text rectangle using PostScript Type1 fonts

```
array imagepsbbox (string text, int font, int size [, int space [, int tightness
[, float angle]]])
```

Size is expressed in pixels.

Space allows you to change the default value of a space in a font. This amount is added to the normal value and can also be negative.

Tightness allows you to control the amount of white space between characters. This amount is added to the normal character width and can also be negative.

Angle is in degrees.

Parameters *space* and *tightness* are expressed in character space units, where 1 unit is 1/1000th of an em-square.

Parameters *space*, *tightness*, and *angle* are optional.

The bounding box is calculated using information available from character metrics, and unfortunately tends to differ slightly from the results achieved by actually rasterizing the text. If the angle is 0 degrees, you can expect the text to need 1 pixel more to every direction.

This function returns an array containing the following elements:

0	lower left x-coordinate
1	lower left y-coordinate
2	upper right x-coordinate
3	upper right y-coordinate

See also **imagepstext()**.

ImagePSEncodeFont (unknown)

Change the character encoding vector of a font

```
int imagepsencodefont (string encodingfile)
```

Loads a character encoding vector from from a file and changes the fonts encoding vector to it. As a PostScript fonts default vector lacks most of the character positions above 127, you'll definitely want to change this if you use an other language than english. The exact format of this file is described in T1libs documentation. T1lib comes with two ready-to-use files, IsoLatin1.enc and IsoLatin2.enc.

If you find yourself using this function all the time, a much better way to define the encoding is to set ps.default_encoding in the configuration file to point to the right encoding file and all fonts you load will automatically have the right encoding.

ImagePSFreeFont (unknown)

Free memory used by a PostScript Type 1 font

```
void imagepsfreefont (int fontindex)
```

See also **ImagePSLoadFont()**.

ImagePSLoadFont (unknown)

Load a PostScript Type 1 font from file

```
int imagepsloadfont (string filename)
```

In the case everything went right, a valid font index will be returned and can be used for further purposes. Otherwise the function returns false and prints a message describing what went wrong.

See also **ImagePSFreeFont()**.

ImagePSText (unknown)

To draw a text string over an image using PostScript Type1 fonts

```
array imagepstext (int image, string text, int font, int size, int foreground,
int background, int x, int y [, int space [, int tightness [, float angle [, int
antialias_steps]]]])
```

Size is expressed in pixels.

Foreground is the color in which the text will be painted. *Background* is the color to which the text will try to fade in with antialiasing. No pixels with the color *background* are actually painted, so the background image does not need to be of solid color.

The coordinates given by *x*, *y* will define the origin (or reference point) of the first character (roughly the lower-left corner of the character). This is different from the **ImageString()**, where *x*, *y* define the upper-right corner of the first character. Refer to PostScipt documentation about fonts and their measuring system if you have trouble understanding how this works.

Space allows you to change the default value of a space in a font. This amount is added to the normal value and can also be negative.

Tightness allows you to control the amount of white space between characters. This amount is added to the normal character width and can also be negative.

Angle is in degrees.

Antialias_steps allows you to control the number of colours used for antialiasing text. Allowed values are 4 and 16. The higher value is recommended for text sizes lower than 20, where the effect in text quality is quite visible. With bigger sizes, use 4. It's less computationally intensive.

Parameters *space* and *tightness* are expressed in character space units, where 1 unit is 1/1000th of an em-square.

Parameters *space*, *tightness*, *angle* and *antialias* are optional.

This function returns an array containing the following elements:

0	lower left x-coordinate
1	lower left y-coordinate
2	upper right x-coordinate
3	upper right y-coordinate

See also **imagepsbbox()**.

ImageRectangle (unknown)

Draw a rectangle

```
int imagerectangle (int im, int x1, int y1, int x2, int y2, int col)
```

ImageRectangle() creates a rectangle of color col in image im starting at upper left coordinate x1, y1 and ending at bottom right coordinate x2, y2. 0, 0 is the top left corner of the image.

ImageSetPixel (unknown)

Set a single pixel

```
int imagesetpixel (int im, int x, int y, int col)
```

ImageSetPixel() draws a pixel at *x*, *y* (top left is 0, 0) in image *im* of color *col*.

See also **ImageCreate()** and **ImageColorAllocate()**.

ImageString (unknown)

Draw a string horizontally

```
int imagestring (int im, int font, int x, int y, string s, int col)
```

ImageString() draws the string *s* in the image identified by *im* at coordinates *x*, *y* (top left is 0, 0) in color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used.

See also **ImageLoadFont()**.

ImageStringUp (unknown)

Draw a string vertically

```
int imagestringup (int im, int font, int x, int y, string s, int col)
```

ImageStringUp() draws the string *s* vertically in the image identified by *im* at coordinates *x*, *y* (top left is 0, 0) in color *col*. If font is 1, 2, 3, 4 or 5, a built-in font is used.

See also **ImageLoadFont()**.

ImageSX (unknown)

Get image width

```
int imagesx (int im)
```

ImageSX() returns the width of the image identified by *im*.

See also **ImageCreate()** and **ImageSY()**.

ImageSY (unknown)

Get image height

```
int imagesy (int im)
```

ImageSY() returns the height of the image identified by *im*.

See also **ImageCreate()** and **ImageSX()**.

ImageTTFBBox (unknown)

Give the bounding box of a text using TypeType fonts

```
array imagettfbbox (int size, int angle, string fontfile, string text)
```

This function calculates and returns the bounding box in pixels for a TrueType text.

text

The string to be measured.

size

The font size.

fontfile

The name of the TrueType font file. (Can also be an URL.)

angle

Angle in degrees in which *text* will be measured.

ImageTTFBBox() returns an array with 8 elements representing four points making the bounding box of the text:

0	lower left corner, X position
1	lower left corner, Y position
2	lower right corner, X position
3	lower right corner, Y position
4	upper right corner, X position
5	upper right corner, Y position
6	upper left corner, X position
7	upper left corner, Y position

The points are relative to the *text* regardless of the angle, so "upper left" means in the top left-hand corner seeing the text horizontallty.

This function requires both the GD library and the FreeType library.

See also **ImageTTFText()**.

ImageTTFText (unknown)

Write text to the image using TrueType fonts

```
array imagettftext (int im, int size, int angle, int x, int y, int col, string
fontfile, string text)
```

ImageTTFText() draws the string *text* in the image identified by *im*, starting at coordinates *x*, *y* (top left is 0, 0), at an angle of *angle* in color *col*, using the TrueType font file identified by *fontfile*.

The coordinates given by *x*, *y* will define the basepoint of the first character (roughly the lower-left corner of the character). This is different from the **ImageString()**, where x, y define the upper-right corner of the first character.

Angle is in degrees, with 0 degrees being left-to-right reading text (3 o'clock direction), and higher values representing a counter-clockwise rotation. (i.e., a value of 90 would result in bottom-to-top reading text).

Fontfile is the path to the TrueType font you wish to use.

Text is the text string which may include UTF-8 character sequences (of the form: {) to access characters in a font beyond the first 255.

Col is the color index. Using the negative of a color index has the effect of turning off antialiasing.

ImageTTFText() returns an array with 8 elements representing four points making the bounding box of the text. The order of the points is upper left, upper right, lower right, lower left. The points are relative to the text regardless of the angle, so "upper left" means in the top left-hand corner when you see the text horizontallty.

This example script will produce a black GIF 400x30 pixels, with the words "Testing..." in white in the font Arial.

Example 1. ImageTTFText

```
<?php
Header ("Content-type: image/gif");
$im = imagecreate (400, 30);
$black = ImageColorAllocate ($im, 0, 0, 0);
$white = ImageColorAllocate ($im, 255, 255, 255);
ImageTTFText ($im, 20, 0, 10, 20, $white, "/path/arial.ttf",
              "Testing... Omega: &#937;");
ImageGif ($im);
ImageDestroy ($im);
?>
```

This function requires both the GD library and the FreeType (http://www.freetype.org/) library.

See also **ImageTTFBBox()**.

XXVIII. IMAP, POP3 and NNTP functions

To get these functions to work, you have to compile PHP with `-with-imap`. That requires the c-client library to be installed. Grab the latest version from ftp://ftp.cac.washington.edu/imap/ and compile it. Then copy `c-client/c-client.a` to `/usr/local/lib/libc-client.a` or some other directory on your link path and copy `c-client/rfc822.h`, `mail.h` and `linkage.h` to `/usr/local/include` or some other directory in your include path.

Note that these functions are not limited to the IMAP protocol, despite their name. The underlying c-client library also supports NNTP, POP3 and local mailbox access methods.

This document can't go into detail on all the topics touched by the provided functions. Further information is provided by the documentation of the c-client library source (`docs/internal.txt`). and the following RFC documents:

- RFC821 (http://www.faqs.org/rfcs/rfc821.html): Simple Mail Transfer Protocol (SMTP).
- RFC822 (http://www.faqs.org/rfcs/rfc822.html): Standard for ARPA internet text messages.
- RFC2060 (http://www.faqs.org/rfcs/rfc2060.html): Internet Message Access Protocol (IMAP) Version 4rev1.
- RFC1939 (http://www.faqs.org/rfcs/rfc1939.html): Post Office Protocol Version 3 (POP3).
- RFC977 (http://www.faqs.org/rfcs/rfc977.html): Network News Transfer Protocol (NNTP).
- RFC2076 (http://www.faqs.org/rfcs/rfc2076.html): Common Internet Message Headers.
- RFC2045 (http://www.faqs.org/rfcs/rfc2045.html) , RFC2046 (http://www.faqs.org/rfcs/rfc2046.html) , RFC2047 (http://www.faqs.org/rfcs/rfc2047.html) , RFC2048 (http://www.faqs.org/rfcs/rfc2048.html) & RFC2049 (http://www.faqs.org/rfcs/rfc2049.html): Multipurpose Internet Mail Extensions (MIME).

A detailed overview is also available in the book Programming Internet Email (http://www.oreilly.com/catalog/progintemail/noframes.html) by David Wood.

imap_append (PHP3 , PHP4)

Append a string message to a specified mailbox

```
int imap_append (int imap_stream, string mbox, string message [, string flags])
```

Returns true on sucess, false on error.

imap_append() appends a string message to the specified mailbox *mbox*. If the optional *flags* is specified, writes the *flags* to that mailbox also.

When talking to the Cyrus IMAP server, you must use "\r\n" as your end-of-line terminator instead of "\n" or the operation will fail.

Example 1. imap_append() example

```
$stream = imap_open("{your.imap.host}INBOX.Drafts","username", "password");

$check = imap_check($stream);
print "Msg Count before append: ". $check->Nmsgs."\n";

imap_append($stream,"{your.imap.host}INBOX.Drafts"
                  ,"From: me@my.host\r\n"
                  ."To: you@your.host\r\n"
                  ."Subject: test\r\n"
                  ."\r\n"
                  ."this is a test message, please ignore\r\n"
                  );

$check = imap_check($stream);
print "Msg Count after append : ". $check->Nmsgs."\n";

imap_close($stream);
```

imap_base64 (PHP3 , PHP4)

Decode BASE64 encoded text

```
string imap_base64 (string text)
```

imap_base64() function decodes BASE-64 encoded text (see RFC2045 (http://www.faqs.org/rfcs/rfc2045.html), Section 6.8). The decoded message is returned as a string.

See also **imap_binary()**.

Example 1. imap_mailboxmsginfo() example

```
$mbox = imap_open("{your.imap.host}INBOX","username", "password")
      || die("can't connect: ".imap_last_error());
```

```
$check = imap_mailboxmsginfo($mbox);

if($check) {
  print "Date: "    . $check->Date    ."<br>\n" ;
  print "Driver: "  . $check->Driver  ."<br>\n" ;
  print "Mailbox: " . $check->Mailbox ."<br>\n" ;
  print "Messages: ". $check->Nmsgs   ."<br>\n" ;
  print "Recent: "  . $check->Recent  ."<br>\n" ;
  print "Size: "    . $check->Size    ."<br>\n" ;
} else
  print "imap_check() failed: ".imap_lasterror(). "<br>\n";

imap_close($mbox);
```

imap_body (PHP3 , PHP4)

Read the message body

```
string imap_body (int imap_stream, int msg_number [, int flags])
```

imap_body() returns the body of the message, numbered *msg_number* in the current mailbox. The optional *flags* are a bit mask with one or more of the following:

- FT_UID - The *msgno* is a UID
- FT_PEEK - Do not set the \Seen flag if not already set
- FT_INTERNAL - The return string is in internal format, will not canonicalize to CRLF.

imap_check (PHP3 , PHP4)

Check current mailbox

```
object imap_check (int imap_stream)
```

Returns information about the current mailbox. Returns FALSE on failure.

The **imap_check()** function checks the current mailbox status on the server and returns the information in an object with following properties:

- Date - last change of mailbox contents
- Driver - protocol used to access this mailbox: POP3, IMAP, NNTP
- Mailbox - the mailbox name
- Nmsgs - number of messages in the mailbox
- Recent - number of recent messages in the mailbox

imap_close (PHP3 , PHP4)

Close an IMAP stream

```
int imap_close (int imap_stream [, int flags])
```

Close the imap stream. Takes an optional *flag* CL_EXPUNGE, which will silently expunge the mailbox before closing, removing all messages marked for deletion.

imap_createmailbox (PHP3 , PHP4)

Create a new mailbox

```
int imap_createmailbox (int imap_stream, string mbox)
```

imap_createmailbox() creates a new mailbox specified by *mbox*. Names containing international characters should be encoded by **imap_utf7_encode()**

Returns true on success and false on error.

See also **imap_renamemailbox()**, **imap_deletemailbox()** and **imap_open()** for the format of *mbox* names.

Example 1. imap_createmailbox() example

```
$mbox = imap_open("{your.imap.host}","username","password",OP_HALFOPEN)
     || die("can't connect: ".imap_last_error());

$name1 = "phpnewbox";
$name2 = imap_utf7_encode("phpnewböx");

$newname = $name1;

echo "Newname will be '$name1'<br>\n";

# we will now create a new mailbox "phptestbox" in your inbox folder,
# check its status after creation and finaly remove it to restore
# your inbox to its initial state
if(@imap_createmailbox($mbox,imap_utf7_encode("{your.imap.host}INBOX.$newname"))) {
  $status = @imap_status($mbox,"{your.imap.host}INBOX.$newname",SA_ALL);
  if($status) {
    print("your new mailbox '$name1' has the following status:<br>\n");
    print("Messages:   ". $status->messages   )."<br>\n";
    print("Recent:     ". $status->recent     )."<br>\n";
    print("Unseen:     ". $status->unseen     )."<br>\n";
    print("UIDnext:    ". $status->uidnext    )."<br>\n";
    print("UIDvalidity:". $status->uidvalidity)."<br>\n";

    if(imap_renamemailbox($mbox,"{your.imap.host}INBOX.$newname","{your.imap.host}INBOX.$
      echo "renamed new mailbox from '$name1' to '$name2'<br>\n";
      $newname=$name2;
    } else {
```

```
      print "imap_renamemailbox on new mail-
box failed: ".imap_last_error()."<br>\n";
    }
  } else {
    print  "imap_status on new mailbox failed: ".imap_last_error()."<br>\n";
  }
  if(@imap_deletemailbox($mbox,"{your.imap.host}INBOX.$newname")) {
    print "new mailbox removed to restore initial state<br>\n";
  } else {
    print  "imap_deletemailbox on new mail-
box failed: ".implode("<br>\n",imap_errors())."<br>\n";
  }

} else {
  print  "could not create new mail-
box: ".implode("<br>\n",imap_errors())."<br>\n";
}

imap_close($mbox);
```

imap_delete (PHP3 , PHP4)

Mark a messge for deletion from current mailbox

```
int imap_delete (int imap_stream, int msg_number [, int flags])
```

Returns true.

imap_delete() function marks message pointed by *msg_number* for deletion. The optional *flags* parameter only has a single option, *FT_UID*, which tells the function to treat the *msg_number* argument as a *UID*. Messages marked for deletion will stay in the mailbox until either **imap_expunge()** is called or **imap_close()** is called with the optional parameter CL_EXPUNGE.

imap_deletemailbox (PHP3 , PHP4)

Delete a mailbox

```
int imap_deletemailbox (int imap_stream, string mbox)
```

imap_deletemailbox() deletes the specified mailbox (see **imap_open()** for the format of *mbox* names).

Returns true on success and false on error.

See also **imap_createmailbox()**, **imap_reanmemailbox()**, and **imap_open()** for the format of *mbox*.

imap_expunge (PHP3 , PHP4)

Delete all messages marked for deletion

```
int imap_expunge (int imap_stream)
```

imap_expunge() deletes all the messages marked for deletion by **imap_delete()**, **imap_move_mail()**, or **imap_setflag_full()**.

Returns true.

imap_fetchbody (PHP3 , PHP4)

Fetch a particular section of the body of the message

```
string imap_fetchbody (int imap_stream, int msg_number, string part_number [,
flags flags])
```

This function causes a fetch of a particular section of the body of the specified messages as a text string and returns that text string. The section specification is a string of integers delimited by period which index into a body part list as per the IMAP4 specification. Body parts are not decoded by this function.

The options for **imap_fetchbody()** is a bitmask with one or more of the following:

- FT_UID - The *msg_number* is a UID
- FT_PEEK - Do not set the \Seen flag if not already set
- FT_INTERNAL - The return string is in "internal" format, without any attempt to canonicalize CRLF.

imap_fetchstructure (PHP3 , PHP4)

Read the structure of a particular message

```
object imap_fetchstructure  (int imap_stream, int msg_number [, int flags])
```

This function fetches all the structured information for a given message. The optional *flags* parameter only has a single option, *FT_UID*, which tells the function to treat the *msg_number* argument as a *UID*. The returned object includes the envelope, internal date, size, flags and body structure along with a similar object for each mime attachement. The structure of the returned objects is as follows:

Table 1. Returned Objects for imap_fetchstructure()

type	Primary body type
encoding	Body transfer encoding
ifsubtype	True if there is a subtype string

subtype	MIME subtype
ifdescription	True if there is a description string
description	Content description string
ifid	True if there is an identification string
id	Identification string
lines	Number of lines
bytes	Number of bytes
ifdisposition	True if there is a disposition string
disposition	Disposition string
ifdparameters	True if the dparameters array exists
dparameters	Disposition parameter array
ifparameters	True if the parameters array exists
parameters	MIME parameters array
parts	Array of objects describing each message part

1. dparameters is an array of objects where each object has an "attribute" and a "value" property.
2. Parameter is an array of objects where each object has an "attributte" and a "value" property.
3. Parts is an array of objects identical in structure to the top-level object, with the limitation that it cannot contain further 'parts' objects.

Table 2. Primary body type

0	text
1	multipart
2	message
3	application
4	audio
5	image
6	video
7	other

Table 3. Transfer encodings

0	7BIT
1	8BIT
2	BINARY
3	BASE64
4	QUOTED-PRINTABLE
5	OTHER

imap_header (PHP3 , PHP4)

Read the header of the message

```
object imap_header (int imap_stream, int msg_number [, int fromlength [, int
subjectlength [, string defaulthost]]])
```

This function returns an object of various header elements.

remail, date, Date, subject, Subject, in_reply_to, message_id,
newsgroups, followup_to, references

message flags:
Recent - 'R' if recent and seen,
'N' if recent and not seen,
' ' if not recent
Unseen - 'U' if not seen AND not recent,
' ' if seen OR not seen and recent
Answered -'A' if answered,
' ' if unanswered
Deleted - 'D' if deleted,
' ' if not deleted
Draft - 'X' if draft,
' ' if not draft
Flagged - 'F' if flagged,
' ' if not flagged

NOTE that the Recent/Unseen behavior is a little odd. If you want to
know if a message is Unseen, you must check for

Unseen == 'U' || Recent == 'N'

toaddress (full to: line, up to 1024 characters)

to[] (returns an array of objects from the To line, containing):
personal
adl
mailbox
host

fromaddress (full from: line, up to 1024 characters)

from[] (returns an array of objects from the From line, containing):
personal
adl
mailbox
host

ccaddress (full cc: line, up to 1024 characters)
cc[] (returns an array of objects from the Cc line, containing):
personal

adl
mailbox
host

bccaddress (full bcc line, up to 1024 characters)
bcc[] (returns an array of objects from the Bcc line, containing):
personal
adl
mailbox
host

reply_toaddress (full reply_to: line, up to 1024 characters)
reply_to[] (returns an array of objects from the Reply_to line, containing):
personal
adl
mailbox
host

senderaddress (full sender: line, up to 1024 characters)
sender[] (returns an array of objects from the sender line, containing):
personal
adl
mailbox
host

return_path (full return-path: line, up to 1024 characters)
return_path[] (returns an array of objects from the return_path line, containing):
personal
adl
mailbox
host

udate (mail message date in unix time)

fetchfrom (from line formatted to fit *fromlength* characters)

fetchsubject (subject line formatted to fit *subjectlength* characters)

imap_rfc822_parse_headers (PHP4 >= 4.0RC1)

Parse mail headers from a string

```
object imap_rfc822_parse_headers (string headers [, string defaulthost])
```

This function returns an object of various header elements, similar to **imap_header()**, except without the flags and other elements that come from the IMAP server.

imap_headers (PHP3 , PHP4)

Returns headers for all messages in a mailbox

```
array imap_headers (int imap_stream)
```

Returns an array of string formatted with header info. One element per mail message.

imap_listmailbox (PHP3 , PHP4)

Read the list of mailboxes

```
array imap_listmailbox (int imap_stream, string ref, string pattern)
```

Returns an array containing the names of the mailboxes. See **imap_getmailboxes()** for a description of *ref* and *pattern*.

Example 1. imap_getmailboxes() example

```
$mbox = imap_open("{your.imap.host}","username","password",OP_HALFOPEN)
      || die("can't connect: ".imap_last_error());

$list = imap_listmailbox($mbox,"{your.imap.host}","*");
if(is_array($list)) {
  reset($list);
  while (list($key, $val) = each($list))
    print imap_utf7_decode($val)."<br>\n";
} else
  print "imap_listmailbox failed: ".imap_last_error()."\n";

imap_close($mbox);
```

imap_getmailboxes (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

Read the list of mailboxes, returning detailed information on each one

```
array imap_getmailboxes (int imap_stream, string ref, string pattern)
```

Returns an array of objects containing mailbox information. Each object has the attributes *name*, specifying the full name of the mailbox; *delimiter*, which is the hierarchy delimiter for the part of the hierarchy this mailbox is in; and *attributes*. *Attributes* is a bitmask that can be tested against:

- LATT_NOINFERIORS - This mailbox has no "children" (there are no mailboxes below this one).
- LATT_NOSELECT - This is only a container, not a mailbox - you cannot open it.
- LATT_MARKED - This mailbox is marked. Only used by UW-IMAPD.
- LATT_UNMARKED - This mailbox is not marked. Only used by UW-IMAPD.

Mailbox names containing international Characters outside the printable ASCII range will be encoded and may be decoded by **imap_utf7_decode()**.

ref should normally be just the server specification as described in **imap_open()**, and *pattern* specifies where in the mailbox hierarchy to start searching. If you want all mailboxes, pass '*' for *pattern*.

There are two special characters you can pass as part of the *pattern*: '*' and '%'. '*' means to return all mailboxes. If you pass *pattern* as '*', you will get a list of the entire mailbox hierarchy. '%' means to return the current level only. '%' as the *pattern* parameter will return only the top level mailboxes; '~/mail/%' on UW_IMAPD will return every mailbox in the ~/mail directory, but none in subfolders of that directory.

Example 1. imap_getmailboxes() example

```
$mbox = imap_open("{your.imap.host}","username","password",OP_HALFOPEN)
      || die("can't connect: ".imap_last_error());

$list = imap_getmailboxes($mbox,"{your.imap.host}","*");
if(is_array($list)) {
  reset($list);
  while (list($key, $val) = each($list))
  {
    print "($key) ";
    print imap_utf7_decode($val->name).",";
    print "'".$val->delimiter."',";
    print $val->attributes."<br>\n";
  }
} else
  print "imap_getmailboxes failed: ".imap_last_error()."\n";

imap_close($mbox);
```

imap_listsubscribed (PHP3 , PHP4)

List all the subscribed mailboxes

```
array imap_listsubscribed (int imap_stream, string ref, string pattern)
```

Returns an array of all the mailboxes that you have subscribed. This is almost identical to **imap_listmailbox()**, but will only return mailboxes the user you logged in as has subscribed.

imap_getsubscribed (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

List all the subscribed mailboxes

```
array imap_getsubscribed (int imap_stream, string ref, string pattern)
```

This function is identical to **imap_getmailboxes()**, except that it only returns mailboxes that the user is subscribed to.

imap_mail_copy (PHP3 , PHP4)

Copy specified messages to a mailbox

```
int imap_mail_copy (int imap_stream, string msglist, string mbox [, int flags])
```

Returns true on success and false on error.

Copies mail messages specified by *msglist* to specified mailbox. *msglist* is a range not just message numbers (as described in RFC2060 (http://www.faqs.org/rfcs/rfc2060.html)).

Flags is a bitmask of one or more of

- CP_UID - the sequence numbers contain UIDS
- CP_MOVE - Delete the messages from the current mailbox after copying

imap_mail_move (PHP3 , PHP4)

Move specified messages to a mailbox

```
int imap_mail_move (int imap_stream, string msglist, string mbox [, int flags])
```

Moves mail messages specified by *msglist* to specified mailbox. *msglist* is a range not just message numbers (as described in RFC2060 (http://www.faqs.org/rfcs/rfc2060.html)).

Flags is a bitmask and may contain the single option

- CP_UID - the sequence numbers contain UIDS

Returns true on success and false on error.

imap_num_msg (PHP3 , PHP4)

Gives the number of messages in the current mailbox

```
int imap_num_msg (int imap_stream)
```

Return the number of messages in the current mailbox.

imap_num_recent (PHP3 , PHP4)

Gives the number of recent messages in current mailbox

```
int imap_num_recent (int imap_stream)
```

Returns the number of recent messages in the current mailbox.

imap_open (PHP3 , PHP4)

Open an IMAP stream to a mailbox

```
int imap_open (string mailbox, string username, string password [, int flags])
```

Returns an IMAP stream on success and false on error. This function can also be used to open streams to POP3 and NNTP servers, but some functions and features are not available on IMAP servers.

A mailbox name consists of a server part and a mailbox path on this server. The special name INBOX stands for the current users personal mailbox. The server part, which is enclosed in '{' and '}', consists of the servers name or ip address, a protocol secification (beginning with '/') and an optional port specifier beginnung with ':'. The server part is mandatory in all mailbox parameters. Mailbos names that contain international characters besides those in the printable ASCII space have to be encoded with **imap_utf7_encode()**.

