PHP3 Manual

Stig Sæther Bakken

Alexander Aulbach

Egon Schmid

Jim Winstead

Lars Torben Wilson

Rasmus Lerdorf

Zeev Suraski

PHP3 Manual

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

1998-04-05

Revision History

Revision PHP 3.0RC3	1998-04-05

Edited by Stig Sæther Bakken

Table of Contents

List of Tables

List of Examples

Preface

PHP Version 3.0 is an HTML-embedded scripting language. Much of its syntax is borrowed from C, Java and Perl with a couple of unique PHP-specific features thrown in. The goal of the language is to allow web developers to write dynamically generated pages quickly.

About this Manual

This manual is written in SGML using the DocBook DTD, using DSSSL (Document Style and Semantics Specification Language) for formatting. The tools used for formatting HTML, TeX and RTF versions are Jade, written by James Clark and The Modular DocBook Stylesheets written by Norman Walsh. PHP3's documentation framework was assembled by Stig Sæther Bakken.

I. Language Reference

Chapter 1. An introduction to PHP3

What is PHP3?

PHP Version 3.0 is a server-side HTML-embedded scripting language.

What can PHP3 do?

Perhaps the strongest and most significant feature in PHP3 is its database integration layer. Writing a database-enabled web page is incredibly simple. The following databases are currently supported:

- Oracle
- Sybase
- mSQL 1.x and 2.x
- MySQL
- Solid
- Generic ODBC
- PostgreSQL
- Adabas D
- FilePro
- Velocis
- dBase
- Unix dbm

PHP3 Concepts

Some examples

Chapter 2. PHP3 features

HTTP authentication with PHP

The HTTP Authentication hooks in PHP are only available when it is running as an Apache module. 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.

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

Example 2-1. HTTP Authentication example

```
<?php
  if(!$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.

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.

GIF creation with PHP

File upload support

HTTP cookie support

Database support

Regular expressions

Error handling

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

Supressing errors

PHP source viewer

Chapter 3. Installation

This chapter will guide you through the configuration and installation of PHP3. Prerequisite knowledge and software:

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

Installing From Source on UNIX

Downloading Source

The source code for the latest version can be found at `http://www.php.net`.

Configuration

There are two ways of configuring PHP3.

- Using the "setup" script that comes with PHP3. 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 PHP3 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 PHP3 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 PHP3 as a CGI program. If you are running a web server PHP3 has module support for, you should generally go for that solution for performance reasons. However, the CGI version enables Apache users to run different PHP3-enabled pages under different user-ids.

Database Support Options

PHP3 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

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 PHP3 supports both 1.0 and 2.0, but if you compile PHP3 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

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

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

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

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

Solid

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

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

Solid home page

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

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

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 and Adabas D. Requires that one (and only one) of these modules or the Velocis module is enabled. This option is only applicable if one of the following options is used: --with-iodbc, --with-solid, --with-adabas, or --with-velocis.

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 and RFC1778.

Other configure options

--enable-maintainer-mode

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

Turns on extra dependencies and compiler warnings used by some of the PHP3 developers.

--with-system-regex

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

Uses the system's regular expression library rather than the bundled one. If you are building PHP3 as a server module, you must use the same library when building PHP3 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 php3.ini 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.

--disable-debug