The options are a bit mask with one or more of the following:

- OP_READONLY - Open mailbox read-only
- OP_ANONYMOUS - Dont use or update a `.newsrc` for news (NNTP only)
- OP_HALFOPEN - For IMAP and NNTP names, open a connection but dont open a mailbox
- CL_EXPUNGE - Expunge mailbox automatically upon mailbox close

To connect to an IMAP server running on port 143 on the local machine, do the following:

```
$mbox = imap_open ("{localhost:143}INBOX", "user_id", "password");
```

To connect to a POP3 server on port 110 on the local server, use:

```
$mbox = imap_open ("{localhost/pop3:110}INBOX", "user_id", "password");
```

To connect to an NNTP server on port 119 on the local server, use:

```
$nntp = imap_open ("{localhost/nntp:119}comp.test", "", "");
```

To connect to a remote server replace "localhost" with the name or the IP address of the server you want to connect to.

Example 1. imap_open() example

```
$mbox = imap_open ("{your.imap.host:143}", "username", "password");

echo "<p><h1>Mailboxes</h1>\n";
$folders = imap_listmailbox ($mbox, "{your.imap.host:143}", "*");

if ($folders == false) {
    echo "Call failed<br>\n";
} else {
    while (list ($key, $val) = each ($folders)) {
        echo $val."<br>\n";
    }
}

echo "<p><h1>Headers in INBOX</h1>\n";
$headers = imap_headers ($mbox);

if ($headers == false) {
    echo "Call failed<br>\n";
} else {
    while (list ($key,$val) = each ($headers)) {
        echo $val."<br>\n";
    }
}

imap_close($mbox);
```

imap_ping (PHP3 , PHP4)

Check if the IMAP stream is still active

```
int imap_ping (int imap_stream)
```

Returns true if the stream is still alive, false otherwise.

imap_ping() function pings the stream to see it is still active. It may discover new mail; this is the preferred method for a periodic "new mail check" as well as a "keep alive" for servers which have inactivity timeout. (As PHP scripts do not tend to run that long, i can hardly imagine that this function will be usefull to anyone.)

imap_renamemailbox (PHP3 , PHP4)

Rename an old mailbox to new mailbox

```
int imap_renamemailbox (int imap_stream, string old_mbox, string new_mbox)
```

This function renames on old mailbox to new mailbox (see **imap_open()** for the format of *mbox* names).

Returns true on success and false on error.

See also **imap_createmailbox()**, **imap_deletemailbox()**, and **imap_open()** for the format of *mbox*.

imap_reopen (PHP3 , PHP4)

Reopen IMAP stream to new mailbox

```
int imap_reopen (string imap_stream, string mailbox [, string flags])
```

This function reopens the specified stream to a new mailbox on an IMAP or NNTP server.

The options are a bit mask with one or more of the following:

- OP_READONLY - Open mailbox read-only
- OP_ANONYMOUS - Dont use or update a `.newsrc` for news (NNTP only)
- OP_HALFOPEN - For IMAP and NNTP names, open a connection but dont open a mailbox.
- CL_EXPUNGE - Expunge mailbox automatically upon mailbox close (see also **imap_delete()** and **imap_expunge()**)

Returns true on success and false on error.

imap_subscribe (PHP3 , PHP4)

Subscribe to a mailbox

```
int imap_subscribe (int imap_stream, string mbox)
```

Subscribe to a new mailbox.

Returns true on success and false on error.

imap_undelete (PHP3 , PHP4)

Unmark the message which is marked deleted

```
int imap_undelete (int imap_stream, int msg_number)
```

This function removes the deletion flag for a specified message, which is set by **imap_delete()** or **imap_mail_move()**.

Returns true on success and false on error.

imap_unsubscribe (PHP3 , PHP4)

Unsubscribe from a mailbox

```
int imap_unsubscribe (int imap_stream, string mbox)
```

Unsubscribe from a specified mailbox.

Returns true on success and false on error.

imap_qprint (PHP3 , PHP4)

Convert a quoted-printable string to an 8 bit string

```
string imap_qprint (string string)
```

Convert a quoted-printable string to an 8 bit string (according to RFC2045 (http://www.faqs.org/rfcs/rfc2045.html), section 6.7).

Returns an 8 bit (binary) string.

See also **imap_8bit()**.

imap_8bit (PHP3 , PHP4)

Convert an 8bit string to a quoted-printable string

```
string imap_8bit (string string)
```

Convert an 8bit string to a quoted-printable string (according to RFC2045 (http://www.faqs.org/rfcs/rfc2045.html), section 6.7).

Returns a quoted-printable string.

See also **imap_qprint()**.

imap_binary (PHP3 >= 3.0.2, PHP4)

Convert an 8bit string to a base64 string

```
string imap_binary (string string)
```

Convert an 8bit string to a base64 string (according to RFC2045 (http://www.faqs.org/rfcs/rfc2045.html), Section 6.8).

Returns a base64 string.

See also **imap_base64()**.

imap_scanmailbox (PHP3 , PHP4)

Read the list of mailboxes, takes a string to search for in the text of the mailbox

```
array imap_scanmailbox (int imap_stream, string content)
```

Returns an array containing the names of the mailboxes that have *string* in the text of the mailbox. This function is simmilar to **imap_listmailbox()**, but it will additionally check for the presence of the string *content* inside the mailbox data.

imap_mailboxmsginfo (PHP3 >= 3.0.2, PHP4)

Get information about the current mailbox

```
object imap_mailboxmsginfo (int imap_stream)
```

Returns information about the current mailbox. Returns FALSE on failure.

The **imap_mailboxmsginfo()** function checks the current mailbox status on the server. It is similar to **imap_status()**, but will additionally sum up the size of all messages in the mailbox, which will take some additional time to execute. It returns the information in an object with following properties.

Table 1. Mailbox properties

Date	date of last change
Driver	driver
Mailbox	name of the mailbox
Nmsgs	number of messages
Recent	number of recent messages
Unread	number of unread messages
Size	mailbox size

imap_rfc822_write_address (PHP3 >= 3.0.2, PHP4)

Returns a properly formatted email address given the mailbox, host, and personal info.

```
string imap_rfc822_write_address  (string mailbox, string host, string personal)
```

Returns a properly formatted email address as defined in RFC822 (http://www.faqs.org/rfcs/rfc822.html) given the mailbox, host, and personal info.

Example 1. imap_rfc822_write_address() example

```
print imap_rfc822_write_address("hartmut","cvs.php.net","Hartmut Holzgraefe")."\n";
```

imap_rfc822_parse_adrlist (PHP3 >= 3.0.2, PHP4)

Parses an address string

```
array imap_rfc822_parse_adrlist  (string address, string default_host)
```

This function parses the address string as defined in RFC822 (http://www.faqs.org/rfcs/rfc822.html) and for each address, returns an array of objects. The objects properties are:

- mailbox - the mailbox name (username)
- host - the host name
- personal - the personal name
- adl - at domain source route

Example 1. imap_rfc822_parse_adrlist() example

```
$address_string = "Hartmut Holzgraefe <hartmut@cvs.php.net>, postmas-
ter@somedomain.net, root";
$address_array  = imap_rfc822_parse_adrlist($address_string,"somedomain.net");
if(! is_array($address_array)) die("somethings wrong\n");

reset($address_array);
while(list($key,$val)=each($address_array)){
  print "mailbox : ".$val->mailbox."<br>\n";
  print "host    : ".$val->host."<br>\n";
  print "personal: ".$val->personal."<br>\n";
  print "adl     : ".$val->adl."<p>\n";
}
```

imap_setflag_full (PHP3 >= 3.0.3, PHP4)

Sets flags on messages

```
string imap_setflag_full (int stream, string sequence, string flag, string
options)
```

This function causes a store to add the specified flag to the flags set for the messages in the specified sequence.

The flags which you can set are "\\Seen", "\\Answered", "\\Flagged", "\\Deleted", "\\Draft", and "\\Recent" (as defined by RFC2060).

The options are a bit mask with one or more of the following:

ST_UID The sequence argument contains UIDs instead of sequence numbers

Example 1. imap_setflag_full() example

```
$mbox = imap_open("{your.imap.host:143}","username","password")
     || die("can't connect: ".imap_last_error());

$status = imap_setflag_full($mbox,"2,5","\\Seen \\Flagged");

print gettype($status)."\n";
print $status."\n";

imap_close($mbox);
```

imap_clearflag_full (PHP3 >= 3.0.3, PHP4)

Clears flags on messages

```
string imap_clearflag_full  (int stream, string sequence, string flag, string
options)
```

This function causes a store to delete the specified flag to the flags set for the messages in the specified sequence. The flags which you can unset are "\\Seen", "\\Answered", "\\Flagged", "\\Deleted", "\\Draft", and "\\Recent" (as defined by RFC2060).

The options are a bit mask with one or more of the following:

ST_UID The sequence argument contains UIDs instead of sequence numbers

imap_sort (PHP3 >= 3.0.3, PHP4)

Sort an array of message headers

```
string imap_sort (int stream, int criteria, int reverse, int options)
```

Returns an array of message numbers sorted by the given parameters.

Reverse is 1 for reverse-sorting.

Criteria can be one (and only one) of the following:

SORTDATE message Date
SORTARRIVAL arrival date
SORTFROM mailbox in first From address
SORTSUBJECT message Subject
SORTTO mailbox in first To address
SORTCC mailbox in first cc address
SORTSIZE size of message in octets

The flags are a bitmask of one or more of the following:

SE_UID Return UIDs instead of sequence numbers
SE_NOPREFETCH Don't prefetch searched messages.

imap_fetchheader (PHP3 >= 3.0.3, PHP4)

Returns header for a message

```
string imap_fetchheader (int imap_stream, int msgno, int flags)
```

This function causes a fetch of the complete, unfiltered RFC822 (http://www.faqs.org/rfcs/rfc822.html) format header of the specified message as a text string and returns that text string.

The options are:

FT_UID The msgno argument is a UID
FT_INTERNAL The return string is in "internal" format,
without any attempt to canonicalize to CRLF
newlines
FT_PREFETCHTEXT The RFC822.TEXT should be pre-fetched at the
same time. This avoids an extra RTT on an
IMAP connection if a full message text is
desired (e.g. in a "save to local file"
operation)

imap_uid (PHP3 >= 3.0.3, PHP4)

This function returns the UID for the given message sequence number

```
int imap_uid (int imap_stream, int msgno)
```

This function returns the UID for the given message sequence number. An UID is an unique identifier that will not change over time while a message sequence number may change whenever the content of the mailbox changes. This function is the inverse of **imap_msgno()**.

imap_msgno (PHP3 >= 3.0.3, PHP4)

This function returns the message sequence number for the given UID

```
int imap_msgno (int imap_stream, int uid)
```

This function returns the message sequence number for the given UID. It is the inverse of **imap_uid()**.

imap_search (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

This function returns an array of messages matching the given search criteria

```
array imap_search (int imap_stream, string criteria, int flags)
```

This function performs a search on the mailbox currently opened in the given imap stream. *criteria* is a string, delimited by spaces, in which the following keywords are allowed. Any multi-word arguments (eg. FROM "joey smith") must be quoted.

- ALL - return all messages matching the rest of the criteria
- ANSWERED - match messages with the \\ANSWERED flag set
- BCC "string" - match messages with "string" in the Bcc: field
- BEFORE "date" - match messages with Date: before "date"
- BODY "string" - match messages with "string" in the body of the message
- CC "string" - match messages with "string" in the Cc: field
- DELETED - match deleted messages
- FLAGGED - match messages with the \\FLAGGED (sometimes referred to as Important or Urgent) flag set
- FROM "string" - match messages with "string" in the From: field
- KEYWORD "string" - match messages with "string" as a keyword
- NEW - match new messages
- OLD - match old messages

- ON "date" - match messages with Date: matching "date"
- RECENT - match messages with the \\RECENT flag set
- SEEN - match messages that have been read (the \\SEEN flag is set)
- SINCE "date" - match messages with Date: after "date"
- SUBJECT "string" - match messages with "string" in the Subject:
- TEXT "string" - match messages with text "string"
- TO "string" - match messages with "string" in the To:
- UNANSWERED - match messages that have not been answered
- UNDELETED - match messages that are not deleted
- UNFLAGGED - match messages that are not flagged
- UNKEYWORD "string" - match messages that do not have the keyword "string"
- UNSEEN - match messages which have not been read yet

For example, to match all unanswered messages sent by Mom, you'd use: "UNANSWERED FROM mom". Searches appear to be case insensitive. This list of criteria is from a reading of the UW c-client source code and may be uncomplete or inaccurate (see also RFC2060, section 6.4.4).

Valid values for flags are SE_UID, which causes the returned array to contain UIDs instead of messages sequence numbers.

imap_last_error (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

This function returns the last IMAP error (if any) that occurred during this page request

```
string imap_last_error (void)
```

This function returns the full text of the last IMAP error message that occurred on the current page. The error stack is untouched; calling **imap_last_error()** subsequently, with no intervening errors, will return the same error.

imap_errors (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

This function returns all of the IMAP errors (if any) that have occurred during this page request or since the error stack was reset.

```
array imap_errors (void)
```

This function returns an array of all of the IMAP error messages generated since the last **imap_errors()** call, or the beginning of the page. When **imap_errors()** is called, the error stack is subsequently cleared.

imap_alerts (PHP3 >= 3.0.12, PHP4 >= 4.0b4)

This function returns all IMAP alert messages (if any) that have occurred during this page request or since the alert stack was reset

```
array imap_alerts (void)
```

This function returns an array of all of the IMAP alert messages generated since the last **imap_alerts()** call, or the beginning of the page. When **imap_alerts()** is called, the alert stack is subsequently cleared. The IMAP specification requires that these messages be passed to the user.

imap_status (PHP3 >= 3.0.4, PHP4)

This function returns status information on a mailbox other than the current one

```
object imap_status (int imap_stream, string mailbox, int options)
```

This function returns an object containing status information. Valid flags are:

- SA_MESSAGES - set status->messages to the number of messages in the mailbox
- SA_RECENT - set status->recent to the number of recent messages in the mailbox
- SA_UNSEEN - set status->unseen to the number of unseen (new) messages in the mailbox
- SA_UIDNEXT - set status->uidnext to the next uid to be used in the mailbox
- SA_UIDVALIDITY - set status->uidvalidity to a constant that changes when uids for the mailbox may no longer be valid
- SA_ALL - set all of the above

status->flags is also set, which contains a bitmask which can be checked against any of the above constants.

Example 1. imap_status() example

```
$mbox = imap_open("{your.imap.host}","username","password",OP_HALFOPEN)
      || die("can't connect: ".imap_last_error());

$status = imap_status($mbox,"{your.imap.host}INBOX",SA_ALL);
if($status) {
  print("Messages:   ". $status->messages   ."<br>\n";
  print("Recent:     ". $status->recent     ."<br>\n";
  print("Unseen:     ". $status->unseen     ."<br>\n";
  print("UIDnext:    ". $status->uidnext    ."<br>\n";
  print("UIDvalidity:". $status->uidvalidity)."<br>\n";
} else
  print "imap_status failed: ".imap_lasterror()."\n";

imap_close($mbox);
```

imap_utf7_decode (PHP3 >= 3.0.15, PHP4 >= 4.0b4)

Decodes a modified UTF-7 encoded string.

```
string imap_utf7_decode (string text)
```

Decodes modified UTF-7 *text* into 8bit data.

Returns the decoded 8bit data, or false if the input string was not valid modified UTF-7. This function is needed to decode mailbox names that contain international characters outside of the printable ASCII range. The modified UTF-7 encoding is defined in RFC 2060 (http://www.faqs.org/rfcs/rfc2060.html), section 5.1.3 (original UTF-7 was defned in RFC1642 (http://www.faqs.org/rfcs/rfc1642.html)).

imap_utf7_encode (PHP3 >= 3.0.15, PHP4 >= 4.0b4)

Converts 8bit data to modified UTF-7 text.

```
string imap_utf7_encode (string data)
```

Converts 8bit *data* to modified UTF-7 text. This is needed to encode mailbox names that contain international characters outside of the printable ASCII range. The modified UTF-7 encoding is defined in RFC 2060 (http://www.faqs.org/rfcs/rfc2060.html), section 5.1.3 (original UTF-7 was defned in RFC1642 (http://www.faqs.org/rfcs/rfc1642.html)).

Returns the modified UTF-7 text.

imap_utf8 (PHP3 >= 3.0.13, PHP4 >= 4.0RC1)

Converts text to UTF8

```
string imap_utf8 (string text)
```

Converts the given *text* to UTF8 (as defined in RFC2044 (http://www.faqs.org/rfcs/rfc2044.html)).

imap_fetch_overview (PHP3 >= 3.0.4, PHP4)

Read an overview of the information in the headers of the given message

```
array imap_fetch_overview (int imap_stream, string sequence [, int flags])
```

This function fetches mail headers for the given *sequence* and returns an overview of their contents. *sequence* will contain a sequence of message indices or UIDs, if *flags* contains FT_UID. The returned value is an array of objects describing one message header each:

- subject - the messages subject
- from - who sent it
- date - when was it sent
- message_id - Message-ID
- references - is a reference to this message id
- size - size in bytes
- uid - UID the message has in the mailbox
- msgno - message sequence number in the maibox
- recent - this message is flagged as recent
- flagged - this message is flagged
- answered - this message is flagged as answered
- deleted - this message is flagged for deletion
- seen - this message is flagged as already read
- draft - this message is flagged as being a draft

Example 1. imap_fetch_overview() example

```
$mbox = imap_open("{your.imap.host:143}","username","password")
     || die("can't connect: ".imap_last_error());

$overview = imap_fetch_overview($mbox,"2,4:6",0);

if(is_array($overview)) {
        reset($overview);
        while( list($key,$val) = each($overview)) {
                print     $val->msgno
                . " - " . $val->date
                . " - " . $val->subject
                . "\n";
        }
}

imap_close($mbox);
```

imap_mail_compose (PHP3 >= 3.0.5, PHP4)

Create a MIME message based on given envelope and body sections

```
string imap_mail_compose (array envelope, array body)
```

imap_mail (PHP3 >= 3.0.14, PHP4 >= 4.0b4)

Send an email message

```
string imap_mail (string to, string subject, string message [, string
additional_headers [, string cc [, string bcc [, string rpath]]]])
```

This function is currently only available in PHP3.

XXIX. Informix functions

The Informix driver for Informix (IDS) 7.x, SE 7.x, Universal Server (IUS) 9.x and IDS 2000 is implemented in "ifx.ec" and "php3_ifx.h" in the informix extension directory. IDS 7.x support is fairly complete, with full support for BYTE and TEXT columns. IUS 9.x support is partly finished: the new data types are there, but SLOB and CLOB support is still under construction.

Configuration notes: You need a version of ESQL/C to compile the PHP Informix driver. ESQL/C versions from 7.2x on should be OK. ESQL/C is now part of the Informix Client SDK.

Make sure that the "INFORMIXDIR" variable has been set, and that $INFORMIXDIR/bin is in your PATH before you run the "configure" script.

The configure script will autodetect the libraries and include directories, if you run "configure –with_informix=yes". You can overide this detection by specifying "IFX_LIBDIR", "IFX_LIBS" and "IFX_INCDIR" in the environment. The configure script will also try to detect your Informix server version. It will set the "HAVE_IFX_IUS" conditional compilation variable if your Informix version >= 9.00.

Runtime considerations: Make sure that the Informix environment variables INFORMIXDIR and INFORMIXSERVER are available to the PHP ifx driver, and that the INFORMIX bin directory is in the PATH. Check this by running a script that contains a call to **phpinfo()()** before you start testing. The **phpinfo()()** output should list these environment variables. This is true for both CGI php and Apache mod_php. You may have to set these environment variables in your Apache startup script.

The Informix shared libraries should also be available to the loader (check LD_LINBRARY_PATH or ld.so.conf/ldconfig).

Some notes on the use of BLOBs (TEXT and BYTE columns): BLOBs are normally addressed by BLOB identifiers. Select queries return a "blob id" for every BYTE and TEXT column. You can get at the contents with "string_var = ifx_get_blob($blob_id);" if you choose to get the BLOBs in memory (with : "ifx_blobinfile(0);"). If you prefer to receive the content of BLOB columns in a file, use "ifx_blobinfile(1);", and "ifx_get_blob($blob_id);" will get you the filename. Use normal file I/O to get at the blob contents.

For insert/update queries you must create these "blob id's" yourself with "**ifx_create_blob()**;". You then plug the blob id's into an array, and replace the blob columns with a question mark (?) in the query string. For updates/inserts, you are responsible for setting the blob contents with **ifx_update_blob()**.

The behaviour of BLOB columns can be altered by configuration variables that also can be set at runtime :

configuration variable : ifx.textasvarchar

configuration variable : ifx.byteasvarchar

runtime functions :

ifx_textasvarchar(0) : use blob id's for select queries with TEXT columns

ifx_byteasvarchar(0) : use blob id's for select queries with BYTE columns

ifx_textasvarchar(1) : return TEXT columns as if they were VARCHAR columns, so that you don't need to use blob id's for select queries.

ifx_byteasvarchar(1) : return BYTE columns as if they were VARCHAR columns, so that you don't need to use blob id's for select queries.

configuration variable : ifx.blobinfile

runtime function :

ifx_blobinfile_mode(0) : return BYTE columns in memory, the blob id lets you get at the contents.

ifx_blobinfile_mode(1) : return BYTE columns in a file, the blob id lets you get at the file name.

If you set ifx_text/byteasvarchar to 1, you can use TEXT and BYTE columns in select queries just like normal (but rather long) VARCHAR fields. Since all strings are "counted" in PHP, this remains "binary safe". It is up to you to handle this correctly. The returned data can contain anything, you are responsible for the contents.

If you set ifx_blobinfile to 1, use the file name returned by ifx_get_blob(..) to get at the blob contents. Note that in this case YOU ARE RESPONSIBLE FOR DELETING THE TEMPORARY FILES CREATED BY INFORMIX when fetching the row. Every new row fetched will create new temporary files for every BYTE column.

The location of the temporary files can be influenced by the environment variable "blobdir", default is "." (the current directory). Something like : putenv(blobdir=tmpblob"); will ease the cleaning up of temp files accidentally left behind (their names all start with "blb").

Automatically trimming "char" (SQLCHAR and SQLNCHAR) data: This can be set with the configuration variable

ifx.charasvarchar : if set to 1 trailing spaces will be automatically trimmed, to save you some "chopping".

NULL values: The configuration variable ifx.nullformat (and the runtime function **ifx_nullformat()**) when set to true will return NULL columns as the string "NULL", when set to false they return the empty string. This allows you to discriminate between NULL columns and empty columns.

ifx_connect (PHP3 >= 3.0.3, PHP4)

Open Informix server connection

```
int ifx_connect ([string database [, string userid [, string password]]])
```

Returns a connection identifier on success, or FALSE on error.

ifx_connect() establishes a connection to an Informix server. All of the arguments are optional, and if they're missing, defaults are taken from values supplied in configuration file (ifx.default_host for the host (Informix libraries will use INFORMIXSERVER environment value if not defined), ifx.default_user for user, ifx.default_password for the password (none if not defined).

In case a second call is made to **ifx_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 **ifx_close()**.

See also **ifx_pconnect()**, and **ifx_close()**.

Example 1. Connect to a Informix database

```
$conn_id = ifx_pconnect ("mydb@ol_srv1", "imyself", "mypassword");
```

ifx_pconnect (PHP3 >= 3.0.3, PHP4)

Open persistent Informix connection

```
int ifx_pconnect ([string database [, string userid [, string password]]])
```

Returns: A positive Informix persistent link identifier on success, or false on error

ifx_pconnect() acts very much like **ifx_connect()** with two major differences.

This function behaves exactly like **ifx_connect()** when PHP is not running as an Apache module. 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 (**ifx_close()** will not close links established by **ifx_pconnect()**).

This type of links is therefore called 'persistent'.

See also: **ifx_connect()**.

ifx_close (PHP3 >= 3.0.3, PHP4)

Close Informix connection

```
int ifx_close ([int link_identifier])
```

Returns: always true.

ifx_close() closes the link to an Informix 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.

ifx_close() will not close persistent links generated by **ifx_pconnect()**.

See also: **ifx_connect()**, and **ifx_pconnect()**.

Example 1. Closing a Informix connection

```
$conn_id = ifx_connect ("mydb@ol_srv", "itsme", "mypassword");
... some queries and stuff ...
ifx_close($conn_id);
```

ifx_query (PHP3 >= 3.0.3, PHP4)

Send Informix query

```
int ifx_query (string query [, int link_identifier [, int cursor_type [, mixed
blobidarray]]])
```

Returns: A positive Informix result identifier on success, or false on error.

A "result_id" resource used by other functions to retrieve the query results. Sets "affected_rows" for retrieval by the **ifx_affected_rows()** function.

ifx_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 **ifx_connect()** was called, and use it.

Executes *query* on connection *conn_id*. For "select-type" queries a cursor is declared and opened. The optional *cursor_type* parameter allows you to make this a "scroll" and/or "hold" cursor. It's a bitmask and can be either IFX_SCROLL, IFX_HOLD, or both or'ed together. Non-select queries are "execute immediate". IFX_SCROLL and IFX_HOLD are symbolic constants and as such shouldn't be between quotes. I you omit this parameter the cursor is a normal sequential cursor.

For either query type the number of (estimated or real) affected rows is saved for retrieval by **ifx_affected_rows()**.

If you have BLOB (BYTE or TEXT) columns in an update query, you can add a *blobidarray* parameter containing the corresponding "blob ids", and you should replace those columns with a "?" in the query text.

If the contents of the TEXT (or BYTE) column allow it, you can also use "ifx_textasvarchar(1)" and "ifx_byteasvarchar(1)". This allows you to treat TEXT (or BYTE) columns just as if they were ordinary (but long) VARCHAR columns for select queries, and you don't need to bother with blob id's.

With ifx_textasvarchar(0) or ifx_byteasvarchar(0) (the default situation), select queries will return BLOB columns as blob id's (integer value). You can get the value of the blob as a string or file with the blob functions (see below).

See also: **ifx_connect()**.

Example 1. Show all rows of the "orders" table as a html table

```
ifx_textasvarchar(1);      // use "text mode" for blobs
$res_id = ifx_query("select * from orders", $conn_id);
if (! $res_id) {
    printf("Can't select orders : %s\n<br>%s<br>\n", ifx_error());
    ifx_errormsg();
    die;
}
ifx_htmltbl_result($res_id, "border=\"1\"");
ifx_free_result($res_id);
```

Example 2. Insert some values into the "catalog" table

```
                      // create blob id's for a byte and text column
$textid = ifx_create_blob(0, 0, "Text column in memory");
$byteid = ifx_create_blob(1, 0, "Byte column in memory");
                      // store blob id's in a blobid array
$blobidarray[] = $textid;
$blobidarray[] = $byteid;
                      // launch query
$query = "insert into catalog (stock_num, manu_code, " .
         "cat_descr,cat_picture) values(1,'HRO',?,?)";
$res_id = ifx_query($query, $conn_id, $blobidarray);
if (! $res_id) {
  ... error ...
}
                     // free result id
ifx_free_result($res_id);
```

ifx_prepare (PHP3 >= 3.0.4, PHP4)

Prepare an SQL-statement for execution

```
int ifx_prepare (string query, int conn_id [, int cursor_def, mixed
blobidarray])
```

Returns a integer *result_id* for use by **ifx_do()**. Sets *affected_rows* for retrieval by the **ifx_affected_rows()** function.

Prepares *query* on connection *conn_id*. For "select-type" queries a cursor is declared and opened. The optional *cursor_type* parameter allows you to make this a "scroll" and/or "hold" cursor. It's a bitmask and can be either IFX_SCROLL, IFX_HOLD, or both or'ed together.

For either query type the estimated number of affected rows is saved for retrieval by **ifx_affected_rows()**.

If you have BLOB (BYTE or TEXT) columns in the query, you can add a *blobidarray* parameter containing the corresponding "blob ids", and you should replace those columns with a "?" in the query text.

If the contents of the TEXT (or BYTE) column allow it, you can also use "ifx_textasvarchar(1)" and "ifx_byteasvarchar(1)". This allows you to treat TEXT (or BYTE) columns just as if they were ordinary (but long) VARCHAR columns for select queries, and you don't need to bother with blob id's.

With ifx_textasvarchar(0) or ifx_byteasvarchar(0) (the default situation), select queries will return BLOB columns as blob id's (integer value). You can get the value of the blob as a string or file with the blob functions (see below).

See also: **ifx_do()**.

ifx_do (PHP3 >= 3.0.4, PHP4)

Execute a previously prepared SQL-statement

```
int ifx_do (int result_id)
```

Returns TRUE on success, FALSE on error.

Executes a previously prepared query or opens a cursor for it.

Does NOT free *result_id* on error.

Also sets the real number of **ifx_affected_rows()** for non-select statements for retrieval by **ifx_affected_rows()**

See also: **ifx_prepare()**. There is a example.

ifx_error (PHP3 >= 3.0.3, PHP4)

Returns error code of last Informix call

```
string ifx_error(void);
```

The Informix error codes (SQLSTATE & SQLCODE) formatted as follows :

x [SQLSTATE = aa bbb SQLCODE=cccc]

where x = space : no error

E : error

N : no more data

W : warning

? : undefined

If the "x" character is anything other than space, SQLSTATE and SQLCODE describe the error in more detail.

See the Informix manual for the description of SQLSTATE and SQLCODE

Returns in a string one character describing the general results of a statement and both SQLSTATE and SQLCODE associated with the most recent SQL statement executed. The format of the string is "(char) [SQLSTATE=(two digits) (three digits) SQLCODE=(one digit)]". The first character can be ' ' (space)

(success), 'W' (the statement caused some warning), 'E' (an error happened when executing the statement) or 'N' (the statement didn't return any data).

See also: **ifx_errormsg()**

ifx_errormsg (PHP3 >= 3.0.4, PHP4)

Returns error message of last Informix call

```
string ifx_errormsg ([int errorcode])
```

Returns the Informix error message associated with the most recent Informix error, or, when the optional "*errorcode*" param is present, the error message corresponding to "*errorcode*".

See also: **ifx_error()**

```
printf("%s\n<br>", ifx_errormsg(-201));
```

ifx_affected_rows (PHP3 >= 3.0.3, PHP4)

Get number of rows affected by a query

```
int ifx_affected_rows (int result_id)
```

result_id is a valid result id returned by **ifx_query()** or **ifx_prepare()**.

Returns the number of rows affected by a query associated with *result_id*.

For inserts, updates and deletes the number is the real number (sqlerrd[2]) of affected rows. For selects it is an estimate (sqlerrd[0]). Don't rely on it. The database server can never return the actual number of rows that will be returned by a SELECT because it has not even begun fetching them at this stage (just after the "PREPARE" when the optimizer has determined the query plan).

Useful after **ifx_prepare()** to limit queries to reasonable result sets.

See also: **ifx_num_rows()**

Example 1. Informix affected rows

```
$rid = ifx_prepare ("select * from emp
                     where name like " . $name, $connid);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
```

ifx_getsqlca (PHP3 >= 3.0.8, PHP4)

Get the contents of sqlca.sqlerrd[0..5] after a query

```
array ifx_getsqlca (int result_id)
```

result_id is a valid result id returned by **ifx_query()** or **ifx_prepare()**.

Returns a pseudo-row (assiociative array) with sqlca.sqlerrd[0] ... sqlca.sqlerrd[5] after the query associated with *result_id*.

For inserts, updates and deletes the values returned are those as set by the server after executing the query. This gives access to the number of affected rows and the serial insert value. For SELECTs the values are those saved after the PREPARE statement. This gives access to the *estimated* number of affected rows. The use of this function saves the overhead of executing a "select dbinfo('sqlca.sqlerrdx')" query, as it retrieves the values that were saved by the ifx driver at the appropriate moment.

Example 1. Retrieve Informix sqlca.sqlerrd[x] values

```
/* assume the first column of 'sometable' is a serial */
$qid = ifx_query("insert into sometable
                  values (0, '2nd column', 'another column' ", $connid);
if (! $qid) {
    ... error ...
}
$sqlca = ifx_getsqlca ($qid);
$serial_value = $sqlca["sqlerrd1"];
echo "The serial value of the inserted row is : " . $serial_value<br>\n";
```

ifx_fetch_row (PHP3 >= 3.0.3, PHP4)

Get row as enumerated array

```
array ifx_fetch_row (int result_id [, mixed position])
```

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

Blob columns are returned as integer blob id values for use in **ifx_get_blob()** unless you have used ifx_textasvarchar(1) or ifx_byteasvarchar(1), in which case blobs are returned as string values. Returns FALSE on error

result_id is a valid resultid returned by **ifx_query()** or **ifx_prepare()** (select type queries only!).

position is an optional parameter for a "fetch" operation on "scroll" cursors: "NEXT", "PREVIOUS", "CURRENT", "FIRST", "LAST" or a number. If you specify a number, an "absolute" row fetch is executed. This parameter is optional, and only valid for SCROLL cursors.

ifx_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, with the column name as key.

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

Example 1. Informix fetch rows

```
$rid = ifx_prepare ("select * from emp where name like " . $name,
                    $connid, IFX_SCROLL);
if (! $rid) {
    ... error ...
}
$rowcount = ifx_affected_rows($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
if (! ifx_do ($rid)) {
   ... error ...
}
$row = ifx_fetch_row ($rid, "NEXT");
while (is_array($row)) {
    for(reset($row); $fieldname=key($row); next($row)) {
        $fieldvalue = $row[$fieldname];
        printf ("%s = %s,", $fieldname, $fieldvalue);
    }
    printf("\n<br>");
    $row = ifx_fetch_row ($rid, "NEXT");
}
ifx_free_result ($rid);
```

ifx_htmltbl_result (PHP3 >= 3.0.3, PHP4)

Formats all rows of a query into a HTML table

```
int ifx_htmltbl_result (int result_id [, string html_table_options])
```

Returns the number of rows fetched or FALSE on error.

Formats all rows of the *result_id* query into a html table. The optional second argument is a string of <table> tag options

Example 1. Informix results as HTML table

```
$rid = ifx_prepare ("select * from emp where name like " . $name,
                    $connid, IFX_SCROLL);
if (! $rid) {
   ... error ...
}
$rowcount = ifx_affected_rows ($rid);
if ($rowcount > 1000) {
    printf ("Too many rows in result set (%d)\n<br>", $rowcount);
    die ("Please restrict your query<br>\n");
}
```

```
if (! ifx_do($rid) {
  ... error ...
}

ifx_htmltbl_result ($rid, "border=\"2\"");

ifx_free_result($rid);
```

ifx_fieldtypes (PHP3 >= 3.0.3, PHP4)

List of Informix SQL fields

```
array ifx_fieldtypes (int result_id)
```

Returns an associative array with fieldnames as key and the SQL fieldtypes as data for query with *result_id*. Returns FALSE on error.

Example 1. Fielnames and SQL fieldtypes

```
$types = ifx_fieldtypes ($resultid);
if (! isset ($types)) {
  ... error ...
}
for ($i = 0; $i < count($types); $i++) {
    $fname = key($types);
    printf("%s :\t type =  %s\n", $fname, $types[$fname]);
    next($types);
}
```

ifx_fieldproperties (PHP3 >= 3.0.3, PHP4)

List of SQL fieldproperties

```
array ifx_fieldproperties (int result_id)
```

Returns an associative array with fieldnames as key and the SQL fieldproperties as data for a query with *result_id*. Returns FALSE on error.

Returns the Informix SQL fieldproperies of every field in the query as an associative array. Properties are encoded as: "SQLTYPE;length;precision;scale;ISNULLABLE" where SQLTYPE = the Informix type like "SQLVCHAR" etc. and ISNULLABLE = "Y" or "N".

Example 1. Informix SQL fieldproperties

```
$properties = ifx_fieldtypes ($resultid);
if (! isset($properties)) {
```

```
  ... error ...
}
for ($i = 0; $i < count($properties); $i++) {
    $fname = key ($properties);
    printf ("%s:\t type =  %s\n", $fname, $properties[$fname]);
    next ($properties);
}
```

ifx_num_fields (PHP3 >= 3.0.3, PHP4)

Returns the number of columns in the query

```
int ifx_num_fields (int result_id)
```

Returns the number of columns in query for *result_id* or FALSE on error

After preparing or executing a query, this call gives you the number of columns in the query.

ifx_num_rows (PHP3 >= 3.0.3, PHP4)

Count the rows already fetched a query

```
int ifx_num_rows (int result_id)
```

Gives the number of rows fetched so far for a query with *result_id* after a **ifx_query()** or **ifx_do()** query.

ifx_free_result (PHP3 >= 3.0.3, PHP4)

Releases resources for the query

```
int ifx_free_result (int result_id)
```

Releases resources for the query associated with *result_id*. Returns FALSE on error.

ifx_create_char (PHP3 >= 3.0.6, PHP4)

Creates an char object

```
int ifx_create_char (string param)
```

Creates an char object. *param* should be the char content.

ifx_free_char (PHP3 >= 3.0.6, PHP4)

Deletes the char object

```
int ifx_free_char (int bid)
```

Deletes the charobject for the given char object-id *bid*. Returns FALSE on error otherwise TRUE.

ifx_update_char (PHP3 >= 3.0.6, PHP4)

Updates the content of the char object

```
int ifx_update_char (int bid, string content)
```

Updates the content of the char object for the given char object *bid*. *content* is a string with new data. Returns FALSE on error otherwise TRUE.

ifx_get_char (PHP3 >= 3.0.6, PHP4)

Return the content of the char object

```
int ifx_get_char (int bid)
```

Returns the content of the char object for the given char object-id *bid*.

ifx_create_blob (PHP3 >= 3.0.4, PHP4)

Creates an blob object

```
int ifx_create_blob (int type, int mode, string param)
```

Creates an blob object.

type: 1 = TEXT, 0 = BYTE

mode: 0 = blob-object holds the content in memory, 1 = blob-object holds the content in file.

param: if mode = 0: pointer to the content, if mode = 1: pointer to the filestring.

Return FALSE on error, otherwise the new blob object-id.

ifx_copy_blob (PHP3 >= 3.0.4, PHP4)

Duplicates the given blob object

```
int ifx_copy_blob (int bid)
```

Duplicates the given blob object. *bid* is the ID of the blob object.

Returns FALSE on error otherwise the new blob object-id.

ifx_free_blob (PHP3 >= 3.0.4, PHP4)

Deletes the blob object

```
int ifx_free_blob (int bid)
```

Deletes the blobobject for the given blob object-id *bid*. Returns FALSE on error otherwise TRUE.

ifx_get_blob (PHP3 >= 3.0.4, PHP4)

Return the content of a blob object

```
int ifx_get_blob (int bid)
```

Returns the content of the blob object for the given blob object-id *bid*.

ifx_update_blob (PHP3 >= 3.0.4, PHP4)

Updates the content of the blob object

```
ifx_update_blob (int bid, string content)
```

Updates the content of the blob object for the given blob object *bid*. *content* is a string with new data. Returns FALSE on error otherwise TRUE.

ifx_blobinfile_mode (PHP3 >= 3.0.4, PHP4)

Set the default blob mode for all select queries

```
void ifx_blobinfile_mode (int mode)
```

Set the default blob mode for all select queries. Mode "0" means save Byte-Blobs in memory, and mode "1" means save Byte-Blobs in a file.

ifx_textasvarchar (PHP3 >= 3.0.4, PHP4)

Set the default text mode

```
void ifx_textasvarchar (int mode)
```

Sets the default text mode for all select-queries. Mode "0" will return a blob id, and mode "1" will return a varchar with text content.

ifx_byteasvarchar (PHP3 >= 3.0.4, PHP4)

Set the default byte mode

```
void ifx_byteasvarchar (int mode)
```

Sets the default byte mode for all select-queries. Mode "0" will return a blob id, and mode "1" will return a varchar with text content.

ifx_nullformat (PHP3 >= 3.0.4, PHP4)

Sets the default return value on a fetch row

```
void ifx_nullformat (int mode)
```

Sets the default return value of a NULL-value on a fetch row. Mode "0" returns "", and mode "1" returns "NULL".

ifxus_create_slob (PHP3 >= 3.0.4, PHP4)

Creates an slob object and opens it

```
int ifxus_create_slob (int mode)
```

Creates an slob object and opens it. Modes: 1 = LO_RDONLY, 2 = LO_WRONLY, 4 = LO_APPEND, 8 = LO_RDWR, 16 = LO_BUFFER, 32 = LO_NOBUFFER -> or-mask. You can also use constants named IFX_LO_RDONLY, IFX_LO_WRONLY etc. Return FALSE on error otherwise the new slob object-id.

ifx_free_slob (unknown)

Deletes the slob object

```
int ifxus_free_slob (int bid)
```

Deletes the slob object. *bid* is the Id of the slob object. Returns FALSE on error otherwise TRUE.

ifxus_close_slob (PHP3 >= 3.0.4, PHP4)

Deletes the slob object

```
int ifxus_close_slob (int bid)
```

Deletes the slob object on the given slob object-id *bid*. Return FALSE on error otherwise TRUE.

ifxus_open_slob (PHP3 >= 3.0.4, PHP4)

Opens an slob object

```
int ifxus_open_slob (long bid, int mode)
```

Opens an slob object. *bid* should be an existing slob id. Modes: 1 = LO_RDONLY, 2 = LO_WRONLY, 4 = LO_APPEND, 8 = LO_RDWR, 16 = LO_BUFFER, 32 = LO_NOBUFFER -> or-mask. Returns FALSE on error otherwise the new slob object-id.

ifxus_tell_slob (PHP3 >= 3.0.4, PHP4)

Returns the current file or seek position

```
int ifxus_tell_slob (long bid)
```

Returns the current file or seek position of an open slob object *bid* should be an existing slob id. Return FALSE on error otherwise the seek position.

ifxus_seek_slob (PHP3 >= 3.0.4, PHP4)

Sets the current file or seek position

```
int ifxus_seek_blob (long bid, int mode, long offset)
```

Sets the current file or seek position of an open slob object. *bid* should be an existing slob id. Modes: 0 = LO_SEEK_SET, 1 = LO_SEEK_CUR, 2 = LO_SEEK_END and *offset* is an byte offset. Return FALSE on error otherwise the seek position.

ifxus_read_slob (PHP3 >= 3.0.4, PHP4)

Reads nbytes of the slob object

```
int ifxus_read_slob (long bid, long nbytes)
```

Reads nbytes of the slob object. *bid* is a existing slob id and *nbytes* is the number of bytes zu read. Return FALSE on error otherwise the string.

ifxus_write_slob (PHP3 >= 3.0.4, PHP4)

Writes a string into the slob object

```
int ifxus_write_slob (long bid, string content)
```

Writes a string into the slob object. *bid* is a existing slob id and *content* the content to write. Return FALSE on error otherwise bytes written.

XXX. InterBase functions

InterBase is a popular database put out by Borland/Inprise. More information about InterBase is available at http://www.interbase.com/. Oh, by the way, InterBase just joined the open source movement!

Note: Full support for InterBase 6 was added in PHP 4.0.

ibase_connect (PHP3 >= 3.0.6, PHP4)

Open a connection to an InterBase database

```
int ibase_connect (string database [, string username [, string password [,
string charset [, int buffers [, int dialect [, string role]]]]]])
```

Establishes a connection to an InterBase server. The *database* argument has to be a valid path to database file on the server it resides on. If the server is not local, it must be prefixed with either 'hostname:' (TCP/IP), '//hostname/' (NetBEUI) or 'hostname@' (IPX/SPX), depending on the connection protocol used. *username* and *password* can also be specified with PHP configuration directives ibase.default_user and ibase.default_password. *charset* is the default character set for a database. *buffers* is the number of database buffers to allocate for the server-side cache. If 0 or omitted, server chooses its own default. *dialect* selects the default SQL dialect for any statement executed within a connection, and it defaults to the highest one supported by client libraries.

In case a second call is made to **ibase_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 **ibase_close()**.

Example 1. Ibase_connect() example

```
<?php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);
    while ($row = ibase_fetch_object ($sth)) {
        print $row->email . "\n";
    }
    ibase_close ($dbh);
?>
```

Note: *buffers* was added in PHP4-RC2.

Note: *dialect* was added in PHP4-RC2. It is functional only with InterBase 6 and versions higher than that.

Note: *role* was added in PHP4-RC2. It is functional only with InterBase 5 and versions higher than that.

See also: **ibase_pconnect()**.

ibase_pconnect (PHP3 >= 3.0.6, PHP4)

Creates an persistent connection to an InterBase database

```
int ibase_connect (string database [, string username [, string password [,
string charset [, string role]]]])
```

ibase_pconnect() acts very much like **ibase_connect()** with two major differences. First, when connecting, the function will first try to find a (persistent) link that's already opened with the same parameters. If one is found, an identifier for it will be returned instead of opening a new connection. Second, the connection to the InterBase server will not be closed when the execution of the script ends. Instead, the link will remain open for future use (**ibase_close()** will not close links established by **ibase_pconnect()**). This type of link is therefore called 'persistent'.

See also **ibase_connect()** for the meaning of parameters passed to this function. They are exactly the same.

ibase_close (PHP3 >= 3.0.6, PHP4)

Close a connection to an InterBase database

```
int ibase_close ([int connection_id])
```

Closes the link to an InterBase database that's associated with a connection id returned from **ibase_connect()**. If the connection id is omitted, the last opened link is assumed. Default transaction on link is committed, other transactions are rolled back.

ibase_query (PHP3 >= 3.0.6, PHP4)

Execute a query on an InterBase database

```
int ibase_query ([int link_identifier, string query [, int bind_args]])
```

Performs a query on an InterBase database, returning a result identifier for use with **ibase_fetch_row()**, **ibase_fetch_object()**, **ibase_free_result()** and **ibase_free_query()**.

> **Note:** Although this function supports variable binding to parameter placeholders, there is not very much meaning using this capability with it. For real life use and an example, see **ibase_prepare()** and **ibase_execute()**.

ibase_fetch_row (PHP3 >= 3.0.6, PHP4)

Fetch a row from an InterBase database

```
array ibase_fetch_row (int result_identifier)
```

Returns the next row specified by the result identifier obtained using the **ibase_query()**.

ibase_fetch_object (PHP3 >= 3.0.7, PHP4 >= 4.0RC1)

Get an object from a InterBase database

```
object ibase_fetch_object (int result_id)
```

Fetches a row as a pseudo-object from a *result_id* obtained either by **ibase_query()** or **ibase_execute()**.

```
<php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);
    while ($row = ibase_fetch_object ($sth)) {
        print $row->email . "\n";
    }
    ibase_close ($dbh);
?>
```

See also **ibase_fetch_row()**.

ibase_free_result (PHP3 >= 3.0.6, PHP4)

Free a result set

```
int ibase_free_result (int result_identifier)
```

Free's a result set the has been created by **ibase_query()**.

ibase_prepare (PHP3 >= 3.0.6, PHP4)

Prepare a query for later binding of parameter placeholders and execution

```
int ibase_prepare ([int link_identifier, string query])
```

Prepare a query for later binding of parameter placeholders and execution (via **ibase_execute()**).

ibase_execute (PHP3 >= 3.0.6, PHP4)

Execute a previously prepared query

```
int ibase_execute (int query [, int bind_args])
```

Execute a query prepared by **ibase_prepare()**. This is a lot more effective than using **ibase_query()** if you are repeating a same kind of query several times with only some parameters changing.

```
<?php
    $updates = array(
        1 => 'Eric',
        5 => 'Filip',
        7 => 'Larry'
    );

    $query = ibase_prepare("UPDATE FOO SET BAR = ? WHERE BAZ = ?");

    while (list($baz, $bar) = each($updates)) {
        ibase_execute($query, $bar, $baz);
    }
?>
```

ibase_free_query (PHP3 >= 3.0.6, PHP4)

Free memory allocated by a prepared query

```
int ibase_free_query (int query)
```

Free a query prepared by **ibase_prepare()**.

ibase_timefmt (PHP3 >= 3.0.6, PHP4)

Sets the format of timestamp, date and time type columns returned from queries

```
int ibase_timefmt (string format [, int columntype])
```

Sets the format of timestamp, date or time type columns returned from queries. Internally, the columns are formatted by c-function strftime(), so refer to it's documentation regarding to the format of the string. *columntype* is one of the constants IBASE_TIMESTAMP, IBASE_DATE and IBASE_TIME. If omitted, defaults to IBASE_TIMESTAMP for backwards compatibility.

```
<?php
    // InterBase 6 TIME-type columns will be returned in
    // the form '05 hours 37 minutes'.
    ibase_timefmt("%H hours %M minutes", IBASE_TIME);
?>
```

You can also set defaults for these formats with PHP configuration directives ibase.timestampformat, ibase.dateformat and ibase.timeformat.

Note: *columntype* was added in PHP 4.0. It has any meaning only with InterBase version 6 and higher.

Note: A backwards incompatible change happened in PHP 4.0 when PHP configuration directive ibase.timeformat was renamed to ibase.timestampformat and directives ibase.dateformat and ibase.timeformat were added, so that the names would match better their functionality.

ibase_num_fields (PHP3 >= 3.0.7, PHP4 >= 4.0RC1)

Get the number of fields in a result set

```
int ibase_num_fields (int result_id)
```

Returns an integer containing the number of fields in a result set.

```
<?php
    $dbh = ibase_connect ($host, $username, $password);
    $stmt = 'SELECT * FROM tblname';
    $sth = ibase_query ($dbh, $stmt);

    if (ibase_num_fields($sth) > 0) {
        while ($row = ibase_fetch_object ($sth)) {
            print $row->email . "\n";
        }
    } else {
        die ("No Results were found for your query");
    }

    ibase_close ($dbh);
?>
```

See also: **ibase_field_info()**.

Note: Ibase_num_fields() is currently not functional in PHP4.

XXXI. LDAP functions

Introduction to LDAP

LDAP is the Lightweight Directory Access Protocol, and is a protocol used to access "Directory Servers". The Directory is a special kind of database that holds information in a tree structure.

The concept is similar to your hard disk directory structure, except that in this context, the root directory is "The world" and the first level subdirectories are "countries". Lower levels of the directory structure contain entries for companies, organisations or places, while yet lower still we find directory entries for people, and perhaps equipment or documents.

To refer to a file in a subdirectory on your hard disk, you might use something like

```
/usr/local/myapp/docs
```

The forwards slash marks each division in the reference, and the sequence is read from left to right.

The equivalent to the fully qualified file reference in LDAP is the "distinguished name", referred to simply as "dn". An example dn might be.

```
cn=John Smith,ou=Accounts,o=My Company,c=US
```

The comma marks each division in the reference, and the sequence is read from right to left. You would read this dn as ..

```
country = US
organization = My Company
organizationalUnit = Accounts
commonName = John Smith
```

In the same way as there are no hard rules about how you organise the directory structure of a hard disk, a directory server manager can set up any structure that is meaningful for the purpose. However, there are some conventions that are used. The message is that you can not write code to access a directory server unless you know something about its structure, any more than you can use a database without some knowledge of what is available.

Complete code example

Retrieve information for all entries where the surname starts with "S" from a directory server, displaying an extract with name and email address.

Example 1. LDAP search example

```
<?php
// basic sequence with LDAP is connect, bind, search, interpret search
// result, close connection

echo "<h3>LDAP query test</h3>";
echo "Connecting ...";
$ds=ldap_connect("localhost");  // must be a valid LDAP server!
echo "connect result is ".$ds."<p>";

if ($ds) {
```

```
    echo "Binding ...";
    $r=ldap_bind($ds);     // this is an "anonymous" bind, typically
                           // read-only access echo "Bind result is
    echo "Bind result is ".$r."<p>";

    echo "Searching for (sn=S*) ...";
    // Search surname entry
    $sr=ldap_search($ds,"o=My Company, c=US", "sn=S*");
    echo "Search result is ".$sr."<p>";

    echo "Number of entires returned is ".ldap_count_entries($ds,$sr)."<p>";

    echo "Getting entries ...<p>";
    $info = ldap_get_entries($ds, $sr);
    echo "Data for ".$info["count"]." items returned:<p>";

    for ($i=0; $i<$info["count"]; $i++) {
        echo "dn is: ". $info[$i]["dn"] ."<br>";
        echo "first cn entry is: ". $info[$i]["cn"][0] ."<br>";
        echo "first email entry is: ". $info[$i]["mail"][0] ."<p>";
    }

    echo "Closing connection";
    ldap_close($ds);

} else {
    echo "<h4>Unable to connect to LDAP server</h4>";
}
?>
```

Using the PHP LDAP calls

You will need to get and compile LDAP client libraries from either the University of Michigan ldap-3.3 package or the Netscape Directory SDK. You will also need to recompile PHP with LDAP support enabled before PHP's LDAP calls will work.

Before you can use the LDAP calls you will need to know ..