```
--disable-debug
```

Does not include debug information in the library or executable. The debug information makes it easier to pinpoint bugs, so it is a good idea to leave debug on as long as PHP3 is in alpha or beta state.

--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 PHP3 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 PHP3 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 `<? ?>` PHP3 tags. You must disable the short form if you want to use PHP3 with XML. With short tags disabled, the only PHP3 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 PHP3 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 PHP3 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 PHP3 is configured, you are ready to build the CGI executable or the PHP3 library. The command **make** should take care of this. If it fails and you can't figure out why, see the Problems section.

VPATH

Testing

If you have built PHP3 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 PHP3 on your platform early instead of having to struggle with it later.

Benchmarking

If you have built PHP3 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 to control this time for you own scripts. **make bench** ignores the configuration file.

Installing PHP on Windows95/NT

Apache/NT and Stronghold/NT

Follwo the instructions for configuration under Unix.

IIS and MS-PWS

You can access php scripts simply by putting the php.exe file into your scripts directory and using a url such as: `http://my.server/scripts/php.exe/page.php`

Redirection If you would like to use a url like: `http://my.server/page.php` you will have to edit your registry.

> **NOTE:** Disclaimer: Be sure you make a backup of your registry before editing it. The PHP Development Team is not responsible for damaged registries. If you damage your registry, you may not be able to restart your computer without reinstalling your OS!

You can edit your registry by running regedit.exe. To do this, choose Run... from the Start menu, and type **regedit** then click on OK. The registry setting you need to edit is: HKEY_LOCAL_MACHINE:System:CurrentControlSet:Services:W3Svc:Parameters:ScriptMap. Add a new string value here with the extension you would like to use for your php scripts, and make the value data the path to the php executable: `.phtm3 "c:\webshare\scripts\php.exe"`

For the ISAPI version of PHP, use something like: `.phtm "c:\webshare\scripts\php3_isapi.dll"`

You must also make any directories containing php scripts executable. This is done in the IIS administrator. Consult your IIS documentation for more information.

For other servers consult your server documentation.

PHP.INI File Under Windows, PHP will look for php3.ini automaticaly, first under the windows os directory (`c:\windows` or `c:\winnt`) then in the directory in which the PHP executable resides. Alternately, you can set the environment variable PHPRC=\pathto\php3.ini, though this does not work with all servers (apache for one).

Problems?

Read the FAQ

Some problems are more common than others. The most common ones are listed in the PHP3 FAQ, found at http://www.php.net/FAQ.php3

Bug reports

If you think you have found a bug in PHP3, please report it. The PHP3 developers probably don't know about it, and unless you report it, chances are it won't be fixed. A form for reporting bugs is available on the PHP3 network of sites, the main form is at http://ca.php.net/bugs.php3.

Other problems

If you are still stuck, someone on the PHP3 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 archive is available at http://www.tryc.on.ca/php3.html. To subscribe to the PHP3 mailing list, send an empty mail to php3-subscribe@php.il.eu.org. The mailing list address is `php3@php.il.eu.org`.

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.

Security

PHP is a powerful tool. As with many other powerful tools, it is possible to shoot yourself in the foot with it. PHP has functionality that, if carelessly used, may cause security problems on your system. The best way of preventing this is to always know what you are doing. Read the Security chapter for more information.

Chapter 4. Configuration

The php3.ini file

The `php3.ini` file is read when PHP's parser 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.

Just about every directive listed here has a corresponding Apache `httpd.conf` directive. Simply prepend *php3_* to the start of the directive names listed here.

General Configuration Directives

`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 4-1. 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).

`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.

`include_path` string

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

Example 4-1. UNIX include_path

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

Example 4-2. 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

`magic_quotes_runtime` boolean

`magic_quotes_sybase` boolean

`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 users can normally set it to `/var/qmail/bin/sendmail`.

Safe Mode Configuration Directives

`safe_mode` boolean

Whether to enable PHP's safe mode.

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

extension_dir string

In what directory PHP should look for dynamically loadable extensions.

extension string

Which dynamically loadable extensions to load when PHP starts up.

MySQL Configuration Directives

mysql.allow_persistent boolean

Whether to allow persistent MySQL connections.

mysql.max_persistent integer

The maximum number of persistent MySQL connections per process.

mysql.max_links integer

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

mSQL Configuration Directives

msql.allow_persistent boolean

Whether to allow persistent mSQL connections.

msql.max_persistent integer

The maximum number of persistent mSQL connections per process.

msql.max_links integer

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

Postgres Configuration Directives

pgsql.allow_persistent boolean

Whether to allow persistent Postgres connections.

pgsql.max_persistent integer

The maximum number of persistent Postgres connections per process.

pgsql.max_links integer

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

Sybase Configuration Directives

sybase.allow_persistent boolean

Whether to allow persistent Sybase connections.

sybase.max_persistent integer

The maximum number of persistent Sybase connections per process.

sybase.max_links integer

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

Sybase-CT Configuration Directives

sybct.allow_persistent boolean

Whether to allow persistent Sybase-CT connections.

sybct.max_persistent integer

The maximum number of persistent Sybase-CT connections per process.

sybct.max_links integer

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

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.

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.

Apache Module

Apache module configuration directives

CGI redirection module/action module

CGI

Virtual hosts

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 recommends agains 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 an 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 explanation of 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 with 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 for example in apache by directives AddHandler and Action (see below).

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

This compile time option to configure script will produce a PHP binary that when used as a CGI will refuse to work if the request is not redirected by the web server, for example Apache. This prevents anyone to call PHP directly with an url like `http://my.host/cgi-bin/php/secretdir/script.php3`.

Usually the redirection in apache configuration is done with the following statements:

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

This option is tested only with 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 either to open all your documents for public use or use doc_root or user_dir runtime configuration directives described below.

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 for some configuration mistake the scripts are not executed but displayed as usual 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 is only accessible 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 php3.ini 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

Chapter 5. Syntax and grammar

PHP's syntax is borrowed primarily from C. Java and Perl have also influenced the syntax.

Escaping from HTML

There are three 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>
```