- The name or address of the directory server you will use
- The "base dn" of the server (the part of the world directory that is held on this server, which could be "o=My Company,c=US")
- Whether you need a password to access the server (many servers will provide read access for an "anonymous bind" but require a password for anything else)

The typical sequence of LDAP calls you will make in an application will follow this pattern:

```
ldap_connect()    // establish connection to server
  |
ldap_bind()       // anonymous or authenticated "login"
  |
do something like search or update the directory
and display the results
  |
ldap_close()      // "logout"
```

More Information

Lots of information about LDAP can be found at

- Netscape (http://developer.netscape.com/tech/directory/)
- University of Michigan (http://www.umich.edu/~dirsvcs/ldap/index.html)
- OpenLDAP Project (http://www.openldap.com/)
- LDAP World (http://elvira.innosoft.com/ldapworld)

The Netscape SDK contains a helpful Programmer's Guide in .html format.

ldap_add (PHP3 , PHP4)

Add entries to LDAP directory

```
int ldap_add (int link_identifier, string dn, array entry)
```

returns true on success and false on error.

The **ldap_add()** function is used to add entries in the LDAP directory. The DN of the entry to be added is specified by dn. Array entry specifies the information about the entry. The values in the entries are indexed by individual attributes. In case of multiple values for an attribute, they are indexed using integers starting with 0.

```
entry["attribute1"] = value
entry["attribute2"][0] = value1
entry["attribute2"][1] = value2
```

Example 1. Complete example with authenticated bind

```
<?php
$ds=ldap_connect("localhost");  // assuming the LDAP server is on this host

if ($ds) {
    // bind with appropriate dn to give update access
    $r=ldap_bind($ds,"cn=root, o=My Company, c=US", "secret");

    // prepare data
    $info["cn"]="John Jones";
    $info["sn"]="Jones";
    $info["mail"]="jonj@here.and.now";
    $info["objectclass"]="person";

    // add data to directory
    $r=ldap_add($ds, "cn=John Jones, o=My Company, c=US", $info);

    ldap_close($ds);
} else {
    echo "Unable to connect to LDAP server";
}
?>
```

ldap_mod_add (PHP3 >= 3.0.8, PHP4)

Add attribute values to current attributes

```
int ldap_mod_add (int link_identifier, string dn, array entry)
```

returns true on success and false on error.

This function adds attribute(s) to the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level additions are done by the **ldap_add()** function.

ldap_mod_del (PHP3 >= 3.0.8, PHP4)

Delete attribute values from current attributes

```
int ldap_mod_del (int link_identifier, string dn, array entry)
```

returns true on success and false on error.

This function removes attribute(s) from the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level deletions are done by the **ldap_del()** function.

ldap_mod_replace (PHP3 >= 3.0.8, PHP4)

Replace attribute values with new ones

```
int ldap_mod_replace (int link_identifier, string dn, array entry)
```

returns true on success and false on error.

This function replaces attribute(s) from the specified dn. It performs the modification at the attribute level as opposed to the object level. Object-level modifications are done by the **ldap_modify()** function.

ldap_bind (PHP3 , PHP4)

Bind to LDAP directory

```
int ldap_bind (int link_identifier [, string bind_rdn [, string bind_password]])
```

Binds to the LDAP directory with specified RDN and password. Returns true on success and false on error.

ldap_bind() does a bind operation on the directory. bind_rdn and bind_password are optional. If not specified, anonymous bind is attempted.

ldap_close (PHP3 , PHP4)

Close link to LDAP server

```
int ldap_close (int link_identifier)
```

Returns true on success, false on error.

ldap_close() closes the link to the LDAP server that's associated with the specified *link_identifier*.

This call is internally identical to **ldap_unbind()**. The LDAP API uses the call **ldap_unbind()**, so perhaps you should use this in preference to **ldap_close()**.

ldap_connect (PHP3 , PHP4)

Connect to an LDAP server

```
int ldap_connect ([string hostname [, int port]])
```

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

ldap_connect() establishes a connection to a LDAP server on a specified *hostname* and *port*. Both the arguments are optional. If no arguments are specified then the link identifier of the already opened link will be returned. If only *hostname* is specified, then the port defaults to 389.

ldap_count_entries (PHP3 , PHP4)

Count the number of entries in a search

```
int ldap_count_entries (int link_identifier, int result_identifier)
```

Returns number of entries in the result or false on error.

ldap_count_entries() returns the number of entries stored in the result of previous search operations. *result_identifier* identifies the internal ldap result.

ldap_delete (PHP3 , PHP4)

Delete an entry from a directory

```
int ldap_delete (int link_identifier, string dn)
```

Returns true on success and false on error.

ldap_delete() function delete a particular entry in LDAP directory specified by dn.

ldap_dn2ufn (PHP3 , PHP4)

Convert DN to User Friendly Naming format

```
string ldap_dn2ufn (string dn)
```

ldap_dn2ufn() function is used to turn a DN into a more user-friendly form, stripping off type names.

ldap_explode_dn (PHP3 , PHP4)

Splits DN into its component parts

```
array ldap_explode_dn (string dn, int with_attrib)
```

ldap_explode_dn() function is used to split the a DN returned by **ldap_get_dn()** and breaks it up into its component parts. Each part is known as Relative Distinguished Name, or RDN. **ldap_explode_dn()** returns an array of all those components. *with_attrib* is used to request if the RDNs are returned with only values or their attributes as well. To get RDNs with the attributes (i.e. in attribute=value format) set *with_attrib* to 0 and to get only values set it to 1.

ldap_first_attribute (PHP3 , PHP4)

Return first attribute

```
string ldap_first_attribute (int link_identifier, int result_entry_identifier,
int ber_identifier)
```

Returns the first attribute in the entry on success and false on error.

Similar to reading entries, attributes are also read one by one from a particular entry. **ldap_first_attribute()** returns the first attribute in the entry pointed by the entry identifier. Remaining attributes are retrieved by calling **ldap_next_attribute()** successively. *ber_identifier* is the identifier to internal memory location pointer. It is passed by reference. The same *ber_identifier* is passed to the **ldap_next_attribute()** function, which modifies that pointer.

see also **ldap_get_attributes()**

ldap_first_entry (PHP3 , PHP4)

Return first result id

```
int ldap_first_entry (int link_identifier, int result_identifier)
```

Returns the result entry identifier for the first entry on success and false on error.

Entries in the LDAP result are read sequentially using the **ldap_first_entry()** and **ldap_next_entry()** functions. **ldap_first_entry()** returns the entry identifier for first entry in the result. This entry identifier is then supplied to **lap_next_entry()** routine to get successive entries from the result.

see also **ldap_get_entries()**.

ldap_free_result (PHP3 , PHP4)

Free result memory

```
int ldap_free_result (int result_identifier)
```

Returns true on success and false on error.

ldap_free_result() frees up the memory allocated internally to store the result and pointed by the *result_identifier*. All result memory will be automatically freed when the script terminates.

Typically all the memory allocated for the ldap result gets freed at the end of the script. In case the script is making successive searches which return large result sets, **ldap_free_result()** could be called to keep the runtime memory usage by the script low.

ldap_get_attributes (PHP3 , PHP4)

Get attributes from a search result entry

```
array ldap_get_attributes (int link_identifier, int result_entry_identifier)
```

Returns a complete entry information in a multi-dimensional array on success and false on error.

ldap_get_attributes() function is used to simplify reading the attributes and values from an entry in the search result. The return value is a multi-dimensional array of attributes and values.

Having located a specific entry in the directory, you can find out what information is held for that entry by using this call. You would use this call for an application which "browses" directory entries and/or where you do not know the structure of the directory entries. In many applications you will be searching for a specific attribute such as an email address or a surname, and won't care what other data is held.

return_value["count"] = number of attributes in the entry
return_value[0] = first attribute
return_value[n] = nth attribute

return_value["attribute"]["count"] = number of values for attribute
return_value["attribute"][0] = first value of the attribute
return_value["attribute"][i] = ith value of the attribute

Example 1. Show the list of attributes held for a particular directory entry

```
// $ds is the link identifier for the directory

// $sr is a valid search result from a prior call to
// one of the ldap directory search calls

$entry = ldap_first_entry($ds, $sr);

$attrs = ldap_get_attributes($ds, $entry);

echo $attrs["count"]." attributes held for this entry:<p>";

for ($i=0; $i<$attrs["count"]; $i++)
    echo $attrs[$i]."<br>";
```

see also **ldap_first_attribute()** and **ldap_next_attribute()**

ldap_get_dn (PHP3 , PHP4)

Get the DN of a result entry

```
string ldap_get_dn (int link_identifier, int result_entry_identifier)
```

Returns the DN of the result entry and false on error.

ldap_get_dn() function is used to find out the DN of an entry in the result.

ldap_get_entries (PHP3 , PHP4)

Get all result entries

```
array ldap_get_entries (int link_identifier, int result_identifier)
```

Returns a complete result information in a multi-dimenasional array on success and false on error.

ldap_get_entries() function is used to simplify reading multiple entries from the result and then reading the attributes and multiple values. The entire information is returned by one function call in a multi-dimensional array. The structure of the array is as follows.

The attribute index is converted to lowercase. (Attributes are case-insensitive for directory servers, but not when used as array indices)

return_value["count"] = number of entries in the result
return_value[0] : refers to the details of first entry

return_value[i]["dn"] = DN of the ith entry in the result

return_value[i]["count"] = number of attributes in ith entry
return_value[i][j] = jth attribute in the ith entry in the result

return_value[i]["attribute"]["count"] = number of values for
attribute in ith entry
return_value[i]["attribute"][j] = jth value of attribute in ith entry

see also **ldap_first_entry()** and **ldap_next_entry()**

ldap_get_values (PHP3 , PHP4)

Get all values from a result entry

```
array ldap_get_values (int link_identifier, int result_entry_identifier, string
attribute)
```

Returns an array of values for the attribute on success and false on error.

ldap_get_values() function is used to read all the values of the attribute in the entry in the result. entry is specified by the *result_entry_identifier*. The number of values can be found by indexing "count" in the resultant array. Individual values are accessed by integer index in the array. The first index is 0.

This call needs a *result_entry_identifier*, so needs to be preceded by one of the ldap search calls and one of the calls to get an individual entry.

You application will either be hard coded to look for certain attributes (such as "surname" or "mail") or you will have to use the **ldap_get_attributes()** call to work out what attributes exist for a given entry.

LDAP allows more than one entry for an attribute, so it can, for example, store a number of email addresses for one person's directory entry all labeled with the attribute "mail"

return_value["count"] = number of values for attribute
return_value[0] = first value of attribute
return_value[i] = ith value of attribute

Example 1. List all values of the "mail" attribute for a directory entry

```
// $ds is a valid link identifier for a directory server

// $sr is a valid search result from a prior call to
//     one of the ldap directory search calls

// $entry is a valid entry identifier from a prior call to
//        one of the calls that returns a directory entry

$values = ldap_get_values($ds, $entry,"mail");

echo $values["count"]." email addresses for this entry.<p>";

for ($i=0; $i < $values["count"]; $i++)
    echo $values[$i]."<br>";
```

ldap_get_values_len (PHP3 >= 3.0.13, PHP4 >= 4.0RC2)

Get all binary values from a result entry

```
array ldap_get_values_len (int link_identifier, int result_entry_identifier,
string attribute)
```

Returns an array of values for the attribute on success and false on error.

ldap_get_values_len() function is used to read all the values of the attribute in the entry in the result. entry is specified by the *result_entry_identifier*. The number of values can be found by indexing "count" in the resultant array. Individual values are accessed by integer index in the array. The first index is 0.

This function is used exactly like **ldap_get_values()** except that it handles binary data and not string data.

> **Note:** This function was added in 4.0.

ldap_list (PHP3 , PHP4)

Single-level search

```
int ldap_list (int link_identifier, string base_dn, string filter [, array
attributes])
```

Returns a search result identifier or false on error.

ldap_list() performs the search for a specified filter on the directory with the scope LDAP_SCOPE_ONELEVEL.

LDAP_SCOPE_ONELEVEL means that the search should only return information that is at the level immediately below the base dn given in the call. (Equivalent to typing "ls" and getting a list of files and folders in the current working directory.)

This call takes an optional fourth parameter which is an array of the attributes required. See **ldap_search()** notes.

Example 1. Produce a list of all organizational units of an organization

```
// $ds is a valid link identifier for a directory server

$basedn = "o=My Company, c=US";
$justthese = array("ou");

$sr=ldap_list($ds, $basedn, "ou=*", $justthese);

$info = ldap_get_entries($ds, $sr);

for ($i=0; $i<$info["count"]; $i++)
    echo $info[$i]["ou"][0] ;
```

ldap_modify (PHP3 , PHP4)

Modify an LDAP entry

```
int ldap_modify (int link_identifier, string dn, array entry)
```

Returns true on success and false on error.

ldap_modify() function is used to modify the existing entries in the LDAP directory. The structure of the entry is same as in **ldap_add()**.

ldap_next_attribute (PHP3 , PHP4)

Get the next attribute in result

```
string ldap_next_attribute (int link_identifier, int result_entry_identifier,
int ber_identifier)
```

Returns the next attribute in an entry on success and false on error.

ldap_next_attribute() is called to retrieve the attributes in an entry. The internal state of the pointer is maintained by the *ber_identifier*. It is passed by reference to the function. The first call to **ldap_next_attribute()** is made with the *result_entry_identifier* returned from **ldap_first_attribute()**.

see also **ldap_get_attributes()**

ldap_next_entry (PHP3 , PHP4)

Get next result entry

```
int ldap_next_entry (int link_identifier, int result_entry_identifier)
```

Returns entry identifier for the next entry in the result whose entries are being read starting with **ldap_first_entry()**. If there are no more entries in the result then it returns false.

ldap_next_entry() function is used to retrieve the entries stored in the result. Successive calls to the **ldap_next_entry()** return entries one by one till there are no more entries. The first call to **ldap_next_entry()** is made after the call to **ldap_first_entry()** with the result_identifier as returned from the **ldap_first_entry()**.

see also **ldap_get_entries()**

ldap_read (PHP3 , PHP4)

Read an entry

```
int ldap_read (int link_identifier, string base_dn, string filter [, array
attributes])
```

Returns a search result identifier or false on error.

ldap_read() performs the search for a specified filter on the directory with the scope LDAP_SCOPE_BASE. So it is equivalent to reading an entry from the directory.

An empty filter is not allowed. If you want to retrieve absolutely all information for this entry, use a filter of "objectClass=*". If you know which entry types are used on the directory server, you might use an appropriate filter such as "objectClass=inetOrgPerson".

This call takes an optional fourth parameter which is an array of the attributes required. See **ldap_search()** notes.

ldap_search (PHP3 , PHP4)

Search LDAP tree

```
int ldap_search (int link_identifier, string base_dn, string filter [, array
attributes])
```

Returns a search result identifier or false on error.

ldap_search() performs the search for a specified filter on the directory with the scope of LDAP_SCOPE_SUBTREE. This is equivalent to searching the entire directory. *base_dn* specifies the base DN for the directory.

There is a optional fourth parameter, that can be added to restrict the attributes and values returned by the server to just those required. This is much more efficient than the default action (which is to return all attributes and their associated values). The use of the fourth parameter should therefore be considered good practice.

The fourth parameter is a standard PHP string array of the required attributes, eg array("mail","sn","cn") Note that the "dn" is always returned irrespective of which attributes types are requested.

Note too that some directory server hosts will be configured to return no more than a preset number of entries. If this occurs, the server will indicate that it has only returned a partial results set.

The search filter can be simple or advanced, using boolean operators in the format described in the LDAP doumentation (see the Netscape Directory SDK (http://developer.netscape.com/tech/directory/) for full information on filters).

The example below retrieves the organizational unit, surname, given name and email address for all people in "My Company" where the surname or given name contains the substring $person. This example uses a boolean filter to tell the server to look for information in more than one attribute.

Example 1. LDAP search

```
// $ds is a valid link identifier for a directory server

// $person is all or part of a person’s name, eg "Jo"

$dn = "o=My Company, c=US";
$filter="(|(sn=$person*)(givenname=$person*))";
$justthese = array( "ou", "sn", "givenname", "mail");

$sr=ldap_search($ds, $dn, $filter, $justthese);

$info = ldap_get_entries($ds, $sr);

print $info["count"]." entries returned<p>";
```

ldap_unbind (PHP3 , PHP4)

Unbind from LDAP directory

```
int ldap_unbind (int link_identifier)
```

Returns true on success and false on error.

ldap_unbind() function unbinds from the LDAP directory.

ldap_err2str (PHP3 >= 3.0.13, PHP4 >= 4.0RC2)

Convert LDAP error number into string error message

```
string ldap_err2str (int errno)
```

returns string error message.

This function returns the string error message explaining the error number errno. While LDAP errno numbers are standardized, different libraries return different or even localized textual error messages. Never check for a specific error message text, but always use an error number to check.

See also **ldap_errno()** and **ldap_error()**.

Example 1. Enumerating all LDAP error messages

```
<?php
  for($i=0; $i<100; $i++) {
    printf("Error $i: %s<br>\n", ldap_str2err($i));
  }
?>
```

ldap_errno (PHP3 >= 3.0.12, PHP4 >= 4.0RC2)

Return the LDAP error number of the last LDAP command

```
int ldap_errno (int link_id)
```

return the LDAP error number of the last LDAP command for this link.

This function returns the standardized error number returned by the last LDAP command for the given link identifier. This number can be converted into a textual error message using **ldap_err2str()**.

Unless you lower your warning level in your php3.ini sufficiently or prefix your LDAP commands with @ (at) characters to suppress warning output, the errors generated will also show up in your HTML output.

Example 1. Generating and catching an error

```
<?php
// This example contains an error, which we will catch.
$ld = ldap_connect("localhost");
$bind = ldap_bind($ld);
// syntax error in filter expression (errno 87),
// must be "objectclass=*" to work.
$res =  @ldap_search($ld, "o=Myorg, c=DE", "objectclass");
if (!$res) {
```

```
    printf("LDAP-Errno: %s<br>\n", ldap_errno($ld));
    printf("LDAP-Error: %s<br>\n", ldap_error($ld));
    die("Argh!<br>\n");
}
$info = ldap_get_entries($ld, $res);
printf("%d matching entries.<br>\n", $info["count"]);
?>
```

see also **ldap_err2str()** and **ldap_error()**.

ldap_error (PHP3 >= 3.0.12, PHP4 >= 4.0RC2)

Return the LDAP error message of the last LDAP command

```
string ldap_error (int link_id)
```

returns string error message.

This function returns the string error message explaining the error generated by the last LDAP command for the given link identifier. While LDAP errno numbers are standardized, different libraries return different or even localized textual error messages. Never check for a specific error message text, but always use an error number to check.

Unless you lower your warning level in your `php3.ini` sufficiently or prefix your LDAP commands with `@` (at) characters to suppress warning output, the errors generated will also show up in your HTML output.

see also **ldap_err2str()** and **ldap_errno()**.

XXXII. Mail functions

The **mail()** function allows you to send mail.

mail (PHP3 , PHP4)

send mail

```
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());
```

XXXIII. 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	Der Wert π (Pi)
M_E	2.7182818284590452354	e
M_LOG2E	1.4426950408889634074	log_2 e
M_LOG10E	0.43429448190325182765	log_10 e
M_LN2	0.69314718055994530942	log_e 2
M_LN10	2.30258509299404568402	log_e 10
M_PI_2	1.57079632679489661923	pi/2
M_PI_4	0.78539816339744830962	pi/4
M_1_PI	0.31830988618379067154	1/pi
M_2_PI	0.63661977236758134308	2/pi
M_2_SQRTPI	1.12837916709551257390	2/sqrt(pi)
M_SQRT2	1.41421356237309504880	sqrt(2)
M_SQRT1_2	0.70710678118654752440	1/sqrt(2)

Only M_PI is available in PHP versions up to and including PHP4RC1. All other constants are available starting with PHP4.0.

Abs (unknown)

absolute value

```
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 (unknown)

arc cosine

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

Returns the arc cosine of arg in radians.

See also **asin()** and **atan()**.

Asin (unknown)

arc sine

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

Returns the arc sine of arg in radians.

See also **acos()** and **atan()**.

Atan (unknown)

arc tangent

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

Returns the arc tangent of arg in radians.

See also **asin()** and **acos()**.

Atan2 (unknown)

arc tangent of two variables

```
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 (PHP3 >= 3.0.6, PHP4)

convert a number between arbitrary bases

```
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 (unknown)

binary to decimal

```
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 (unknown)

round fractions up

```
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 (unknown)

cosine

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

Returns the cosine of arg in radians.

See also **sin()** and **tan()**.

DecBin (unknown)

decimal to binary

```
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 (unknown)

decimal to hexadecimal

```
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 (unknown)

decimal to octal

```
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()**.

deg2rad (PHP3 >= 3.0.4, PHP4)

Converts the number in degrees to the radian equivalent

```
double deg2rad (double number)
```

This function converts *number* from degrees to the radian equivalent.

See also **rad2deg()**.

Exp (unknown)

e to the power of...

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

Returns e raised to the power of *arg*.

See also **pow()**.

Floor (unknown)

round fractions down

```
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 (PHP3 , PHP4)

show largest possible random value

```
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 (unknown)

hexadecimal to decimal

```
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 (unknown)

natural logarithm

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

Returns the natural logarithm of arg.

Log10 (unknown)

base-10 logarithm

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

Returns the base-10 logarithm of arg.

max (PHP3 , PHP4)

find highest value

```
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 (PHP3 , PHP4)

find lowest value

```
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 (PHP3 >= 3.0.6, PHP4)

generate a better random value

```
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()**.

> **Note:** In versions before 3.0.7 the meaning of *max* was *range*. To get the same results in these versions the short example should be `mt_rand (5, 11)` to get a random number between 5 und 15.

See also **mt_srand()**, **mt_getrandmax()**, **srand()**, **rand()** and **getrandmax()**.

mt_srand (PHP3 >= 3.0.6, PHP4)

seed the better random number generator

```
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 (PHP3 >= 3.0.6, PHP4)

show largest possible random value

```
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 (PHP3 , PHP4)

format a number with grouped thousands

```
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 (unknown)

octal to decimal

```
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 (PHP3 , PHP4)

get value of pi

```
double pi (void)
```

Returns an approximation of pi.

pow (PHP3 , PHP4)

exponential expression

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

Returns base raised to the power of exp.

See also **exp()**.

rad2deg (PHP3 >= 3.0.4, PHP4)

Converts the radian number to the equivalent number in degrees

```
double rad2deg (double number)
```

This function converts *number* from radian to degrees.

See also **deg2rad()**.

rand (PHP3 , PHP4)

Generate a random value

```
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()**.

> **Note:** In versions before 3.0.7 the meaning of *max* was *range*. To get the same results in these versions the short example should be `rand (5, 11)` to get a random number between 5 und 15.

See also **srand()**, **getrandmax()**, **mt_rand()**, **mt_srand()** and **mt_getrandmax()**.

round (PHP3 , PHP4)

Rounds a float.

```
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 (unknown)

sine

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

Returns the sine of arg in radians.

See also **cos()** and **tan()**.

Sqrt (unknown)

square root

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

Returns the square root of arg.

srand (PHP3 , PHP4)

seed the random number generator

```
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 (unknown)

tangent

```
float tan (float arg)
```

Returns the tangent of arg in radians.

See also **sin()** and **cos()**.

XXXIV. 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_RECUR_NONE, MCAL_RECUR_DAILY, MCAL_RECUR_WEEKLY, MCAL_RECUR_MONTHLY_MDAY, MCAL_RECUR_MONTHLY_WDAY, MCAL_RECUR_YEARLY, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Opens up an MCAL connection

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Close an MCAL stream

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

Closes the given mcal stream.

mcal_fetch_event (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Fetches an event from the calendar stream

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return a list of events between two given datetimes

```
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_append_event (PHP4 >= 4.0RC1)

Store a new event into an MCAL calendar

```
int mcal_append_event (int mcal_stream)
```

mcal_append_event() Stores the global event into an MCAL calendar for the given stream.

Returns the uid of the newly inserted event.

mcal_store_event (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Modify an existing event in an MCAL calendar

```
int mcal_store_event (int mcal_stream)
```

mcal_store_event() Stores the modifications to the current global event for the given stream.

Returns true on success and false on error.

mcal_delete_event (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Delete an event from an MCAL calendar

```
int mcal_delete_event (int uid)
```

mcal_delete_event() deletes the calendar event specified by the uid.

Returns true.

mcal_snooze (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Turn off an alarm for an event

```
int mcal_snooze (int uid)
```

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

Returns true.

mcal_list_alarms (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

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

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Initializes a streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the category of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the title of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the description of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the start date and time of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the end date and time of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the alarm of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the class of the streams global event structure

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns if the given year is a leap year or not

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the number of days in the given month

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

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

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

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

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the day of the week of the given date

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

mcal_day_of_week() returns the day of the week of the given date

mcal_day_of_year (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the day of the year of the given date

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

mcal_day_of_year() returns the day of the year of the given date

mcal_date_compare (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Compares two dates

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns the next recurrence of the event

```
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.

mcal_event_set_recur_none (PHP3 >= 3.0.15, PHP4 >= 4.0RC1)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_none  (int stream)
```

mcal_event_set_recur_none() sets the streams global event structure to not recur (event->recur_type is set to MCAL_RECUR_NONE).

mcal_event_set_recur_daily (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_daily  (int stream, int year, int month, int day, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_weekly (int stream, int year, int month, int day, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_monthly_mday (int stream, int year, int month, int day, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_monthly_wday (int stream, int year, int month, int day, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Sets the recurrence of the streams global event structure

```
int mcal_event_set_recur_yearly (int stream, int year, int month, int day, 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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Returns an object containing the current streams event structure

```
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

XXXV. Microsoft SQL Server functions

mssql_close (PHP3 , PHP4)

Close MS SQL Server connection

```
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 (PHP3 , PHP4)

Open MS SQL server connection

```
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 (PHP3 , PHP4)

Move internal row pointer

```
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 (PHP3 , PHP4)

Fetch row as array

```
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 (PHP3 , PHP4)

Get field information

```
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 (PHP3 , PHP4)

Fetch row as object

```
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 (PHP3 , PHP4)

Get row as enumerated array

```
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_length (PHP3 >= 3.0.3, PHP4 >= 4.0b4)

Get the length of a field

```
int mssql_field_length (int result [, int offset])
```

mssql_field_name (PHP3 >= 3.0.3, PHP4 >= 4.0b4)

Get the name of a field

```
int mssql_field_name (int result [, int offset])
```

mssql_field_seek (PHP3 , PHP4)

Set field offset

```
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_field_type (PHP3 >= 3.0.3, PHP4 >= 4.0b4)

Get the type of a field

```
string mssql_field_type (int result [, int offset])
```

mssql_free_result (PHP3 , PHP4)

Free result memory

```
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_get_last_message (PHP3 , PHP4)

Returns the last message from server (over min_message_severity?)

```
string mssql_get_last_message  (void )
```

mssql_min_error_severity (PHP3 , PHP4)

Sets the lower error severity

```
void mssql_min_error_severity  (int severity)
```

mssql_min_message_severity (PHP3 , PHP4)

Sets the lower message severity

```
void mssql_min_message_severity  (int severity)
```

mssql_num_fields (PHP3 , PHP4)

Get number of fields in result

```
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()**, and **mssql_num_rows()**.

mssql_num_rows (PHP3 , PHP4)

Get number of rows in result

```
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 (PHP3 , PHP4)

Open persistent MS SQL connection

```
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 (PHP3 , PHP4)

Send MS SQL query

```
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 (PHP3 , PHP4)

Get result data

```
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 (PHP3 , PHP4)

Select MS SQL database

```
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()**

XXXVI. Miscellaneous functions

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

connection_aborted (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Returns true if client disconnected

```
int connection_aborted (void )
```

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

connection_status (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Returns connection status bitfield

```
int connection_status (void )
```

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

connection_timeout (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Return true if script timed out

```
int connection_timeout (void )
```

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

define (PHP3 , PHP4)

Defines a named constant.

```
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 (PHP3 , PHP4)

Checks whether a given named constant exists

```
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 (unknown)

Output a message and terminate the current script

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

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

Example 1. die example

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

See also **exit()**.

eval (unknown)

Evaluate a string as PHP code

```
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 (unknown)

Terminate current script

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

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

See also **die()**.

func_get_arg (PHP4 CVS only)

Return an item from the argument list

```
int func_get_arg (int arg_num)
```

Returns the argument which is at the *arg_num*'th offset into a user-defined function's argument list. Function arguments are counted starting from zero. **Func_get_arg()** will generate a warning if called from outside of a function definition.

If *arg_num* is greater than the number of arguments actually passed, a warning will be generated and **func_get_arg()** will return FALSE.

```
<?php
function foo() {
     $numargs = func_num_args();
     echo "Number of arguments: $numargs<br>\n";
     if ($numargs >= 2) {
         echo "Second argument is: " . func_get_arg (1) . "<br>\n";
     }
}

foo (1, 2, 3);
?>
```

Func_get_arg() may be used in conjunction with **func_num_args()** and **func_get_args()** to allow user-defined functions to accept variable-length argument lists.

> **Note:** This function was added in PHP 4.

func_get_args (PHP4 CVS only)

Returns an array comprising a function's argument list

```
int func_get_args (void )
```

Returns an array in which each element is the corresponding member of the current user-defined function's argument list. **Func_get_args()** will generate a warning if called from outside of a function definition.

```
<?php
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs<br>\n";
    if ($numargs >= 2) {
        echo "Second argument is: " . func_get_arg (1) . "<br>\n";
    }
    $arg_list = func_get_args();
    for ($i = 0; $i < $numargs; $i++) {
        echo "Argument $i is: " . $arg_list[$i] . "<br>\n";
    }
}

foo (1, 2, 3);
```

```
?>
```

Func_get_args() may be used in conjunction with **func_num_args()** and **func_get_arg()** to allow user-defined functions to accept variable-length argument lists.

> **Note:** This function was added in PHP 4.

func_num_args (PHP4 CVS only)

Returns the number of arguments passed to the function

```
int func_num_args (void )
```

Returns the number of arguments passed into the current user-defined function. **Func_num_args()** will generate a warning if called from outside of a function definition.

```
<?php
function foo() {
    $numargs = func_num_args();
    echo "Number of arguments: $numargs\n";
}

foo (1, 2, 3);  // Prints 'Number of arguments: 3'
?>
```

Func_num_args() may be used in conjunction with **func_get_arg()** and **func_get_args()** to allow user-defined functions to accept variable-length argument lists.

> **Note:** This function was added in PHP 4.

function_exists (PHP3 >= 3.0.7, PHP4)

Return true if the given function has been defined

```
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 (PHP3 , PHP4)

Tells what the user's browser is capable of

```
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.php3.

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

ignore_user_abort (PHP3 >= 3.0.7, PHP4 >= 4.0b4)

Set whether a client disconnect should abort script execution

```
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 (PHP3 >= 3.0.6, PHP4)

Parse a binary IPTC http://www.xe.net/iptc/ block into single tags.

```
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 (PHP3 , PHP4)

Leak memory

```
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 (PHP3 , PHP4)

Pack data into binary string.

```
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, however, there are some formatting codes that are missing such as Perl's "u" format code. 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 (PHP3 >= 3.0.4, PHP4)

Register a function for execution on shutdown

```
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 (PHP3 >= 3.0.5, PHP4)

Generates a storable representation of a value

```
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 (PHP3 , PHP4)

Delay execution

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

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

See also **usleep()**.

uniqid (PHP3 , PHP4)

Generate a unique id

```
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 (PHP3 , PHP4)

Unpack data from binary string

```
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 (PHP3 >= 3.0.5, PHP4)

Creates a PHP value from a stored representation

```
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 (PHP3 , PHP4)

Delay execution in microseconds

```
void usleep (int micro_seconds)
```

The **sleep()** function delays program execution for the given number of *micro_seconds*.
See also **sleep()**.

> **Note:** This function does not work on Windows systems.

XXXVII. mSQL functions

msql (PHP3 , PHP4)

Send mSQL query

```
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 (PHP3 >= 3.0.6, PHP4)

Returns number of affected rows

```
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 (PHP3 , PHP4)

Close mSQL connection

```
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 (PHP3 , PHP4)

Open mSQL connection

```
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 (PHP3 , PHP4)

Create mSQL database

```
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 (PHP3 , PHP4)

Create mSQL database

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

Identical to **msql_create_db()**.

msql_data_seek (PHP3 , PHP4)

Move internal row pointer

```
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 (PHP3 , PHP4)

Get current mSQL database name

```
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 (PHP3 , PHP4)

Drop (delete) mSQL database

```
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 (PHP3 , PHP4)

Drop (delete) mSQL database

See **msql_drop_db()**.

msql_error (PHP3 , PHP4)

Returns error message of last msql call

```
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 (PHP3 , PHP4)

Fetch row as array

```
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 (PHP3 , PHP4)

Get field information

```
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 (PHP3 , PHP4)

Fetch row as object

```
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 (PHP3 , PHP4)

Get row as enumerated array

```
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 (PHP3 , PHP4)

Get field name

```
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 (PHP3 , PHP4)

Set field offset

```
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 (PHP3 , PHP4)

Get table name for field

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

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

msql_fieldtype (PHP3 , PHP4)

Get field type

```
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 (PHP3 , PHP4)

Get field flags

```
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 (PHP3 , PHP4)

Get field length

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

Msql_fieldlen() returns the length of the specified field.

msql_free_result (PHP3 , PHP4)

Free result memory

```
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 (PHP3 , PHP4)

Free result memory

See **msql_free_result()**

msql_list_fields (PHP3 , PHP4)

List result fields

```
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 (PHP3 , PHP4)

List result fields

See **msql_list_fields()**.

msql_list_dbs (PHP3 , PHP4)

List mSQL databases on server

```
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 (PHP3 , PHP4)

List mSQL databases on server

See **msql_list_dbs()**.

msql_list_tables (PHP3 , PHP4)

List tables in an mSQL database

```
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 (PHP3 , PHP4)

List tables in an mSQL database

See **msql_list_tables()**.

msql_num_fields (PHP3 , PHP4)

Get number of fields in result

```
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 (PHP3 , PHP4)

Get number of rows in result

```
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 (PHP3 , PHP4)

Get number of fields in result

```
int msql_numfields (int query_identifier)
```

Identical to **msql_num_fields()**.

msql_numrows (PHP3 , PHP4)

Get number of rows in result

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

Identical to **msql_num_rows()**.

msql_pconnect (PHP3 , PHP4)

Open persistent mSQL connection

```
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 (PHP3 , PHP4)

Send mSQL query

```
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 (PHP3 , PHP4)

Make regular expression for case insensitive match

See **sql_regcase()**.

msql_result (PHP3 , PHP4)

Get result data

```
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 (PHP3 , PHP4)

Select mSQL database

```
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 (PHP3 , PHP4)

Select mSQL database

See **msql_select_db()**.

msql_tablename (PHP3 , PHP4)

Get table name of field

```
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++;
}
?>
```

XXXVIII. 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 (PHP3 , PHP4)

Get number of affected rows in previous MySQL operation

```
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_change_user (PHP3 >= 3.0.13)

Change logged in user on active connection

```
int mysql_change_user (string user, string password [, string database [, int
link_identifier]])
```

mysql_change_user() changes the logged in user on the current active connection, or, if specified on the connection given by the link identifier. If a database is specified, this will default or current database after the user has been changed. If the new user/password combination fails to be authorized the current connected user stays active.

> **Note:** This function was introduced in PHP 3.0.13 and requires MySQL 3.23.3 or higher.

mysql_close (PHP3 , PHP4)

close MySQL connection

```
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:** 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()**.

Example 1. MySQL close example

```
<?php
    $link = mysql_connect ("kraemer", "marliesle", "secret") {
        or die ("Could not connect");
    }
    print ("Connected successfully");
    mysql_close ($link);
?>
```

See also: **mysql_connect()**, and **mysql_pconnect()**.

mysql_connect (PHP3 , PHP4)

Open a connection to a MySQL Server

```
int mysql_connect ([string hostname [:port] [:/path/to/socket] [, string
username [, string password]]])
```

Returns: A positive MySQL link identifier on success, or an error message on failure.

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" was added in PHP 3.0B4.

Support for the ":/path/to/socket" was added in PHP 3.0.10.

You can suppress the error message on failure by prepending '@' to the function name.

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()**.

Example 1. MySQL connect example

```
<?php
    $link = mysql_connect ("kraemer", "marliesle", "secret") {
        or die ("Could not connect");
    }
    print ("Connected successfully");
    mysql_close ($link);
?>
```

See also **mysql_pconnect()**, and **mysql_close()**.

mysql_create_db (PHP3 , PHP4)

Create a MySQL database

```
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.

Example 1. MySQL create database example

```
<?php
    $link = mysql_pconnect ("kron", "jutta", "geheim") {
        or die ("Could not connect");
    }
    if (mysql_create_db ("my_db")) {
        print ("Database created successfully\n");
    } else {
        printf ("Error creating database: %s\n", mysql_error ());
    }
?>
```

For downwards compatibility **mysql_createdb()** can also be used.

See also: **mysql_drop_db()**.

mysql_data_seek (PHP3 , PHP4)

Move internal result pointer

```
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.

Example 1. MySQL data seek example

```
<?php
    $link = mysql_pconnect ("kron", "jutta", "geheim") {
        or die ("Could not connect");
    }

    mysql_select_db ("samp_db") {
        or die ("Could not select database");
    }

    $query = "SELECT last_name, first_name FROM friends";
    $result = mysql_query ($query) {
```

```
        or die ("Query failed");
    }

    # fetch rows in reverse order

    for ($i = mysql_num_rows ($result) - 1; $i >=0; $i-) {
        if (!mysql_data_seek ($result, $i)) {
            printf ("Cannot seek to row %d\n", $i);
            continue;
        }

        if(!($row = mysql_fetch_object ($result)))
            continue;

        printf ("%s %s<BR>\n", $row->last_name, $row->first_name);
    }

    mysql_free_result ($result);
?>
```

mysql_db_query (PHP3 , PHP4)

Send an MySQL query to MySQL

```
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 (PHP3 , PHP4)

Drop (delete) a MySQL database

```
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 (PHP3 , PHP4)

Returns the number of the error message from previous MySQL operation

```
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 (PHP3 , PHP4)

Returns the text of the error message from previous MySQL operation

```
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 (PHP3 , PHP4)

Fetch a result row as an associative array

```
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. (This feature was added in PHP 3.0.7)

For further details, see also **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 (PHP3 , PHP4)

Get column information from a result and return as an object

```
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 (PHP3 , PHP4)

Get the length of each output in a result

```
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 (PHP3 , PHP4)

Fetch a result row as an object

```
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 (PHP3 , PHP4)

Get a result row as an enumerated array

```
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 (PHP3 , PHP4)

Get the name of the specified field in a result

```
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 (PHP3 , PHP4)

Set result pointer to a specified field offset

```
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 (PHP3 , PHP4)

Get name of the table the specified field is in

```
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 (PHP3 , PHP4)

Get the type of the specified field in a result

```
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 (PHP3 , PHP4)

Get the flags associated with the specified field in a result

```
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 (PHP3 , PHP4)

Returns the length of the specified field

```
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 (PHP3 , PHP4)

Free result memory

```
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 (PHP3 , PHP4)

Get the id generated from the previous INSERT operation

```
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 (PHP3 , PHP4)

List MySQL result fields

```
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 (PHP3 , PHP4)

List databases available on on MySQL server

```
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 (PHP3 , PHP4)

List tables in a MySQL database

```
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 (PHP3 , PHP4)

Get number of fields in result

```
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 (PHP3 , PHP4)

Get number of rows in result

```
int mysql_num_rows (int result)
```

mysql_num_rows() returns the number of rows in a result set. This command is only valid for SELECT statements. To retrieve the number of rows returned from a INSERT, UPDATE or DELETE, use **mysql_affected_rows()**.

See also: **mysql_db_query()**, **mysql_query()** and, **mysql_fetch_row()**.

For downward compatibility **mysql_numrows()** can also be used.

mysql_pconnect (PHP3 , PHP4)

Open a persistent connection to a MySQL Server

```
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 (PHP3 , PHP4)

Send an SQL query to MySQL

```
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 (PHP3 , PHP4)

Get result data

```
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 (PHP3 , PHP4)

Select a MySQL database

```
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 (PHP3 , PHP4)

Get table name of field

```
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++;
}
?>
```

XXXIX. Network Functions

checkdnsrr (PHP3 , PHP4)

Check DNS records corresponding to a given Internet host name or IP address

```
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 (PHP3 , PHP4)

Close connection to system logger

```
int closelog(void);
```

Closelog() closes the descriptor being used to write to the system logger. The use of **closelog()** is optional.

debugger_off (PHP3)

Disable internal PHP debugger

```
int debugger_off(void);
```

Disables the internal PHP debugger. The debugger is still under development.

debugger_on (PHP3)

Enable internal PHP debugger

```
int debugger_on (string address)
```

Enables the internal PHP debugger, connecting it to *address*. The debugger is still under development.

fsockopen (PHP3 , PHP4)

Open Internet or Unix domain socket connection

```
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()**, and **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 (PHP3 , PHP4)

Get the Internet host name corresponding to a given IP address

```
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 (PHP3 , PHP4)

Get the IP address corresponding to a given Internet host name

```
string gethostbyname (string hostname)
```

Returns the IP address of the Internet host specified by *hostname*.

See also **gethostbyaddr()**.

gethostbynamel (PHP3 , PHP4)

Get a list of IP addresses corresponding to a given Internet host name

```
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 (PHP3 , PHP4)

Get MX records corresponding to a given Internet host name

```
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.

getprotobyname (PHP4 >= 4.0b4)

Get protocol number associated with protocol name

```
int getprotobyname (string name)
```

Getprotobyname() returns the protocol number associated with the protocol *name* as per /etc/protocols.

See also: **getprotobynumber()**.

getprotobynumber (PHP4 >= 4.0b4)

Get protocol name associated with protocol number

```
string getprotobynumber (int number)
```

Getprotobynumber() returns the protocol name associated with protocol *number* as per /etc/protocols.

See also: **getprotobyname()**.

getservbyname (PHP4 >= 4.0b4)

Get port number associated with an Internet service and protocol

```
int getservbyname (string service, string protocol)
```

Getservbyname() returns the Internet port which corresponds to *service* for the specified *protocol* as per /etc/services. *protocol* is either TCP or UDP.

See also: **getservbyport()**.

getservbyport (PHP4 >= 4.0b4)

Get Internet service which corresponds to port and protocol

```
string getservbyport (int port, string protocol)
```

Getservbyport() returns the Internet service associated with *port* for the specified *protocol* as per /etc/services. *protocol* is either TCP or UDP.

See also: **getservbyname()**.

openlog (PHP3 , PHP4)

Open connection to system logger

```
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 (PHP3 >= 3.0.7, PHP4)

Open persistent Internet or Unix domain socket connection

```
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 (PHP3 , PHP4)

Set blocking/non-blocking mode on a socket

```
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 (PHP3 , PHP4)

Generate a system log message

```
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.

XL. 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 (PHP3 >= 3.0.7, PHP4)

Fetches the machine's default NIS domain.

```
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();
    echo "Default NIS domain is: " . $domain;
?>
```

yp_order (PHP3 >= 3.0.7, PHP4)

Returns the order number for a map.

```
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);
    echo "Order number for this map is: " . $order;
?>
```

See also **yp-get-default-domain()**.

yp_master (PHP3 >= 3.0.7, PHP4)

Returns the machine name of the master NIS server for a map.

```
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);
    echo "Master for this map is: " . $master;
?>
```

See also **yp-get-default-domain()**.

yp_match (PHP3 >= 3.0.7, PHP4)

Returns the matched line.

```
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");
    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_first (PHP3 >= 3.0.7, PHP4)

Returns the first key-value pair from the named map.

```
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");
    $key = key($entry);
    echo "First entry in this map has key " . $key
          . " and value " . $entry[$key];
?>
```

See also **yp-get-default-domain()**

yp_next (PHP3 >= 3.0.7, PHP4)

Returns the next key-value pair in the named map.

```
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()**.

XLI. ODBC functions

odbc_autocommit (PHP3 >= 3.0.6, PHP4)

Toggle autocommit behaviour

```
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 (PHP3 >= 3.0.6, PHP4)

handling of binary column data

```
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 (PHP3 >= 3.0.6, PHP4)

Close an ODBC connection

```
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 (PHP3 >= 3.0.6, PHP4)

Close all ODBC connections

```
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 (PHP3 >= 3.0.6, PHP4)

Commit an ODBC transaction

```
int odbc_commit (int connection_id)
```

Returns: `true` on success, `false` on failure. All pending transactions on *`connection_id`* are committed.

odbc_connect (PHP3 >= 3.0.6, PHP4)

Connect to a datasource

```
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 (PHP3 >= 3.0.6, PHP4)

Get cursorname

```
string odbc_cursor (int result_id)
```

odbc_cursor will return a cursorname for the given result_id.

odbc_do (PHP3 >= 3.0.6, PHP4)

synonym for **odbc_exec()**

```
string odbc_do (int conn_id, string query)
```

odbc_do will execute a query on the given connection

odbc_exec (PHP3 >= 3.0.6, PHP4)

Prepare and execute a SQL statement

```
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 (PHP3 >= 3.0.6, PHP4)

execute a prepared statement

```
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 (PHP3 >= 3.0.6, PHP4)

Fetch one result row into array

```
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 (PHP3 >= 3.0.6, PHP4)

Fetch a row

```
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 (PHP3 >= 3.0.6, PHP4)

Get the columnname

```
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 (PHP3 >= 3.0.6, PHP4)

datatype of a field

```
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 (PHP3 >= 3.0.6, PHP4)

get the Length of a field

```
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 (PHP3 >= 3.0.6, PHP4)

free resources associated with a result

```
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 (PHP3 >= 3.0.6, PHP4)

handling of LONG columns

```
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 (PHP3 >= 3.0.6, PHP4)

number of columns in a result

```
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 (PHP3 >= 3.0.6, PHP4)

Open a persistent database connection

```
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 (PHP3 >= 3.0.6, PHP4)

Prepares a statement for execution

```
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 (PHP3 >= 3.0.6, PHP4)

Number of rows in a result

```
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 (PHP3 >= 3.0.6, PHP4)

get result data

```
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 (PHP3 >= 3.0.6, PHP4)

Print result as HTML table

```
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 (PHP3 >= 3.0.6, PHP4)

Rollback a transaction

```
int odbc_rollback (int connection_id)
```

Rolls back all pending statements on *`connection_id`*. Returns `true` on success, `false` on failure.

odbc_setoption (PHP3 >= 3.0.6, PHP4)

Adjust ODBC settings. Returns false if an error occurs, otherwise true.

```
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);
```

XLII. Oracle functions

Ora_Bind (unknown)

bind a PHP variable to an Oracle parameter

```
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 (unknown)

close an Oracle cursor

```
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 (unknown)

get name of Oracle result column

```
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 (unknown)

get type of Oracle result column

```
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 (unknown)

commit an Oracle transaction

```
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 (unknown)

disable automatic commit

```
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 (unknown)

enable automatic commit

```
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 (unknown)

get Oracle error message

```
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 (unknown)

get Oracle error code

```
int Ora_ErrorCode (int cursor_or_connection)
```

Returns the numeric error code of the last executed statement on the specified cursor or connection.

Note: Support for connection ids was added in 3.0.4.

Ora_Exec (unknown)

execute parsed statement on an Oracle cursor

```
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 (unknown)

fetch a row of data from a cursor

```
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 (unknown)

get data from a fetched row

```
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 (unknown)

close an Oracle connection

```
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 (unknown)

open an Oracle connection

```
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 (unknown)

open an Oracle cursor

```
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 (unknown)

parse an SQL statement

```
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 (unknown)

roll back transaction

```
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.

XLIII. 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 (unknown)

Use a PHP variable for the define-step during a SELECT

```
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 (unknown)

Bind a PHP variable to an Oracle Placeholder

```
int OCIBindByName (int stmt, string ph_name, mixed &variable, intlength [, 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 (unknown)

Establishes a connection to Oracle

```
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 (unknown)

Connect to an Oracle database and log on using a persistant connection. Returns a new session.

```
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 (unknown)

Connect to an Oracle database and log on using a new connection. Returns a new session.

```
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 (unknown)

Disconnects from Oracle

```
int OCILogOff (int connection)
```

OCILogOff() closes an Oracle connection.

OCIExecute (unknown)

Execute a statement

```
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 (unknown)

Commits outstanding transactions

```
int OCICommit (int connection)
```

OCICommit() commits all outstanding statements for Oracle connection *connection*.

OCIRollback (unknown)

Rolls back outstanding transactions

```
int OCIRollback (int connection)
```

OCIRollback() rolls back all outstanding statements for Oracle connection *connection*.

OCINewDescriptor (unknown)

Initialize a new empty descriptor LOB/FILE (LOB is default)

```
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 (unknown)

Gets the number of affected rows

```
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 (unknown)

Return the number of result columns in a statement

```
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 (unknown)

Returns coulumn value for fetched row

```
mixed 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 (unknown)

Fetches the next row into result-buffer

```
int OCIFetch (int statement)
```

OCIFetch() fetches the next row (for SELECT statements) into the internal result-buffer.

OCIFetchInto (unknown)

Fetches the next row into result-array

```
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 (unknown)

Fetch all rows of result data into an array.

```
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 (unknown)

test whether a result column is NULL

```
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 (unknown)

return result column size

```
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 (unknown)

Return a string containing server version information.

```
string OCIServerVersion (int conn)
```

Example 1. OCIServerVersion

```
<?php
   $conn = OCILogon("scott","tiger");
   print "Server Version: " . OCIServerVersion($conn);
   OCILogOff($conn);
?>
```

OCIStatementType (unknown)

Return the type of an OCI statement.

```
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 (unknown)

return a new cursor (Statement-Handle) - use this to bind ref-cursors!

```
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 (unknown)

Free all resources associated with a statement.

```
int OCIFreeStatement (int stmt)
```

OCIFreeStatement() returns true if successful, or false if unsuccessful.

OCIFreeCursor (unknown)

Free all resources associated with a cursor.

```
int OCIFreeCursor (int stmt)
```

OCIFreeCursor() returns true if successful, or false if unsuccessful.

OCIColumnName (unknown)

Returns the name of a column.

```
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 (unknown)

Returns the data type of a column.

```
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 (unknown)

Parse a query and return a statement

```
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 (unknown)

Return the last error of stmt|conn|global. If no error happened returns false.

```
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 (unknown)

Enables or disables internal debug output. By default it is disabled

```
void OCIInternalDebug (int onoff)
```

OCIInternalDebug() enables internal debug output. Set *onoff* to 0 to turn debug output off, 1 to turn it on.

XLIV. 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; you will also need the JPEG library (ftp://ftp.uu.net/graphics/jpeg/) and the TIFF library (http://www.libtiff.org/) to compile this. These two libs also quite often make problems when configuring php. Follow the messages of configure to fix possible problems. If you use pdflib 2.01 check how the lib was installed. There should be file or link libpdf.so. Version 2.01 just creates a lib with the name libpdf2.01.so which cannot be found when linking the test programm in configure. You will have to create a symbolic link from libpdf.so to libpdf2.01.so.).

Version 2.20 of pdflib has introduced more changes to its API and support for chinese and japanese fonts. This unfortunately causes some changes of the pdf module of php4 (not php3). If you use pdflib 2.20 handle the in memory generation of PDF documents with care. Until pdflib 3.0 is released it might be unstable. The encoding parameter of **pdf_set_font()** has changed to a string. This means that instead of e.g. 4 you have to use 'winansi'.

If you use pdflib 2.30 the **pdf_set_text_matrix()** will have gone. It is not supported any more. In general it is a good advise to consult the release notes of the used version of pdflib for possible changes.

Any version of PHP4 after March, 9th 2000 do not support versions of pdflib older than 3.0. PHP3 on the other hand should not be used with version newer than 2.01.