Instruction separation

Variable types

Variable initialization

Variable Scope

The scope of a variable is the context within which it is defined. For the most part all PHP variables only have a single scope. However, within user-defined functions a local function scope is introduced. Any variable used inside a function is by default limited to the local function scope. For example:

```
    $a=1; /* global scope */
    Function Test() {
        echo $a; /* reference to local scope variable */
    }
    Test();
```

This script will not produce any output because the echo statement refers to a local version of the $a variable, and it has not been assigned a value within this scope. You may notice that this is a little bit different from the C language in that global variables in C are automatically available to functions unless specifically overridden by a local definition. This can cause some problems in that people may inadvertently change a global variable. In PHP global variables must be declared global inside a function if they are going to be used in that function. An example:

```
    $a=1;
```

```
$b=2;
Function Sum() {
    global $a,$b;

    $b = $a + $b;
}
Sum();
echo $b;
```

The above script will output "3". By declaring $a and $b global within the function, all references to either variable will refer to the global version. There is no limit to the number of global variables that can be manipulated by a function.

A second way to access variables from the global scope is to use the special PHP-defined $GLOBALS array. The previous example can be rewritten as:

```
$a=1;
$b=2;
Function Sum() {
    $GLOBALS["b"] = $GLOBALS["a"] + $GLOBALS["b"];
}
Sum();
echo $b;
```

The $GLOBALS array is an associative array with the name of the global variable being the key and the contents of that variable being the value of the array element.

Another important feature of variable scoping is the *static* variable. A static variable exists only in a local function scope, but it does not lose its value when program execution leaves this scope. Consider the following example:

```
Function Test() {
    $a=0;
    echo $a;
    $a++;
}
```

This function is quite useless since every time it is called it sets $a to 0 and prints "0". The $a++ which increments the variable serves no purpose since as soon as the function exits the $a variable disappears. To make a useful counting function which will not lose track of the current count, the $a variable is declared static:

```
Function Test() {
    static $a=0;
    echo $a;
    $a++;
}
```

Now, every time the Test() function is called it will print the value of $a and increment it.