Please consult the excellent 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 (http://www.fastio.com). 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.x

```
<?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.x looks as the following (you can see the same example in the documentation for the clibpdf module):

Example 3. pdfclock example from pdflib 2.x 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 (unknown)

Returns an empty info structure for a pdf document

```
info pdf_get_info (string filename)
```

The **PDF_get_info()** function returns an empty 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.x 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 (unknown)

Fills the creator field of the info structure

```
void pdf_set_info_creator (info info, string creator)
```

The **PDF_set_info_creator()** function sets the creator field 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.x 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 (unknown)

Fills the title field of the info structure

```
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 (unknown)

Fills the subject field of the info structure

```
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 (unknown)

Fills the keywords field of the info structure

```
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 (unknown)

Fills the author field of the info structure

```
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 (unknown)

Opens a new pdf document

```
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 (unknown)

Closes a pdf document

```
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 (unknown)

Starts new page

```
void pdf_begin_page (int pdf document, double width, double height)
```

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 (unknown)

Ends a page

```
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 (unknown)

Output text at current position

```
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_boxed (unknown)

Output text in a box

```
int pdf_show_boxed (int pdf_document, string text, double x-coor, double y-coor,
double width, double height, string mode)
```

The **PDF_show_boxed()** function outputs the string *text* in a box with its lower left position at (*x-coor*, *y-coor*). The boxes dimension is *height* by *width*. The parameter *mode* determines how the text is type set. If *width* and *height* are zero, the *mode* can be "left", "right" or "center". If *width* or *height* is unequal zero it can also be "justify" and "fulljustify".

Returns the number of characters that could not be processed because they did not fit into the box.

See also **PDF_show()**, **PDF_show_xy()**.

PDF_show_xy (unknown)

Output text at given position

```
void pdf_show_xy (int pdf_document, string text, double x-coor, double y-coor)
```

The **PDF_show_xy()** function outputs the string *text* at position (*x-coor*, *y-coor*).

See also **PDF_show()**.

PDF_set_font (unknown)

Selects a font face and size

```
void pdf_set_font (int pdf_document, string font_name, double size, string
encoding [, int embed])
```

The **PDF_set_font()** function sets the current font face, font size and encoding. If you use pdflib 0.6 you will need to provide the Adobe Font Metrics (afm-files) for the font in the font path (default is ./fonts). If you use php3 or a version of pdflib older than 2.20 the fourth 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 you use php4 and a version of pdflib >= 2.20 the encoding parameter has changed to a string. Use 'winansi', 'builtin' etc. instead. If the last parameter is set to 1 the font is embedded into the pdf document otherwise it is not. To embed a font is usually a good idea if the font is not widely spread and you cannot ensure that the person watching your document has access on fonts in the document. I font is only embedded once even if you call **PDF_set_font()** several times.

Note: This function has to be called after **PDF_begin_page()** in order to create a valid pdf document.

Note: If you reference a font in a .upr file make sure the name in the afm file and the font name are the same. Otherwise, the font will be embedded several times (Thanks to Paul Haddon for finding this.)

PDF_set_leading (unknown)

Sets distance between text lines

```
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_parameter (unknown)

Sets certain parameters

```
void pdf_set_parameter (int pdf document, string name, string value)
```

The **PDF_set_parameter()** function sets several parameters of pdflib.

PDF_set_text_rendering (unknown)

Determines how text is rendered

```
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 (unknown)

Sets horizontal scaling of text

```
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 (unknown)

Sets the text rise

```
void pdf_set_text_rise (int pdf document, double rise)
```

The **PDF_set_text_rise()** function sets the text rising to *rise* points.

PDF_set_text_matrix (unknown)

Sets the text matrix

```
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 (unknown)

Sets text position

```
void pdf_set_text_pos (int pdf document, double x-coor, double y-coor)
```

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 (unknown)

Sets character spacing

```
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 (unknown)

Sets spacing between words

```
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_skew (unknown)

Skews the coordinate system

```
void pdf_skew (int pdf document, double alpha, double beta)
```

The **PDF_skew()** function skew the coordinate system by *alpha* (x) and *beta* (y) degrees. *alpha* and *beta* may not be 90 or 270 degrees.

PDF_continue_text (unknown)

Outputs text in next line

```
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 (unknown)

Returns width of text using current font

```
double pdf_stringwidth (int pdf document, string text)
```

The **PDF_stringwidth()** function returns the width of the string in *text* by using the current font. It requires a font to be set before with **PDF_set_font()**.

See also **PDF_set_font()**.

PDF_save (unknown)

Saves the current environment

```
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()** to restore the environment before **PDF_save()**.

See also **PDF_restore()**.

PDF_restore (unknown)

Restores formerly saved environment

```
void pdf_restore (int pdf document)
```

The **PDF_restore()** function restores the environment saved with **PDF_save()**. It works like the postscript command grestore.

Example 1. Save and Restore

```
<?php PDF_save($pdf);
// do all kinds of rotations, transformations, ...
PDF_restore($pdf) ?>
```

See also **PDF_save()**.

PDF_translate (unknown)

Sets origin of coordinate system

```
void pdf_translate (int pdf document, double x-coor, double y-coor)
```

The **PDF_translate()** function sets the origin of coordinate system to the point (*x-coor*, *y-coor*) relativ the current origin. 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 (unknown)

Sets scaling

```
void pdf_scale (int pdf document, double x-scale, double y-scale)
```

The **PDF_scale()** function sets 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 (unknown)

Sets rotation

```
void pdf_rotate (int pdf document, double angle)
```

The **PDF_rotate()** function sets the rotation in degress to *angle*.

PDF_setflat (unknown)

Sets flatness

```
void pdf_setflat (int pdf document, double value)
```

The **PDF_setflat()** function sets the flatness to a value between 0 and 100.

PDF_setlinejoin (unknown)

Sets linejoin parameter

```
void pdf_setlinejoin (int pdf document, long value)
```

The **PDF_setlinejoin()** function sets the linejoin parameter between a value of 0 and 2.

PDF_setlinecap (unknown)

Sets linecap parameter

```
void pdf_setlinecap (int pdf document, int value)
```

The **PDF_setlinecap()** function sets the linecap parameter between a value of 0 and 2.

PDF_setmiterlimit (unknown)

Sets miter limit

`void` **`pdf_setmiterlimit`** (`int` *`pdf document`*, `double` *`value`*)

The **PDF_setmiterlimit()** function sets the miter limit to a value greater of equal than 1.

PDF_setlinewidth (unknown)

Sets line width

`void` **`pdf_setlinewidth`** (`int` *`pdf document`*, `double` *`width`*)

The **PDF_setlinewidth()** function sets the line width to *`width`*.

PDF_setdash (unknown)

Sets dash pattern

`void` **`pdf_setdash`** (`int` *`pdf document`*, `double` *`white`*, `double` *`black`*)

The **PDF_setdash()** function sets the dash pattern *`white`* white points and *`black`* black points. If both are 0 a solid line is set.

PDF_moveto (unknown)

Sets current point

`void` **`pdf_moveto`** (`int` *`pdf document`*, `double` *`x-coor`*, `double` *`y-coor`*)

The **PDF_moveto()** function sets the current point to the coordinates *`x-coor`* and *`y-coor`*.

PDF_curveto (unknown)

Draws a curve

`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 (unknown)

Draws a line

```
void pdf_lineto (int pdf document, double x-coor, double y-coor)
```

The **PDF_lineto()** function draws a line from the current point to the point with coordinates (*x-coor*, *y-coor*).

See also **PDF_moveto()**, **PDF_curveto()**, **PDF_stroke()**.

PDF_circle (unknown)

Draws a circle

```
void pdf_circle (int pdf document, double x-coor, double y-coor, double radius)
```

The **PDF_circle()** function draws a circle with center at point (*x-coor*, *y-coor*) and radius *radius*.

See also **PDF_arc()**, **PDF_stroke()**.

PDF_arc (unknown)

Draws an arc

```
void pdf_arc (int pdf document, double x-coor, double y-coor, double radius,
double start, double end)
```

The **PDF_arc()** function draws an arc with center at point (*x-coor*, *y-coor*) and radius *radius*, starting at angle *start* and ending at angle *end*.

See also **PDF_circle()**, **PDF_stroke()**.

PDF_rect (unknown)

Draws a rectangle

```
void pdf_rect (int pdf document, double x-coor, double y-coor, double width,
double height)
```

The **PDF_rect()** function draws a rectangle with its lower left corner at point (*x-coor*, *y-coor*). This width is set to *widgth*. This height is set to *height*.

See also **PDF_stroke()**.

PDF_closepath (unknown)

Closes path

```
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 (unknown)

Draws line along path

```
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 (unknown)

Closes path and draws line along path

```
void pdf_closepath_stroke (int pdf document)
```

The **PDF_closepath_stroke()** function is a combination of **PDF_closepath()** and **PDF_stroke()**. It also clears the path.

See also **PDF_closepath()**, **PDF_stroke()**.

PDF_fill (unknown)

Fills current path

```
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 (unknown)

Fills and strokes current path

```
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 (unknown)

Closes, fills and strokes current path

```
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 (unknown)

Ends current path

```
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 (unknown)

Clips to current path

```
void pdf_clip (int pdf document)
```

The **PDF_clip()** function clips all drawing to the current path.

PDF_setgray_fill (unknown)

Sets filling color to gray value

```
void pdf_setgray_fill (int pdf document, double gray value)
```

The **PDF_setgray_fill()** function sets the current gray value to fill a path.
See also **PDF_setrgbcolor_fill()**.

PDF_setgray_stroke (unknown)

Sets drawing color to gray value

```
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 (unknown)

Sets drawing and filling color to gray value

```
void pdf_setgray (int pdf document, double gray value)
```

The **PDF_setgray()** function sets the current drawing and filling color to the given gray value.
See also **PDF_setrgbcolor_stroke()**, **PDF_setrgbcolor_fill()**.

PDF_setrgbcolor_fill (unknown)

Sets filling color to rgb color value

```
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 (unknown)

Sets drawing color to rgb color value

```
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 (unknown)

Sets drawing and filling color to rgb color value

```
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 (unknown)

Adds bookmark for current page

```
int pdf_add_outline (int pdf document, string text [, int parent [, int open]])
```

The **PDF_add_outline()** function adds a bookmark with text *text* that points to the current page. The bookmark is inserted as a child of *parent* and is by default open if *open* is not 0. The return value is an identifier for the bookmark which can be used as a parent for other bookmarks. Therefore you can build up hierarchies of bookmarks.

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 (unknown)

Sets transition between pages

```
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 (unknown)

Sets duration between pages

```
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 (unknown)

Opens a GIF image

```
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 (unknown)

Opens an image created with PHP's image functions

```
int pdf_open_memory_image (int pdf document, 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 (unknown)

Opens a JPEG image

```
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 (unknown)

Closes an image

```
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 (unknown)

Places an image on the page

```
void pdf_place_image (int pdf document, int image, double x-coor, double y-coor,
double scale)
```

The **PDF_place_image()** function places an image on the page at postion (*x-coor*, *x-coor*). The image can be scaled at the same time.

See also **PDF_put_image()**.

PDF_put_image (unknown)

Stores an image in the PDF for later use

```
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 (unknown)

Places a stored image on the page

```
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 (PHP3 >= 3.0.12, PHP4 >= 4.0b2)

Adds annotation

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*).

XLV. 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

Note: The Perl-compatible regular expression functions are available in PHP 4 and in PHP 3.0.9 and up.

preg_match (PHP3 >= 3.0.9, PHP4)

Perform a regular expression match

```
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 (PHP3 >= 3.0.9, PHP4)

Perform a global regular expression match

```
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 (PHP3 >= 3.0.9, PHP4)

Perform a regular expression search and replace

```
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 (PHP3 >= 3.0.9, PHP4)

Split string by a regular expression

```
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 (PHP3 >= 3.0.9, PHP4)

Quote regular expression characters

```
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 (PHP4)

Return array entries that match the pattern

```
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 (unknown)

describes possible modifiers in regex patterns

The current possible PCRE modifiers are listed below. The names in parentheses refer to internal PCRE names for these modifiers.

(PCRE_CASELESS)

If this modifier is set, letters in the pattern match both upper and lower case letters.

(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 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.

(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.

(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.

If this modifier is set, does normal substitution of \\ references in the replacement string, evaluates it as PHP code, and uses the result for replacing the search string.

Only uses this modifier; it is ignored by other PCRE functions.

Note: This modifier was added in PHP 4.0.

(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.

(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 modifier is set. There is no equivalent to this modifier in Perl.

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.

(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.

(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 (unknown)

describes PCRE regex syntax

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.

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 looka-

head 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.

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 meta-character, 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 non-printing 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 mul-

tiline 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 capturing 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 pat-

tern 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 case-

lessly.

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.

XLVI. PHP options & information

error_log (PHP3 , PHP4)

send an error message somewhere

```
int error_log (string message, int message_type [, string destination [, string
extra_headers]])
```

Sends an error message to the web server's error log, a TCP port or to a file. The first parameter, *message*, is the error message that should be logged. The second parameter, *message_type* says where the message should go:

Table 1. error_log() log types

0	*message is sent to PHP's system logger, using the Operating System's system logging mechanism or a file, depending on what the error_log configuration directive is set to.*
1	*message is sent by email to the address in the destination parameter. This is the only message type where the fourth parameter, extra_headers is used. This message type uses the same internal function as* ***Mail() does.***
2	*message is sent through the PHP debugging connection. This option is only available if remote debugging has been enabled. In this case, the destination parameter specifies the host name or IP address and optionally, port number, of the socket receiving the debug information.*
3	*message is appended to the file destination.*

Example 1. error_log() examples

```
// Send notification through the server log if we can not
// connect to the database.
if (!Ora_Logon($username, $password)) {
    error_log("Oracle database not available!", 0);
}

// Notify administrator by email if we run out of FOO
if (!($foo = allocate_new_foo()) {
    error_log("Big trouble, we're all out of FOOs!", 1,
              "operator@mydomain.com");
}
```

```
// other ways of calling error_log():
error_log("You messed up!", 2, "127.0.0.1:7000");
error_log("You messed up!", 2, "loghost");
error_log("You messed up!", 3, "/var/tmp/my-errors.log");
```

error_reporting (PHP3 , PHP4)

set which PHP errors are reported

```
int error_reporting ([int level])
```

Sets PHP's error reporting level and returns the old level. The error reporting level is a bitmask of the following values (follow the links for the internal values to get their meanings):

Table 1. error_reporting() bit values

value	internal name
1	E_ERROR
2	E_WARNING
4	E_PARSE
8	E_NOTICE
16	E_CORE_ERROR
32	E_CORE_WARNING

extension_loaded (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

find out whether an extension is loaded

```
bool extension_loaded (string name)
```

Returns true if the extension identified by *name* is loaded. You can see the names of various extensions by using **phpinfo()**.

See also **phpinfo()**.

Note: This function was added in 3.0.10.

getenv (PHP3 , PHP4)

Get the value of an environment variable

```
string getenv (string varname)
```

Returns the value of the environment variable *varname*, or false on an error.

```
$ip = getenv("REMOTE_ADDR"); // get the ip number of the user
```

You can see a list of all the environmental variables by using **phpinfo()**. You can find out what many of them mean by taking a look at the CGI specification (http://hoohoo.ncsa.uiuc.edu/cgi/), specifically the page on environmental variables (http://hoohoo.ncsa.uiuc.edu/cgi/env.html).

get_cfg_var (PHP3 , PHP4)

Get the value of a PHP configuration option.

```
string get_cfg_var (string varname)
```

Returns the current value of the PHP configuration variable specified by *varname*, or false if an error occurs.

It will not return configuration information set when the PHP was compiled, or read from an Apache configuration file (using the php3_configuration_option directives).

To check whether the system is using a configuration file, try retrieving the value of the cfg_file_path configuration setting. If this is available, a configuration file is being used.

get_current_user (PHP3 , PHP4)

Get the name of the owner of the current PHP script.

```
string get_current_user (void)
```

Returns the name of the owner of the current PHP script.

See also **getmyuid()**, **getmypid()**, **getmyinode()**, and **getlastmod()**.

get_magic_quotes_gpc (PHP3 >= 3.0.6, PHP4)

Get the current active configuration setting of magic quotes gpc.

```
long get_magic_quotes_gpc (void)
```

Returns the current active configuration setting of magic_quotes_gpc. (0 for off, 1 for on).

See also **get_magic_quotes_runtime()**, **set_magic_quotes_runtime()**.

get_magic_quotes_runtime (PHP3 >= 3.0.6, PHP4)

Get the current active configuration setting of magic_quotes_runtime.

```
long get_magic_quotes_runtime  (void)
```

Returns the current active configuration setting of magic_quotes_runtime. (0 for off, 1 for on).

See also **get_magic_quotes_gpc()**, **set_magic_quotes_runtime()**.

getlastmod (PHP3 , PHP4)

Get time of last page modification.

```
int getlastmod (void)
```

Returns the time of the last modification of the current page. The value returned is a Unix timestamp, suitable for feeding to **date()**. Returns false on error.

Example 1. getlastmod() example

```
// outputs e.g. 'Last modified: March 04 1998 20:43:59.'
echo "Last modified: ".date( "F d Y H:i:s.", getlastmod() );
```

See alse **date()**, **getmyuid()**, **get_current_user()**, **getmyinode()**, and **getmypid()**.

getmyinode (PHP3 , PHP4)

Get the inode of the current script.

```
int getmyinode (void)
```

Returns the current script's inode, or false on error.

See also **getmyuid()**, **get_current_user()**, **getmypid()**, and **getlastmod()**.

> **Note:** This function is not supported on Windows systems.

getmypid (PHP3 , PHP4)

Get PHP's process ID.

```
int getmypid (void)
```

Returns the current PHP process ID, or false on error.

Note that when running as a server module, separate invocations of the script are not guaranteed to have distinct pids.

See also **getmyuid()**, **get_current_user()**, **getmyinode()**, and **getlastmod()**.

getmyuid (PHP3 , PHP4)

Get PHP script owner's UID.

```
int getmyuid (void)
```

Returns the user ID of the current script, or false on error.

See also **getmypid()**, **get_current_user()**, **getmyinode()**, and **getlastmod()**.

getrusage (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Get the current resource usages.

```
array getrusage ([int who])
```

This is an interface to getrusage(2). It returns an associative array containing the data returned from the system call. If who is 1, getrusage will be called with RUSAGE_CHILDREN.

All entries are accessible by using their documented field names.

Example 1. Getrusage Example

```
$dat = getrusage();
echo $dat["ru_nswap"];         # number of swaps
echo $dat["ru_majflt"];        # number of page faults
echo $dat["ru_utime.tv_sec"];  # user time used (seconds)
echo $dat["ru_utime.tv_usec"]; # user time used (microseconds)
```

See your system's man page for more details.

phpinfo (PHP3 , PHP4)

Output lots of PHP information.

```
int phpinfo (void)
```

Outputs a large amount of information about the current state of PHP. This includes information about PHP compilation options and extensions, the PHP version, server information and environment (if compiled as a module), the PHP environment, OS version information, paths, master and local values of configuration options, HTTP headers, and the GNU Public License.

See also **phpversion()**.

phpversion (PHP3 , PHP4)

Get the current PHP version.

```
string phpversion (void)
```

Returns a string containing the version of the currently running PHP parser.

Example 1. phpversion() example

```
// prints e.g. 'Current PHP version: 3.0rel-dev'
echo "Current PHP version: ".phpversion();
```

See also **phpinfo()**.

php_logo_guid (PHP4 >= 4.0b4)

Get the logo guid

```
string php_logo_guid (void)
```

Note: This funcionality was added in PHP4 Beta 4.

putenv (PHP3 , PHP4)

Set the value of an environment variable.

```
void putenv (string setting)
```

Adds *setting* to the environment.

Example 1. Setting an Environment Variable

```
putenv("UNIQID=$uniqid");
```

set_magic_quotes_runtime (PHP3 >= 3.0.6, PHP4)

Set the current active configuration setting of magic_quotes_runtime.

```
long set_magic_quotes_runtime  (int new_setting)
```

Set the current active configuration setting of magic_quotes_runtime. (0 for off, 1 for on)

See also **get_magic_quotes_gpc()**, **get_magic_quotes_runtime()**.

set_time_limit (PHP3 , PHP4)

limit the maximum execution time

```
void set_time_limit (int seconds)
```

Set the number of seconds a script is allowed to run. If this is reached, the script returns a fatal error. The default limit is 30 seconds or, if it exists, the max_execution_time value defined in the configuration file. If seconds is set to zero, no time limit is imposed.

When called, **set_time_limit()** restarts the timeout counter from zero. In other words, if the timeout is the default 30 seconds, and 25 seconds into script execution a call such as set_time_limit(20) is made, the script will run for a total of 45 seconds before timing out.

Note that **set_time_limit()** has no effect when PHP is running in safe mode. There is no workaround other than turning off safe mode or changing the time limit in the configuration file.

zend_logo_guid (PHP4 >= 4.0b4)

Get the zend guid

```
string zend_logo_guid (void)
```

Note: This funcionality was added in PHP4 Beta 4.

XLVII. POSIX functions

This module contains an interface to those functions defined in the IEEE 1003.1 (POSIX.1) standards document which are not accessible through other means. POSIX.1 for example defined the open(), read(), write() and close() functions, too, which traditionally have been part of PHP3 for a long time. Some more system specific functions have not been available before, though, and this module tries to remedy this by providing easy access to these functions.

posix_kill (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Send a signal to a process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the current process identifier

```
int posix_getpid (void )
```

Return the process identifier of the current process.

posix_getppid (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the parent process identifier

```
int posix_getppid (void )
```

Return the process identifier of the parent process of the current process.

posix_getuid (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the real user ID of the current process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the effective user ID of the current process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the real group ID of the current process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the effective group ID of the current process

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Set the effective UID of the current process

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Set the effective GID of the current process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the group set of the current process

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return login name

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return the current process group identifier

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Make the current process a session leader

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

set process group id for job control

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Get process group id for job control

```
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_getsid (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Get the current sid of the process

```
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 (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Get system name

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Get process times

```
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 (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Get path name of controlling terminal

```
string posix_ctermid (void )
```

Needs to be written.

posix_ttyname (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Determine terminal device name

```
string posix_ttyname (int fd)
```

Needs to be written.

posix_isatty (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Determine if a file descriptor is an interactive terminal

```
bool posix_isatty (int fd)
```

Needs to be written.

posix_getcwd (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Pathname of current directory

```
string posix_getcwd (void )
```

Needs to be written ASAP.

posix_mkfifo (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Create a fifo special file (a named pipe)

```
bool posix_getcwd (string pathname, int mode)
```

Needs to be written ASAP..

posix_getgrnam (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return info about a group by name

```
array posix_getgrnam (string name)
```

Needs to be written.

posix_getgrgid (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return info about a group by group id

```
array posix_getgrgid (int gid)
```

Needs to be written.

posix_getpwnam (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return info about a user by username

```
array posix_getpwnam (string username)
```

Returns an associative array containing information about a user referenced by an alphanumeric username, passed in the *username* parameter.

The array elements returned are:

Table 1. The user information array

Element	Description
name	The name element contains the username of the user. This is a short, usually less than 16 character "handle" of the user, not her real, full name. This should be the same as the *username parameter used when calling the function, and hence redundant.*
passwd	The passwd element contains the user's password in an encrypted format. Often, for example on a system employing "shadow" passwords, an asterisk is returned instead.
uid	User ID of the user in numeric form.
gid	The group ID of the user. Use the function **posix_getgrgid() to resolve the group name and a list of its members.**
gecos	GECOS is an obsolete term that refers to the finger information field on a Honeywell batch processing system. The field, however, lives on, and its contents have been formalized by POSIX. The field contains a comma separated list containing the user's full name, office phone, office number, and home phone number. On most systems, only the user's full name is available.
dir	This element contains the absolute path to the home directory of the user.
shell	The shell element contains the absolute path to the executable of the user's default shell.

posix_getpwuid (PHP3 >= 3.0.13, PHP4 >= 4.0b4)

Return info about a user by user id

```
array posix_getpwuid (int uid)
```

Returns an associative array containing information about a user referenced by a numeric user ID, passed in the *uid* parameter.

The array elements returned are:

Table 1. The user information array

Element	Description
name	The name element contains the username of the user. This is a short, usually less than 16 character "handle" of the user, not her real, full name.
passwd	The passwd element contains the user's password in an encrypted format. Often, for example on a system employing "shadow" passwords, an asterisk is returned instead.
uid	User ID, should be the same as the *uid parameter used when calling the function, and hence redundant.*
gid	The group ID of the user. Use the function **posix_getgrgid() to resolve the group name and a list of its members.**
gecos	GECOS is an obsolete term that refers to the finger information field on a Honeywell batch processing system. The field, however, lives on, and its contents have been formalized by POSIX. The field contains a comma separated list containing the user's full name, office phone, office number, and home phone number. On most systems, only the user's full name is available.
dir	This element contains the absolute path to the home directory of the user.
shell	The shell element contains the absolute path to the executable of the user's default shell.

posix_getrlimit (PHP3 >= 3.0.10, PHP4 >= 4.0b4)

Return info about system ressource limits

```
array posix_getrlimit (void )
```

Needs to be written ASAP.

XLVIII. 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 uses unix domain sockets. A table is shown below describing these new connection 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 sockets".

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** or **end**. If the transaction fails the transaction should be closed with **rollback** or **abort**.

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_Close (unknown)

closes a PostgreSQL connection

```
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 (unknown)

returns number of affected tuples

```
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 (unknown)

opens a connection

```
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 (unknown)

database name

```
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 (unknown)

error message

```
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 (unknown)

execute a query

```
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 (unknown)

fetch row as array

```
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 (unknown)

fetch row as object

```
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 (unknown)

get row as enumerated array

```
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 (unknown)

Test if a field is NULL

```
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 (unknown)

Returns the name of a field

```
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 (unknown)

Returns the number of a column

```
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 (unknown)

Returns the printed length

```
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 (unknown)

Returns the internal storage size of the named field

```
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 (unknown)

Returns the type name for the corresponding field number

```
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 (unknown)

Frees up memory

```
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 (unknown)

Returns the last object identifier

```
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 (unknown)

Returns the host name

```
string pg_host (int connection_id)
```

pg_Host() will return the host name of the given PostgreSQL connection identifier is connected to.

pg_loclose (PHP3 , PHP4)

close a large object

```
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 (PHP3 , PHP4)

create a large object

```
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 (PHP3 , PHP4)

open a large object

```
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 (PHP3 , PHP4)

read a large object

```
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 (PHP3 , PHP4)

read a entire large object

```
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 (PHP3 , PHP4)

delete a large object

```
void pg_lounlink (int conn, int lobjid)
```

pg_lounlink() deletes a large object with the *lobjid* identifier for that large object.

pg_lowrite (PHP3 , PHP4)

write a large object

```
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 (unknown)

Returns the number of fields

```
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 (unknown)

Returns the number of rows

```
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 (unknown)

Returns options

```
string pg_options (int connection_id)
```

pg_Options() will return a string containing the options specified on the given PostgreSQL connection identifier.

pg_pConnect (unknown)

Make a persistent database connection

```
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 (unknown)

Returns the port number

```
int pg_port (int connection_id)
```

pg_Port() will return the port number that the given PostgreSQL connection identifier is connected to.

pg_Result (unknown)

Returns values from a result identifier

```
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 (PHP3 , PHP4)

Returns the tty name

```
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.

XLIX. Program Execution functions

escapeshellcmd (PHP3 , PHP4)

escape shell metacharacters

```
string escapeshellcmd (string command)
```

EscapeShellCmd() escapes any characters in a string that might be used to trick a shell command into executing arbitrary commands. This function should be used to make sure that any data coming from user input is escaped before this data is passed to the **exec()** or **system()** functions, or to the backtick operator. A standard use would be:

```
system(EscapeShellCmd($cmd))
```

See also **exec()**, **popen()**, **system()**, and the backtick operator.

exec (PHP3 , PHP4)

Execute an external program

```
string exec (string command [, string array [, int return_var]])
```

exec() executes the given *command*, however it does not output anything. It simply returns the last line from the result of the command. If you need to execute a command and have all the data from the command passed directly back without any interference, use the **PassThru()** function.

If the *array* argument is present, then the specified array will be filled with every line of output from the command. Note that if the array already contains some elements, **exec()** will append to the end of the array. If you do not want the function to append elements, call **unset()** on the array before passing it to **exec()**.

If the *return_var* argument is present along with the *array* argument, then the return status of the executed command will be written to this variable.

Note that if you are going to allow data coming from user input to be passed to this function, then you should be using **EscapeShellCmd()** to make sure that users cannot trick the system into executing arbitrary commands.

See also **system()**, **PassThru()**, **popen()**, **EscapeShellCmd()**, and the backtick operator.

passthru (PHP3 , PHP4)

Execute an external program and display raw output

```
void passthru (string command [, int return_var])
```

The **passthru()** function is similar to the **Exec()** function in that it executes a *command*. If the *return_var* argument is present, the return status of the Unix command will be placed here. This function

should be used in place of **Exec()** or **System()** when the output from the Unix command is binary data which needs to be passed directly back to the browser. A common use for this is to execute something like the pbmplus utilities that can output an image stream directly. By setting the content-type to *image/gif* and then calling a pbmplus program to output a gif, you can create PHP scripts that output images directly.

See also **exec()**, **system()**, **popen()**, **EscapeShellCmd()**, and the backtick operator.

system (PHP3 , PHP4)

Execute an external program and display output

```
string system (string command [, int return_var])
```

System() is just like the C version of the function in that it executes the given *command* and outputs the result. If a variable is provided as the second argument, then the return status code of the executed command will be written to this variable.

Note, that if you are going to allow data coming from user input to be passed to this function, then you should be using the **EscapeShellCmd()** function to make sure that users cannot trick the system into executing arbitrary commands.

The **System()** call also tries to automatically flush the web server's output buffer after each line of output if PHP is running as a server module.

Returns the last line of the command output on success, and false on failure.

If you need to execute a command and have all the data from the command passed directly back without any interference, use the **PassThru()** function.

See also **exec()**, **PassThru()**, **popen()**, **EscapeShellCmd()**, and the backtick operator.

L. GNU Recode functions

This module contains an interface to the GNU Recode library, version 3.5. To be able to use the functions defined in this module you must compile you PHP interpreter using the –with-recode option. In order to do so, you must have GNU Recode 3.5 or higher installed on your system.

The GNU Recode library converts files between various coded character sets and surface encodings. When this cannot be achieved exactly, it may get rid of the offending characters or fall back on approximations. The library recognises or produces nearly 150 different character sets and is able to convert files between almost any pair. Most RFC 1345 character sets are supported.

recode_string (PHP3 >= 3.0.13, PHP4 >= 4.0RC1)

Recode a string according to a recode request

```
string recode_string (string request, string string)
```

Recode the string *string* according to the recode request *request*. Returns FALSE, if unable to comply, TRUE otherwise.

A simple recode request may be "lat1..iso646-de". See also the GNU Recode documentation of your installation for detailed instructions about recode requests.

recode (PHP4 >= 4.0RC1)

Recode a string according to a recode request

```
string recode_string (string request, string string)
```

Note: This is an alias for **recode_string()**. It has been added in PHP4.

recode_file (PHP3 >= 3.0.13, PHP4 >= 4.0RC1)

Recode from file to file according to recode request

```
bool recode_file (int input, int output)
```

Recode the file referenced by file handle *input* into the file referenced by file handle *output* according to the recode request. Returns FALSE, if unable to comply, TRUE otherwise.

This function does not currently process filehandles referencing remote files (URLs). Both filehandles must refer to local files.

LI. 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. It's in manpage format, so you'll want to do something along the lines of **man /usr/local/src/regex/regex.7** in order to read it.

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 newline
   characters in $string. */
```

ereg (PHP3 , PHP4)

regular expression match

```
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 (PHP3 , PHP4)

replace regular expression

```
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 (PHP3 , PHP4)

case insensitive regular expression match

```
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 (PHP3 , PHP4)

replace regular expression case insensitive

```
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 (PHP3 , PHP4)

split string into array by regular expression

```
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 the regular expression *`pattern`*. If *`limit`* is set, the returned array will contaion a maximum of *`limit`* elements with the last element containing the whole rest of *`string`*. If an error occurs, **split()** returns false.

To get the first five fields from a line from `/etc/passwd`:

Example 1. split() example

```
$passwd_list = split( ":", $passwd_line, 5 );
```

To parse a date which may be delimited with slashes, dots, or hyphens:

Example 2. split() example

```
$date = "04/30/1973";  // Delimiters may be slash, dot, or hyphen
list( $month, $day, $year ) = split( '[/.-]', $date );
echo "Month: $month; Day: $day; Year: $year<br>\n";
```

Note that *`pattern`* is case-sensitive.

Note that if you don't require the power of regular expressions, it is faster to use **explode()**, which doesn't incur the overhead of the regular expression engine.

Please note that *`pattern`* is a regular expression. If you want to split on any of the characters which are considered special by regular expressions, you'll need to escape them first. If you think **split()** (or any other regex function, for that matter) is doing something weird, please read the file `regex.7`, included in the `regex/` subdirectory of the PHP distribution. It's in manpage format, so you'll want to do something along the lines of **man /usr/local/src/regex/regex.7** in order to read it.

See also: **explode()** and **implode()**.

sql_regcase (PHP3 , PHP4)

make regular expression for case insensitive match

```
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.

LII. 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 (PHP3 >= 3.0.6, PHP4)

Get a semaphore id

```
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 (PHP3 >= 3.0.6, PHP4)

Acquire a semaphore

```
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 (PHP3 >= 3.0.6, PHP4)

Release a semaphore

```
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 (PHP3 >= 3.0.6, PHP4)

Creates or open a shared memory segment

```
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 (PHP3 >= 3.0.6, PHP4)

Disconnects from shared memory segment

```
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 (PHP3 >= 3.0.6, PHP4)

Removes shared memory from Unix systems

```
int shm_remove (int shm_identifier)
```

Removes shared memory from Unix systems. All data will be destroyed.

shm_put_var (PHP3 >= 3.0.6, PHP4)

Inserts or updates a variable in shared memory

```
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 (PHP3 >= 3.0.6, PHP4)

Returns a variable from shared memory

```
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 (PHP3 >= 3.0.6, PHP4)

Removes a variable from shared memory

```
int shm_remove_var (int id, int variable_key)
```

Removes a variable with a given *variable_key* and frees the occupied memory.

LIII. 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.

If you are familiar with the session management of PHPLIB, you will notice that some concepts are similar to PHP's session support.

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.

`track_vars` and `gpc_globals` configuration settings influence how the session variables get restored. If `track_vars` is enabled, then the restored session variables will be available in the global associative array $HTTP_STATE_VARS. If `gpc_globals` is enabled, then the session variables will be restored to corresponding global variables. If both of these settings are enabled, then the globals variables and the $HTTP_STATE_VARS entries will reference the same value.

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>

<php?
# 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.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 (PHP4)

Initialize session data

```
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 (PHP4)

Destroys all data registered to a session

```
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 (PHP4)

Get and/or set the current session name

```
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.

The session name references the session id in cookies and URLs. It should contain only alphanumeric characters; it should be short and descriptive (i.e. for users with enabled cookie warnings). The session name is resetted to the default value stored in `session.name` at request startup time. Thus, you need to call **session_name()** for every request (and before **session_start()** or **session_register()** are called).

Example 1. session_name() examples

```
<?php

# set the session name to WebsiteID

$previous_name = session_name ("WebsiteID");
```

```
echo "The previous session name was $previous_name<p>";
?>
```

Note: This function was added in PHP 4.0.

session_module_name (PHP4)

Get and/or set the current session module

```
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 (PHP4)

Get and/or set the current session save path

```
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 (PHP4)

Get and/or set the current session id

```
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 (PHP4)

Register one or more variables with the current session

```
bool session_register (mixed name [, mixed ...])
```

session_register() variable number of arguments, any of which can be either a string holding the variable name or an array consisting of such variable names or other arrays. For each encountered variable name, **session_register()** registers the global variable named by it 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 (PHP4)

Unregister a variable from the current session

```
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 (PHP4)

Find out if a variable is registered in a session

```
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 (PHP4)

Decodes session data from a string

```
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 (PHP4)

Encodes the current session data as a string

```
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.

LIV. 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 (PHP3 , PHP4)

Fetch an SNMP object

```
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 (PHP3 >= 3.0.12, PHP4 >= 4.0b2)

Set an SNMP object

```
bool snmpset (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 (PHP3 , PHP4)

Fetch all the SNMP objects from an agent

```
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 (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Query for a tree of information about a network entity

```
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 (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Fetch the current value of the UCD library's quick_print setting

```
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 (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Set the value of quick_print within the UCD SNMP library.

```
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.

LV. 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 (unknown)

Quote string with slashes in a C style.

```
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 (unknown)

Quote string with slashes

```
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 (PHP3 >= 3.0.9, PHP4)

Convert binary data into hexadecimal representation

```
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 (unknown)

Remove trailing whitespace.

```
string chop (string str)
```

Returns the argument string without trailing whitespace, including newlines.

Example 1. Chop() example

```
$trimmed = chop ($line);
```

See also **trim()**.

Chr (unknown)

Return a specific character.

```
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 (PHP3 >= 3.0.6, PHP4)

Split a string into smaller chunks.

```
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 (PHP3 >= 3.0.6, PHP4)

Convert from one Cyrillic character set to another.

```
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

count_chars (PHP4 >= 4.0b4)

Return information abouts characters used in a string.

```
mixed count_chars (string string [, mode])
```

Counts the number of occurances of every byte-value (0..255) in *string* and returns it in various ways. The optional parameter *Mode* default to 0. Depending on *mode* **count_chars()** returns one of the following:

- 0 - an array with the byte-value as key and the freqency of every byte as value.
- 1 - same as 0 but only byte-values with a frequency greater zero are listed.
- 2 - same as 0 but only byte-values with a frequency equal to zero are listed.
- 3 - a string containing all used byte-values is returned.
- 4 - a string containing all not used byte-values is returned.

Note: This function was added in PHP 4.0.

crypt (PHP3 , PHP4)

DES-encrypt a string.

```
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 (unknown)

Output one or more strings.

```
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";

echo "This spans
multiple lines. The newlines will be
output as well";

echo "This spans\nmultiple lines. The newlines will be\noutput as well.";
```

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 (PHP3 , PHP4)

Split a string by string

```
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 (PHP3 , PHP4)

Flush the output buffer.

```
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_html_translation_table (PHP4 >= 4.0b4)

Returns the translation table used by **htmlspecialchars()** and **htmlentities()**.

```
string get_html_translation_table  (int table)
```

get_html_translation_table() will return the translation table that is used internally for **htmlspecialchars()** and **htmlentities()**. Ther are two new defines (*HTML_ENTITIES*, *HTML_SPECIALCHARS*) that allow you to specify the table you want.

Example 1. Translation Table Example

```
$trans = get_html_translation_table (HTML_ENTITIES);
$str = "Hallo & <Frau> & Krämer";
$encoded = strtr ($str, $trans);
```

The `$encoded` variable will now contain: "Hallo & <Frau> & Krämer".

The cool thing is using **array_flip()** to change the direction of the translation.

```
$trans = array_flip ($trans);
$original = strtr ($str, $trans);
```

The content of `$original` would be: "Hallo & <Frau> & Krämer".

Note: This function was added in PHP 4.0.

See also: **htmlspecialchars()**, **htmlentities()**, **strtr()**, and **array_flip()**.

get_meta_tags (PHP3 >= 3.0.4, PHP4)

Extracts all meta tag content attributes from a file and returns an array.

```
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 (PHP3 , PHP4)

Convert all applicable characters to HTML entities.

```
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 (PHP3 , PHP4)

Convert special characters to HTML entities.

```
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 '&'
- '"' (double quote) becomes '"'
- '<' (less than) becomes '<'
- '>' (greater than) becomes '>'

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 (PHP3 , PHP4)

Join array elements with a string

```
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 (PHP3 , PHP4)

Join array elements with a string.

```
string join (string glue, array pieces)
```

join() is an alias to **implode()**, and is identical in every way.

See also **explode()**, **implode()**, and **split()**.

levenshtein (PHP3 , PHP4)

Calculate Levenshtein distance between two strings

```
int levenshtein (string str1, string str2)
```

This function return the Levenshtein-Distance between the two argument strings or -1, if one of the argument strings is longer than the limit of 255 characters.

The Levenshtein distance is defined as the minimal number of characters you have to replace, insert or delete to transform *str1* into *str2*. The complexity of the algorithm is `O(m*n)`, where `n` and `m` are the length of *str1* and *str2* (rather good when compared to **similar_text()**, which is O(max(n,m)**3), but still expensive).

See also **soundex()**, **similar_text()** and **metaphone()**.

ltrim (PHP3 , PHP4)

Strip whitespace from the beginning of a string.

```
string ltrim (string str)
```

This function strips whitespace from the start of a string and returns the stripped string. The whitespace characters it currently strips are: "\n", "\r", "\t", "\v", "\0", and a plain space.

See also **chop()** and **trim()**.

md5 (PHP3 , PHP4)

Calculate the md5 hash of a string.

```
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 (unknown)

Calculate the metaphone key of a string.

```
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 (PHP3 , PHP4)

Converts newlines to HTML line breaks.

```
string nl2br (string string)
```

Returns *string* with '
' inserted before all newlines.

See also **htmlspecialchars()** and **htmlentities()**.

Ord (unknown)

Return ASCII value of character.

```
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 (PHP3 , PHP4)

Parses the string into variables.

```
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 (unknown)

Output a string

```
print (string arg)
```

Outputs *arg*.

See also: **echo()**, **printf()**, and **flush()**.

printf (PHP3 , PHP4)

Output a formatted string.

```
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 (PHP3 >= 3.0.6, PHP4)

Convert a quoted-printable string to an 8 bit string.

```
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 (unknown)

quote meta characters

```
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 (PHP3 , PHP4)

Decode URL-encoded strings

```
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 (PHP3 , PHP4)

URL-encode according to RFC1738.

```
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 (PHP3 , PHP4)

Set locale information.

```
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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Calculate the similarity between two strings.

```
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 (PHP3 , PHP4)

Calculate the soundex key of a string

```
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 (PHP3 , PHP4)

Return a formatted string

```
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
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
X - the argument is treated as an integer and presented as a hexadecimal number (with
```

See also: **printf()** and **number_format()**.

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 (PHP3 >= 3.0.2, PHP4)

Binary safe case-insensitive string comparison.

```
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.

Example 1. strcasecmp() example

```
$var1 = "Hello";
$var2 = "hello";
if ( !strcasecmp($var1,$var2) ) {
    echo '$var1 is equal to $var2 in a case-insensitive string comparison';
}
```

See also **ereg()**, **strcmp()**, **substr()**, **stristr()**, and **strstr()**.

strchr (PHP3 , PHP4)

Find the first occurrence of a character.

```
string strchr (string haystack, string needle)
```

This function is an alias for **strstr()**, and is identical in every way.

strcmp (PHP3 , PHP4)

Binary safe string comparison

```
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 (PHP3 >= 3.0.3, PHP4)

Find length of initial segment not matching mask.

```
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 (PHP3 >= 3.0.8, PHP4 >= 4.0b2)

Strip HTML and PHP tags from a string

```
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 (unknown)

un-quote string quoted with addcslashes

```
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 (unknown)

Un-quote string quoted with addslashes

```
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 (PHP3 >= 3.0.6, PHP4)

Case-insensitive **strstr()**.

```
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 (PHP3 , PHP4)

Get string length.

```
int strlen (string str)
```

Returns the length of *string*.

strpos (PHP3 , PHP4)

Find position of first occurrence of a string.

```
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.

Note: It is easy to mistake the return values for "character found at position 0" and "character not found". Here's how to detect the difference:

```
// in PHP 4.0b3 and newer:
$pos = strpos ("b", $mystring);
if ($pos === false) { // note: three equal signs
    // not found...
}

// in versions older than 4.0b3:
$pos = strpos ("b", $mystring);
if (is_string ($pos) && !$pos) {
    // not found...
}
```

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 (PHP3 , PHP4)

Find the last occurrence of a character in a string.

```
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 (PHP4 >= 4.0b4)

Repeat a string.

```
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 (PHP3 , PHP4)

Reverse a string.

```
string strrev (string string)
```

Returns *string*, reversed.

strrpos (PHP3 , PHP4)

Find position of last occurrence of a char in a string.

```
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 (PHP3 >= 3.0.3, PHP4)

Find length of initial segment matching mask.

```
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 (PHP3 , PHP4)

Find first occurrence of a string.

```
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.

Example 1. Strstr() example

```
$email = 'sterling@designmultimedia.com';
$domain = strstr ($email, '@');
print $domain; // prints designmultimedia.com
```

See also **stristr()**, **strrchr()**, **substr()**, and **ereg()**.

strtok (PHP3 , PHP4)

Tokenize string

```
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 (PHP3 , PHP4)

Make a string lowercase.

```
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.

Example 1. Strtolower() example

```
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtolower($str);
print $str; # Prints mary had a little lamb and she loved it so
```

See also **strtoupper()** and **ucfirst()**.

strtoupper (PHP3 , PHP4)

Make a string uppercase.

```
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.

Example 1. Strtoupper() example

```
$str = "Mary Had A Little Lamb and She LOVED It So";
$str = strtoupper ($str);
print $str; # Prints MARY HAD A LITTLE LAMB AND SHE LOVED IT SO
```

See also **strtolower()** and **ucfirst()**.

str_replace (PHP3 >= 3.0.6, PHP4)

Replace all occurrences of needle in haystack with str.

```
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.

> **Note: Str_replace()** was added in PHP 3.0.6, but was buggy up until PHP 3.0.8.

See also **ereg_replace()** and **strtr()**.

strtr (PHP3 , PHP4)

Translate certain characters.

```
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");
```

strtr() can be called with only two arguments. If called with two arguments it behaves in a new way: *from* then has to be an array that contains string -> string pairs that will be replaced in the source string. **strtr()** will always look for the longest possible match first and will *NOT* try to replace stuff that it has already worked on.

Examples:

```
$trans = array ("hello" => "hi", "hi" => "hello");
echo strtr("hi all, I said hello", $trans) . "\n";
```

This will show: "hello all, I said hi",

> **Note:** This feature (two arguments) was added in PHP 4.0.

See also **ereg_replace()**.

substr (PHP3 , PHP4)

Return part of a string

```
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()**.

substr_replace (PHP4 >= 4.0b4)

Replace text within a portion of a string.

```
string substr_replace (string string, string replacement, int start [, int
length])
```

substr_replace() replaces the part of *string* delimited by the *start* and (optionally) *length* parameters with the string given in *replacement*. The result is returned.

If *start* is positive, the replacing will begin at the *start*'th offset into *string*.

If *start* is negative, the replacing will begin at the *start*'th character from the end of *string*.

If *length* is given and is positive, it represents the length of the portion of *string* which is to be replaced. If it is negative, it represents the number of characters from the end of *string* at which to stop replacing. If it is not given, then it will default to strlen(*string*); i.e. end the replacing at the end of *string*.

Example 1. Substr_replace() example

```
<?php
$var = 'ABCDEFGH:/MNRPQR/';
echo "Original: $var<hr>\n";

/* These two examples replace all of $var with 'bob'. */
echo substr_replace ($var, 'bob', 0) . "<br>\n";
echo substr_replace ($var, 'bob', 0, strlen ($var)) . "<br>\n";

/* Insert 'bob' right at the beginning of $var. */
echo substr_replace ($var, 'bob', 0, 0) . "<br>\n";

/* These next two replace 'MNRPQR' in $var with 'bob'. */
echo substr_replace ($var, 'bob', 10, -1) . "<br>\n";
echo substr_replace ($var, 'bob', -7, -1) . "<br>\n";

/* Delete 'MNRPQR' from $var. */
echo substr_replace ($var, '', 10, -1) . "<br>\n";
?>
```

See also **str_replace()** and **substr()**.

Note: Substr_replace() was added in PHP 4.0.

trim (PHP3 , PHP4)

Strip whitespace from the beginning and end of a string.

```
string trim (string str)
```

This function strips whitespace from the start and the end of a string and returns the stripped string. The whitespace characters it currently strips are: "\n", "\r", "\t", "\v", "\0", and a plain space.

See also **chop()** and **ltrim()**.

ucfirst (PHP3 , PHP4)

Make a string's first character uppercase.

```
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.

Example 1. ucfirst() example

```
$text = 'mary had a little lamb and she loved it so.';
$text = ucfirst($text); // $text is now Mary had a lit-
tle lamb and she loved it so.
```

See also **strtoupper()** and **strtolower()**.

ucwords (PHP3 >= 3.0.3, PHP4)

Uppercase the first character of each word in a string

```
string ucwords (string str)
```

Capitalizes the first character of each word in *str* if that character is alphabetic.

Example 1. ucwords() example

```
$text = "mary had a little lamb and she loved it so.";
$text = ucwords($text); // $text is now: Mary Had A Lit-
tle Lamb And She Loved It So.
```

See also **strtoupper()**, **strtolower()** and **ucfirst()**.

LVI. Shockwave Flash functions

PHP offers the ability to create Shockwave Flash files via Paul Haeberli's libswf module. You can download libswf at http://reality.sgi.com/grafica/flash/. Once you have libswf all you need to do is to configure `-with-swf[=DIR]` where DIR is the location of libswf.a.

Once you've successfully installed PHP with Shockwave Flash support you can then go about creating Shockwave files from PHP. You would be surprised at what you can do, take the following code:

Example 1. SWF example

```
<php
swf_openfile ("test.swf", 256, 256, 30, 1, 1, 1);
swf_ortho2 (-100, 100, -100, 100);
swf_defineline (1, -70, 0, 70, 0, .2);
swf_definerect (4, 60, -10, 70, 0, 0);
swf_definerect (5, -60, 0, -70, 10, 0);
swf_addcolor (0, 0, 0, 0);

swf_definefont (10, "Mod");
swf_fontsize (5);
swf_fontslant (10);
swf_definetext (11, "This be Flash wit PHP!", 1);

swf_pushmatrix ();
swf_translate (-50, 80, 0);
swf_placeobject (11, 60);
swf_popmatrix ();

for ($i = 0; $i < 30; $i++) {
    $p = $i/(30-1);
    swf_pushmatrix ();
    swf_scale (1-($p*.9), 1, 1);
    swf_rotate (60*$p,  'z');
    swf_translate (20+20*$p, $p/1.5, 0);
    swf_rotate (270*$p,  'z');
    swf_addcolor ($p, 0, $p/1.2, -$p);
    swf_placeobject (1, 50);
    swf_placeobject (4, 50);
    swf_placeobject (5, 50);
    swf_popmatrix ();
    swf_showframe ();
}

for ($i = 0; $i < 30; $i++) {
    swf_removeobject (50);
    if (($i%4) == 0) {
        swf_showframe ();
    }
}

swf_startdoaction ();
swf_actionstop ();
swf_enddoaction ();

swf_closefile ();
?>
```

It will produce the animation found at the following url: http://www.designmultimedia.com/swfphp/test.swf (http://www.designmultimedia.com/swfphp/test.swf).

Note: SWF support was added in PHP4 RC2.

swf_openfile (PHP4 >= 4.0RC2)

Open a new Shockwave Flash file

```
void swf_openfile  (string filename, float width, float height, float framerate,
float r, float g, float b)
```

The **swf_openfile()** function opens a new file named *filename* with a width of *width* and a height of *height* a frame rate of *framerate* and background with a red color of *r* a green color of *g* and a blue color of *b*.

The **swf_openfile()** must be the first function you call, otherwise your script will cause a segfault. If you want to send your output to the screen make the filename: "php://stdout" (support for this is in 4.0.1 and up).

swf_closefile (PHP4 >= 4.0RC2)

Close the current Shockwave Flash file

```
void swf_closefile (void);
```

Close a file that was opened by the **swf_openfile()** function.

swf_labelframe (PHP4 >= 4.0RC2)

Label the current frame

```
void swf_labelframe  (string name)
```

Label the current frame with the name given by the *name* parameter.

swf_showframe (PHP4 >= 4.0RC2)

Display the current frame

```
void swf_showframe (void);
```

The swf_showframe function will output the current frame.

swf_setframe (PHP4 >= 4.0RC2)

Switch to a specified frame

```
void swf_setframe  (int framenumber)
```

The **swf_setframe()** changes the active frame to the frame specified by *framenumber*.

swf_getframe (PHP4 >= 4.0RC2)

Get the frame number of the current frame.

```
int swf_getframe (void);
```

The **swf_getframe()** function gets the number of the current frame.

swf_mulcolor (PHP4 >= 4.0RC2)

Sets the global multiply color to the rgba value specified

```
void swf_mulcolor  (float r, float g, float b, float a)
```

The **swf_mulcolor()** function sets the global multiply color to the *rgba* color specified. This color is then used (implicitly) by the **swf_placeobject()**, **swf_modifyobject()** and the **swf_addbuttonrecord()** functions. The color of the object will be multiplied by the *rgba* values when the object is written to the screen.

Note: The *rgba* values can be either positive or negative.

swf_addcolor (PHP4 >= 4.0RC2)

Set the global add color to the rgba value specified

```
void swf_addcolor  (float r, float g, float b, float a)
```

The **swf_addcolor()** function sets the global add color to the *rgba* color specified. This color is then used (implicitly) by the **swf_placeobject()**, **swf_modifyobject()** and the **swf_addbuttonrecord()** functions. The color of the object will be add by the *rgba* values when the object is written to the screen.

Note: The *rgba* values can be either positive or negative.

swf_placeobject (PHP4 >= 4.0RC2)

Place and object onto the screen

```
void swf_placeobject  (int objid, int depth)
```

Places the object specified by *objid* in the current frame at a depth of *depth*. The *objid* parameter and the *depth* must be between 1 and 65535.

This uses the current mulcolor (specified by **swf_mulcolor()**) and the current addcolor (specified by **swf_addcolor()** to color the object and it uses the current matrix to position the object.

Note: Full RGBA colors are supported.

swf_modifyobject (PHP4 >= 4.0RC2)

Modify an object.

```
void swf_modifyobject  (int depth, int how)
```

Updates the position and/or color of the object at the specified depth, *depth*. The parameter *how* determines what is updated. *how* can either be the constant MOD_MATRIX or MOD_COLOR or it can be a combination of both (MOD_MATRIX|MOD_COLOR).

MOD_COLOR uses the current mulcolor (specified by the function **swf_mulcolor()**) and addcolor (specified by the function **swf_addcolor()**) to color the object. MOD_MATRIX uses the current matrix to position the object.

swf_removeobject (PHP4 >= 4.0RC2)

Remove an object

```
void swf_removeobject  (int depth)
```

Removes the object at the depth specified by *depth*.

swf_nextid (PHP4 >= 4.0RC2)

Returns the next free object id

```
int swf_nextid (void);
```

The **swf_nextid()** function returns the next available object id.

swf_startdoaction (PHP4 >= 4.0RC2)

Start a description of an action list for the current frame

```
void swf_startdoaction (void);
```

The **swf_startdoaction()** function starts the description of an action list for the current frame. This must be called before actions are defined for the current frame.

swf_actiongotoframe (PHP4 >= 4.0RC2)

Play a frame and then stop

```
void swf_actiongotoframe  (int framenumber)
```

The **swf_actionGotoFrame()** function will go to the frame specified by *framenumber*, play it, and then stop.

swf_actiongeturl (PHP4 >= 4.0RC2)

Get a URL from a Shockwave Flash movie

```
void swf_actiongeturl  (string url, string target)
```

The **swf_actionGetUrl()** function gets the URL specified by the parameter *url* with the target *target*.

swf_actionnextframe (PHP4 >= 4.0RC2)

Go foward one frame

```
void swf_actionnextframe (void);
```

Go foward one frame.

swf_actionprevframe (PHP4 >= 4.0RC2)

Go backwards one frame

```
void swf_actionprevframe (void);
```

swf_actionplay (PHP4 >= 4.0RC2)

Start playing the flash movie from the current frame

```
void swf_actionplay (void);
```

Start playing the flash movie from the current frame.

swf_actionstop (PHP4 >= 4.0RC2)

Stop playing the flash movie at the current frame

```
void swf_actionstop (void);
```

Stop playing the flash movie at the current frame.

swf_actiontogglequality (PHP4 >= 4.0RC2)

Toggle between low and high quality

```
void swf_actiontogglequality (void);
```

Toggle the flash movie between high and low quality.

swf_actionwaitforframe (PHP4 >= 4.0RC2)

Skip actions if a frame has not been loaded

```
void swf_actionwaitforframe  (int framenumber, int skipcount)
```

The **swf_actionWaitForFrame()** function will check to see if the frame, specified by the *framenumber* parameter has been loaded, if not it will skip the number of actions specified by the *skipcount* parameter. This can be useful for "Loading..." type animations.

swf_actionsettarget (PHP4 >= 4.0RC2)

Set the context for actions

```
void swf_actionsettarget  (string target)
```

The **swf_actionSetTarget()** function sets the context for all actions. You can use this to control other flash movies that are currently playing.

swf_actiongotolabel (PHP4 >= 4.0RC2)

Display a frame with the specified label

```
void swf_actiongotolabel (string label)
```

The **swf_actionGotoLabel()** function displays the frame with the label given by the *label* parameter and then stops.

swf_enddoaction (PHP4 >= 4.0RC2)

End the current action

```
void swf_enddoaction (void);
```

Ends the current action started by the **swf_startdoaction()** function.

swf_defineline (PHP4 >= 4.0RC2)

Define a line

```
void swf_defineline (int objid, float x1, float y1, float x2, float y2, float width)
```

The **swf_defineline()** defines a line starting from the x coordinate given by *x1* and the y coordinate given by *y1* parameter. Up to the x coordinate given by the *x2* parameter and the y coordinate given by the *y2* parameter. It will have a width defined by the *width* parameter.

swf_definerect (PHP4 >= 4.0RC2)

Define a rectangle.

```
void swf_definerect (int objid, float x1, float y1, float x2, float y2, float width)
```

The **swf_definerect()** defines a rectangle with an upper left hand coordinate given by the x, *x1*, and the y, *y1*. And a lower right hand coordinate given by the x coordinate, *x2*, and the y coordinate, *y2* . Width of the rectangles border is given by the *width* parameter, if the width is 0.0 then the rectangle is filled.

swf_definepoly (unknown)

Define a polygon

```
void swf_definepoly  (int objid, array coords, int npoints, float width)
```

The **swf_definepoly()** function defines a polygon given an array of x, y coordinates (the coordinates are defined in the parameter *coords*). The parameter *npoints* is the number of overall points that are contained in the array given by *coords*. The *width* is the width of the polygon's border, if set to 0.0 the polygon is filled.

swf_startshape (PHP4 >= 4.0RC2)

Start a complex shape

```
void swf_startshape  (int objid)
```

The **swf_startshape()** function starts a complex shape, with an object id given by the *objid* parameter.

swf_shapelinesolid (PHP4 >= 4.0RC2)

Set the current line style

```
void swf_shapelinesolid  (float r, float g, float b, float a, float width)
```

The **swf_shapeLineSolid()** function sets the current line style to the color of the *rgba* parameters and width to the *width* parameter. If 0.0 is given as a width then no lines are drawn.

swf_shapefilloff (PHP4 >= 4.0RC2)

Turns off filling

```
void swf_shapefilloff (void);
```

The **swf_shapeFillOff()** function turns off filling for the current shape.

swf_shapefillsolid (PHP4 >= 4.0RC2)

Set the current fill style to the specified color

```
void swf_shapefillsolid  (float r, float g, float b, float a)
```

The **swf_shapeFillSolid()** function sets the current fill style to solid, and then sets the fill color to the values of the *rgba* parameters.

swf_shapefillbitmaptile (PHP4 >= 4.0RC2)

Set current fill mode to clipped bitmap

```
void swf_shapefillbitmapclip  (int bitmapid)
```

Sets the fill to bitmap clipped, empty spaces will be filled by the bitmap given by the *bitmapid* parameter.

swf_shapefillbitmaptile (PHP4 >= 4.0RC2)

Set current fill mode to tiled bitmap

```
void swf_shapefillbitmaptile  (int bitmapid)
```

Sets the fill to bitmap tile, empty spaces will be filled by the bitmap given by the *bitmapid* parameter (tiled).

swf_shapemoveto (PHP4 >= 4.0RC2)

Move the current position

```
void swf_shapemoveto  (float x, float y)
```

The **swf_shapeMoveTo()** function moves the current position to the x coordinate given by the *x* parameter and the y position given by the *y* parameter.

swf_shapelineto (PHP4 >= 4.0RC2)

Draw a line

```
void swf_shapelineto  (float x, float y)
```

The **swf_shapeLineTo()** draws a line to the x,y coordinates given by the *x* parameter & the *y* parameter. The current position is then set to the x,y parameters.

swf_shapecurveto (PHP4 >= 4.0RC2)

Draw a quadratic bezier curve between two points

```
void swf_shapecurveto  (float x1, float y1, float x2, float y2)
```

The **swf_shapecurveto()** function draws a quadratic bezier curve from the x coordinate given by *x1* and the y coordinate given by *y1* to the x coordinate given by *x2* and the y coordinate given by *y2*. The current position is then set to the x,y coordinates given by the *x2* and *y2* parameters

swf_shapecurveto3 (PHP4 >= 4.0RC2)

Draw a cubic bezier curve

```
void swf_shapecurveto3  (float x1, float y1, float x2, float y2, float x3, float y3)
```

Draw a cubic bezier curve using the x,y coordinate pairs *x1*, *y1* and *x2*,*y2* as off curve control points and the x,y coordinate *x3*, *y3* as an endpoint. The current position is then set to the x,y coordinate pair given by *x3*,*y3*.

swf_shapearc (PHP4 >= 4.0RC2)

Draw a circular arc

```
void swf_shapearc  (float x, float y, float r, float ang1, float ang2)
```

The **swf_shapeArc()** function draws a circular arc from angle A given by the *ang1* parameter to angle B given by the *ang2* parameter. The center of the circle has an x coordinate given by the *x* parameter and a y coordinate given by the *y*, the radius of the circle is given by the *r* parameter.

swf_endshape (PHP4 >= 4.0RC2)

Completes the definition of the current shape

```
void swf_endshape (void);
```

The **swf_endshape()** completes the definition of the current shape.

swf_definefont (PHP4 >= 4.0RC2)

Defines a font

```
void swf_definefont  (int fontid, string fontname)
```

The **swf_definefont()** function defines a font given by the *fontname* parameter and gives it the id specified by the *fontid* parameter. It then sets the font given by *fontname* to the current font.

swf_setfont (PHP4 >= 4.0RC2)

Change the current font.

```
void swf_setfont  (int fontid)
```

The **swf_setfont()** sets the current font to the value given by the *fontid* parameter.

swf_fontsize (PHP4 >= 4.0RC2)

Change the font size

```
void swf_fontsize  (float size)
```

The **swf_fontsize()** function changes the font size to the value given by the *size* parameter.

swf_fontslant (PHP4 >= 4.0RC2)

Set the font slant

```
void swf_fontslant  (float slant)
```

Set the current font slant to the angle indicated by the *slant* parameter. Positive values create a foward slant, negative values create a negative slant.

swf_fonttracking (PHP4 >= 4.0RC2)

Set the current font tracking

```
void swf_fonttracking  (float tracking)
```

Set the font tracking to the value specified by the *tracking* parameter. This function is used to increase the spacing between letters and text, positive values increase the space and negative values decrease the space between letters.

swf_getfontinfo (PHP4 >= 4.0RC2)

The height in pixels of a capital A and a lowercase x

```
array swf_getfontinfo (void);
```

The **swf_getfontinfo()** function returns an associative array with the following parameters:

- Aheight - The height in pixels of a capital A.
- xheight - The height in pixels of a lowercase x.

swf_definetext (PHP4 >= 4.0RC2)

Define a text string

```
void swf_definetext  (int objid, string str, int docenter)
```

Define a text string (the *str* parameter) using the current font and font size. The *docenter* is where the word is centered, if *docenter* is 1, then the word is centered in x.

swf_textwidth (PHP4 >= 4.0RC2)

Get the width of a string

```
float swf_textwidth  (string str)
```

The **swf_textwidth()** function gives the width of the string, *str*, in pixels, using the current font and font size.

swf_definebitmap (PHP4 >= 4.0RC2)

Define a bitmap

```
void swf_definebitmap  (int objid, string image_name)
```

The **swf_definebitmap()** function defines a bitmap given a GIF, JPEG, RGB or FI image. The image will be converted into a Flash JPEG or Flash color map format.

swf_getbitmapinfo (PHP4 >= 4.0RC2)

Get information about a bitmap

```
array swf_getbitmapinfo  (int bitmapid)
```

The **swf_getbitmapinfo()** function returns an array of information about a bitmap given by the *bitmapid* parameter. The returned array has the following elements:

- "size" - The size in bytes of the bitmap.
- "width" - The width in pixels of the bitmap.
- "height" - The height in pixels of the bitmap.

swf_startsymbol (PHP4 >= 4.0RC2)

Define a symbol

```
void swf_startsymbol  (int objid)
```

Define an object id as a symbol. Symbols are tiny flash movies that can be played simultaneously. The *objid* parameter is the object id you want to define as a symbol.

swf_endsymbol (PHP4 >= 4.0RC2)

End the definition of a symbol

```
void swf_endsymbol (void);
```

The **swf_endsymbol()** function ends the definition of a symbol that was started by the **swf_startsymbol()** function.

swf_startbutton (PHP4 >= 4.0RC2)

Start the definition of a button

```
void swf_startbutton  (int objid, int type)
```

The **swf_startbutton()** function starts off the definition of a button. The *type* parameter can either be TYPE_MENUBUTTON or TYPE_PUSHBUTTON. The TYPE_MENUBUTTON constant allows the focus to travel from the button when the mouse is down, TYPE_PUSHBUTTON does not allow the focus to travel when the mouse is down.

swf_addbuttonrecord (PHP4 >= 4.0RC2)

Controls location, appearance and active area of the current button

```
void swf_addbuttonrecord  (int states, int shapeid, int depth)
```

The **swf_addbuttonrecord()** function allows you to define the specifics of using a button. The first parameter, *states*, defines what states the button can have, these can be any or all of the following constants: BSHitTest, BSDown, BSOver or BSUp. The second parameter, the *shapeid* is the look of the button, this is usually the object id of the shape of the button. The *depth* parameter is the placement of the button in the current frame.

Example 1. swf_addbuttonrecord() function example

```
swf_startButton ($objid, TYPE_MENUBUTTON);
  swf_addButtonRecord (BSDown|BSOver, $buttonImageId, 340);
  swf_onCondition (MenuEnter);
    swf_actionGetUrl ("http://www.designmultimedia.com", "_level1");
  swf_onCondition (MenuExit);
    swf_actionGetUrl ("", "_level1");
swf_endButton ();
```

swf_oncondition (PHP4 >= 4.0RC2)

Describe a transition used to trigger an action list

```
void swf_oncondition  (int transition)
```

The **swf_onCondition()** function describes a transition that will trigger an action list. There are several types of possible transitions, the following are for buttons defined as TYPE_MENUBUTTON:

- IdletoOverUp
- OverUptoIdle
- OverUptoOverDown
- OverDowntoOverUp
- IdletoOverDown
- OutDowntoIdle
- MenuEnter (IdletoOverUp|IdletoOverDown)

- MenuExit (OverUptoIdle|OverDowntoIdle)

For TYPE_PUSHBUTTON there are the following options:

- IdletoOverUp
- OverUptoIdle
- OverUptoOverDown
- OverDowntoOverUp
- OverDowntoOutDown
- OutDowntoOverDown
- OutDowntoIdle
- ButtonEnter (IdletoOverUp|OutDowntoOverDown)
- ButtonExit (OverUptoIdle|OverDowntoOutDown)

swf_endbutton (PHP4 >= 4.0RC2)

End the definition of the current button

```
void swf_endbutton (void);
```

The **swf_endButton()** function ends the definition of the current button.

swf_viewport (PHP4 >= 4.0RC2)

Select an area for future drawing

```
void swf_viewport  (double xmin, double xmax, double ymin, double ymax)
```

The **swf_viewport()** function selects an area for future drawing for *xmin* to *xmax* and *ymin* to *ymax*, if this function is not called the area defaults to the size of the screen.

swf_ortho (unknown)

Defines an orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho  (double xmin, double xmax, double ymin, double ymax, double
zmin, double zmax)
```

The **swf_ortho()** funcion defines a orthographic mapping of user coordinates onto the current viewport.

swf_ortho2 (PHP4 >= 4.0RC2)

Defines 2D orthographic mapping of user coordinates onto the current viewport

```
void swf_ortho2  (double xmin, double xmax, double ymin, double ymax)
```

The **swf_ortho2()** function defines a two dimensional orthographic mapping of user coordinates onto the current viewport, this defaults to one to one mapping of the area of the Flash movie. If a perspective transformation is desired, the **swf_perspective ()** function can be used.

swf_perspective (PHP4 >= 4.0RC2)

Define a perspective projection transformation

```
void swf_perspective  (double fovy, double aspect, double near, double far)
```

The **swf_perspective()** function defines a perspective projection transformation. The *fovy* parameter is field-of-view angle in the y direction. The *aspect* parameter should be set to the aspect ratio of the viewport that is being drawn onto. The *near* parameter is the near clipping plane and the *far* parameter is the far clipping plane.

> **Note:** Various distortion artifacts may appear when performing a perspective projection, this is because Flash players only have a two dimensional matrix. Some are not to pretty.

swf_polarview (PHP4 >= 4.0RC2)

Define the viewer's position with polar coordinates

```
void swf_polarview  (double dist, double azimuth, double incidence, double
twist)
```

The **swf_polarview()** function defines the viewer's position in polar coordinates. The *dist* parameter gives the distance between the viewpoint to the world space origin. The *azimuth* parameter defines the azimuthal angle in the x,y coordinate plane, measured in distance from the y axis. The *incidence* parameter defines the angle of incidence in the y,z plane, measured in distance from the z axis. The incidence angle is defined as the angle of the viewport relative to the z axis. Finally the *twist* specifies the amount that the viewpoint is to be rotated about the line of sight using the right hand rule.

swf_lookat (PHP4 >= 4.0RC2)

Define a viewing transformation

```
void swf_lookat  (double view_x, double view_y, double view_z, double
reference_x, double reference_y, double reference_z, double twist)
```

The **swf_lookat()** function defines a viewing transformation by giving the viewing position (the parameters *view_x*, *view_y*, and *view_z*) and the coordinates of a reference point in the scene, the reference point is defined by the *reference_x*, *reference_y* , and *reference_z* parameters. The *twist* controls the rotation along with viewer's z axis.

swf_pushmatrix (PHP4 >= 4.0RC2)

Push the current transformation matrix back unto the stack

```
void swf_pushmatrix (void);
```

The **swf_pushmatrix()** function pushes the current transformation matrix back onto the stack.

swf_popmatrix (PHP4 >= 4.0RC2)

Restore a previous transformation matrix

```
void swf_popmatrix (void);
```

The **swf_popmatrix()** function pushes the current transformation matrix back onto the stack.

swf_scale (PHP4 >= 4.0RC2)

Scale the current transformation

```
void swf_scale  (double x, double y, double z)
```

The **swf_scale()** scales the x coordinate of the curve by the value of the *x* parameter, the y coordinate of the curve by the value of the *y* parameter, and the z coordinate of the curve by the value of the *z* parameter.

swf_translate (PHP4 >= 4.0RC2)

Translate the current transformations

```
void swf_translate  (double x, double y, double z)
```

The **swf_translate()** function translates the current transformation by the *x*, *y*, and *z* values given.

swf_rotate (PHP4 >= 4.0RC2)

Rotate the current transformation

```
void swf_rotate  (double angle, string axis)
```

The **swf_rotate()** rotates the current transformation by the angle given by the *angle* parameter around the axis given by the *axis* parameter. Valid values for the axis are 'x' (the x axis), 'y' (the y axis) or 'z' (the z axis).

swf_posround (PHP4 >= 4.0RC2)

Enables or Disables the rounding of the translation when objects are placed or moved

```
void swf_posround  (int round)
```

The **swf_posround()** function enables or disables the rounding of the translation when objects are placed or moved, there are times when text becomes more readable because rounding has been enabled. The *round* is whether to enable rounding or not, if set to the value of 1, then rounding is enabled, if set to 0 then rounding is disabled.

LVII. Sybase functions

sybase_affected_rows (PHP3 >= 3.0.6, PHP4)

get number of affected rows in last query

```
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 (PHP3 , PHP4)

close Sybase connection

```
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 (PHP3 , PHP4)

open Sybase server connection

```
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 (PHP3 , PHP4)

move internal row pointer

```
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 (PHP3 , PHP4)

fetch row as array

```
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 (PHP3 , PHP4)

get field information

```
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 (PHP3 , PHP4)

fetch row as object

```
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 (PHP3 , PHP4)

get row as enumerated array

```
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 (PHP3 , PHP4)

set field offset

```
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 (PHP3 , PHP4)

free result memory

```
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 (PHP3 , PHP4)

get number of fields in result

```
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 (PHP3 , PHP4)

get number of rows in result

```
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 (PHP3 , PHP4)

open persistent Sybase connection

```
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 (PHP3 , PHP4)

send Sybase query

```
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 (PHP3 , PHP4)

get result data

```
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 (PHP3 , PHP4)

select Sybase database

```
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()**

LVIII. URL Functions

base64_decode (PHP3 , PHP4)

Decodes data encoded with MIME base64

```
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 (PHP3 , PHP4)

Encodes data with MIME base64

```
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 (PHP3 , PHP4)

Parse a URL and return its components

```
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 (PHP3 , PHP4)

Decodes URL-encoded string

```
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 (PHP3 , PHP4)

URL-encodes string

```
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()**.

LIX. Variable Functions

doubleval (PHP3 , PHP4)

Get double value of a variable

```
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.

```
$var = '122.34343The';
$double_value_of_var = doubleval ($var);
print $double_value_of_var; // prints 122.34343
```

See also **intval()**, **strval()**, **settype()** and Type juggling.

empty (unknown)

Determine whether a variable is set

```
int empty (mixed var)
```

Returns false if *var* is set and has a non-empty or non-zero value; true otherwise.

```
$var = 0;
if (empty($var)) { #evaluates true
    print '$var is either 0 or not at all set';
}
if (!isset($var)) { // evaluates false
    print 'The $var is not set at all';
}
```

Note that this is meaningless when used on anything which isn't a variable; i.e. **empty (addslashes ($name))** has no meaning since it would be checking whether something which isn't a variable is a variable with a false value.

See also **isset()** and **unset()**.

gettype (PHP3 , PHP4)

Get the type of a variable

```
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 (PHP3 , PHP4)

Get integer value of a variable

```
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 (PHP3 , PHP4)

Finds whether a variable is an array

```
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 (PHP3 , PHP4)

Finds whether a variable is a double

```
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 (PHP3 , PHP4)

Finds whether a variable is a float

```
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 (PHP3 , PHP4)

Find whether a variable is an integer

```
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 (PHP3 , PHP4)

Find whether a variable is an integer

```
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 (PHP3 , PHP4)

Finds whether a variable is an integer

```
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_numeric (PHP4 >= 4.0RC1)

Finds whether a variable is a number or a numeric string

```
int is_numeric (mixed var)
```

Returns true if *var* is a number or a numeric string, false otherwise.

See also **is_double()**, **is_float()**, **is_int()**, **is_real()**, **is_string()**, **is_object()**, **is_array()**, and **is_integer()**.

is_object (PHP3 , PHP4)

Finds whether a variable is an object

```
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 (PHP3 , PHP4)

Finds whether a variable is a real

```
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 (PHP3 , PHP4)

Finds whether a variable is a string

```
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 (unknown)

Determine whether a variable is set

```
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()**.

print_r (PHP4)

Prints human-readable information about a variable

```
void print_r (mixed expression)
```

This function displays information about the values of variables in a way that's readable by humans. If given a string, integer or double, the value itself will be printed. If given an array, values will be presented in a format that shows keys and elements. Similar notation is used for objects.

Compare **print_r()** to **var_dump()**.

```
<?php
$a = array (1, 2, array ("a", "b", "c"));
print_r ($a);
?>
```

settype (PHP3 , PHP4)

Set the type of a variable

```
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 (PHP3 , PHP4)

Get string value of a variable

```
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 (unknown)

Unset a given variable

```
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()**.

var_dump (PHP3 >= 3.0.5, PHP4)

Dumps information about a variable

```
void var_dump (mixed expression)
```

This function returns structured information about an expression that includes its type and value. Arrays are explored recursively with values indented to show structure.

Compare **var_dump()** to **print_r()**.

```
<pre>
<?php
    $a = array (1, 2, array ("a", "b", "c"));
```

```
    var_dump ($a);
?>
</pre>
```

LX. 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 (PHP3 , PHP4)

Add a new virtual user with a password

```
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 (PHP3 , PHP4)

Add an alias to a virtual user

```
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 (PHP3 , PHP4)

Changes a virtual users password

```
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 (PHP3 , PHP4)

Removes an alias

```
int vm_delalias (string vdomain, string basepwd, string alias)
```

Removes an alias.

vm_deluser (PHP3 , PHP4)

Removes a virtual user

```
int vm_deluser (string vdomain, string username)
```

Removes a virtual user..

LXI. 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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Serialize a single value into a WDDX packet

```
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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Serialize variables into a WDDX packet

```
string wddx_serialize_vars (mixed var_name [, mixed ...])
```

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 ver-
sion='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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Starts a new WDDX packet with structure inside it

```
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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Ends a WDDX packet with the specified ID

```
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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Ends a WDDX packet with the specified ID

```
wddx_add_vars (int packet_id, mixed name_var [, mixed ...])
```

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 (PHP3 >= 3.0.7, PHP4 >= 4.0b2)

Deserializes a WDDX packet

```
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.

LXII. 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.

PHP function to set handler	Event description
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, $openEntityNames, $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 (PHP3 >= 3.0.6, PHP4)

create an XML parser

```
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_object (PHP4 >= 4.0b4)

Use XML Parser withing an object

```
void xml_set_object (int parser, object &object)
```

This function makes *parser* useable from within *object*. All callback functions settet via **xml_set_element_handler()** etc are assumed to be methods of *object*.

```
<?php
class xml  {
var $parser;

function xml() {
    $this->parser = xml_parser_create();
    xml_set_object($this->parser,&$this);
    xml_set_element_handler($this->parser,"tag_open","tag_close");
    xml_set_character_data_handler($this->parser,"cdata");
}

function parse($data) {
    xml_parse($this->parser,$data);
}

function tag_open($parser,$tag,$attributes) {
    var_dump($parser,$tag,$attributes);
}

function cdata($parser,$cdata) {
    var_dump($parser,$cdata);
}

function tag_close($parser,$tag) {
```

```
    var_dump($parser,$tag);
}

} // end of class xml

$xml_parser = new xml();
$xml_parser->parse("<A ID=\"hallo\">PHP</A>");
?>
```

Note: xml_set_object() handling was added in PHP 4.0.

xml_set_element_handler (PHP3 >= 3.0.6, PHP4)

set up start and end element handlers

```
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 (PHP3 >= 3.0.6, PHP4)

set up character data handler

```
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 (PHP3 >= 3.0.6, PHP4)

Set up processing instruction (PI) handler

```
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 (PHP3 >= 3.0.6, PHP4)

set up default handler

```
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 (PHP3 >= 3.0.6, PHP4)

Set up unparsed entity declaration handler

```
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 (PHP3 >= 3.0.6, PHP4)

set up notation declaration handler

```
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 (PHP3 >= 3.0.6, PHP4)

set up external entity reference handler

```
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 (PHP3 >= 3.0.6, PHP4)

start parsing an XML document

```
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 (PHP3 >= 3.0.6, PHP4)

get XML parser error code

```
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 (PHP3 >= 3.0.6, PHP4)

get XML parser error string

```
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 (PHP3 >= 3.0.6, PHP4)

get current line number for an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

Get current column number for an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

get current byte index for an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

Free an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

set options in an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

get options from an XML parser

```
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 (PHP3 >= 3.0.6, PHP4)

Converts a string with ISO-8859-1 characters encoded with UTF-8 to single-byte ISO-8859-1.

```
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 (PHP3 >= 3.0.6, PHP4)

encodes an ISO-8859-1 string to UTF-8

```
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 DOUBLE de-
pending 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)
- SET_VAR_DOUBLE(name,value)
- SET_VAR_LONG(name,value)

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 f
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* .

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
```