Static variables are also essential when functions are called recursively. A recursive function is one which calls itself. Care must be taken when writing a recursive function because it is possible to make it recurse indefinitely. You must make sure you have an adequate way of terminating the recursion. The following simple function recursively counts to 10:

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

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

Variable variables

Sometimes it is convenient to be able to have variable variable names. That is, a variable name which can be set and used dynamically. A normal variable is set with a statement such as:

```
$a = "hello";
```

A variable variable takes the value of a variable and treats that as the name of a variable. In the above example, *hello*, can be used as the name of a variable by using two dollar signs. ie.

```
$$a = "world";
```

At this point two variables have been defined and stored in the PHP symbol tree: $a with contents "hello" and $hello with contents "world". Therefore, this statement:

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

produces the exact same output as:

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

ie. they both produce: *hello world*.

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

Variables from outside PHP

HTML Forms (GET and POST)

HTTP Cookies

PHP transparently supports HTTP cookies as defined by Netscape's Spec. Cookies are a mechanism for storing data in the remote browser and thus tracking or identifying return users. You can set cookies using the **SetCookie** function. Cookies are part of the HTTP header, so the SetCookie function must be called before any output is sent to the browser. This is the same restriction as for the **Header** function. 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 5-2. SetCookie Example

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

Environment variables

Server configuration directives

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 a 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 a double (15)
$foo = 5 + "10 Small Pigs";   // $foo is an 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**.

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**.

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;
```

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.

```
$foo = 1 + "10.5";      // $foo is a double (11.5)
$foo = 1 + "-1.3e3";    // $foo is a double (-1299)
$foo = 1 + "bob-1.3e3"; // $foo is a double (1)
$foo = 1 + "bob3";      // $foo is an integer (1)
$foo = 1 + "10 Small Pigs";     // $foo is an integer (11)
$foo = 1 + "10 Little Piggies"; // $foo is a double (11); the string contains
'e'
```

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

Array manipulation

Chapter 6. Language constructs

Any PHP 3 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.

Expressions

Expressions are the most important building stones of PHP. In PHP 3.0, almost anything you write is an expression. The simplest yet most accurate way to define an expressions is "anything that has a value".

Simple examples that come in mind 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 3 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 3 takes expressions much further, in the same way many other languages do. PHP 3 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 '$a++' has no value (is not an expression), and thus you can't assign it or use it in any way. PHP 3 enhances the increment/decrement capabilities by making these expressions as well, like in C. In PHP 3, like in C, there are two types of increment - pre-increment and post-increment. Both pre-increment and post-increment essentially increment the variable, and the effect on the variable is idential. The difference is with the value of the increment expression. Pre-increment, which is written '++$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), < (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 3, 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.

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, the value assigned to $c is the
original value of $a, which is 5 */
$e = $d = ++$b;     /* pre-increment, the value assigned into $d and $e is
the incremented value of $b, which is 6 */
/* at this point, both $d and $e are equal to 6 */
$f = double($d++);  /* $f would be assigned twice the value of $d *before*
the increment, 2*6 = 12 */
$g = double($++e);  /* $g would be assigned twice the value of $e *after* the
increment, 2*7 = 14 */
$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! Any non-empty string is considered to be TRUE; empty strings are FALSE. With non-scalar values (arrays and objects) - if the value contains no elements it's considered FALSE, otherwise it's considered TRUE.

PHP 3 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 mark any valid PHP3 expression.

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 sentence 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 pgoram.

ELSE

Often you'd want to execute a statement if a certain condition is met, and a different statement if the condition is not met. This is what ELSE is for. ELSE extends an IF statement to execute a statement in case the expression in the IF statement evaluates to FALSE. For example, the following code would display 'a is bigger than b' if $a is bigger than $b, and 'a is NOT bigger than b' otherwise:

```
if ($a>$b) {
    print "a is bigger than b";
} else {
    print "a is NOT bigger than b";
}
```

The ELSE statement is only executed if the IF expression evaluated to FALSE, and if there were any ELSEIF expressions - only if they evaluated to FALSE as well (see below).

ELSEIF

ELSEIF, as its name suggests, is a combination of IF and ELSE. Like ELSE, it extends an IF statement to execute a different statement in case the original IF expression evaluates to FALSE. However, unlike ELSE, it will execute that alternative expression only if the ELSEIF expression evaluates to TRUE. For example, the following code would display 'a is bigger than b' if $a>$b, 'a is equal to b' if $a==$b, and 'a is smaller than b' if $a<$b:

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

There may be several ELSEIFs within the same IF statement. The first ELSEIF expression (if any) that evaluates to TRUE would be executed. In PHP 3, 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 IF expression and any previous ELSEIF expressions evaluated to FALSE, and the current ELSEIF expression evaluated to TRUE.

Alternative syntax for IF statements: IF(): ... ENDIF;

PHP 3 offers a different way to group statements within an IF statement. This is most commonly used when you nest HTML blocks inside IF statements, but can be used anywhere. Instead of using curly braces, the IF(expr) should be followed by a colon, the list of one or more statements, and end with ENDIF;. Consider the following example:

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

In the above example, the HTML block "A = 5" is nested within an IF statement written in the alternative syntax. The HTML block would be displayed only if $a is equal to 5.

The alternative syntax applies to ELSE and ELSEIF (expr) as well. The following is an IF statement with ELSEIF and ELSE in the alternative format:

```
if ($a==5):
    print "a equals to 5";
    print "...";
elseif ($a==6):
    print "a equals to 6";
    print "!!!";
else
    print "a is not 5 nor 6";
endif;
```

WHILE

WHILE loops are the simplest type of loop in PHP 3. 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) 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++;
}
```

Of course, the first example appears to be the nicest one, but you may find that being able to use empty expressions in FOR loops comes in handy in many occasions.

There is only one format for FOR loops in PHP 3.

```
    FOR(expr): ... ENDFOR;  is NOT supported.
```

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:

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

/* example 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 messups. 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, PHP begins 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:

```
/* example 3 */
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).

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

```
/* example 4 */
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";
}
```

Another fact worth mentioning is that the CASE expression may be any expression that evaluates to a scalar type, that is, integer or real numbers and strings. Arrays or objects won't crash PHP, but they're meaningless in that context.

REQUIRE

INCLUDE

FUNCTION

OLD_FUNCTION

CLASS

Chapter 7. Expressions

Mathematical expressions

Operators

=

+=, -=

++, --

Overloaded operators

II. Function Reference

Reference 1. Adabas D Functions

The Adabas D functions are deprecated, you probably want to use the Unified ODBC functions instead.

ada_afetch

Name

`ada_afetch` — fetch a result row into an array

Description

See **odbc_fetch_into**

ada_autocommit

Name

`ada_autocommit` — toggle autocommit behaviour

Description

See **odbc_autocommit**.

ada_close

Name

`ada_close` — close a connection to an Adabas D server

Description

See **odbc_close**.

ada_commit

Name

`ada_commit` — commit a transaction

Description

See **odbc_commit**

ada_connect

Name

`ada_connect` — connect to an Adabas D datasource

Description

See **odbc_connect**.

ada_exec

Name

`ada_exec` — prepare and execute a SQL statement

Description

See **odbc_exec** or **odbc_do**.

ada_fetchrow

Name

`ada_fetchrow` — fetch a row from a result

Description

See **odbc_fetch_row**.

ada_fieldname

Name

`ada_fieldname` — get the columnname

Description

See **odbc_field_name**.

ada_fieldnum

Name

`ada_fieldnum` — get column number

Description

See **odbc_field_num**.

ada_fieldtype

Name

`ada_fieldtype` — get the datatype of a field

Description

See **odbc_field_type**.

ada_freeresult

Name

`ada_freeresult` — >free resources associated with a result

Description

See **odbc_free_result**.

ada_numfields

Name

`ada_numfields` — get the number of columns in a result

Description

See **odbc_num_fields**.

ada_numrows

Name

`ada_numrows` — number of rows in a result

Description

See **odbc_num_rows**.

ada_result

Name

`ada_result` — get data from results

Description

See **odbc_result**.

ada_resultall

Name

`ada_resultall` — print result as HTML table

Description

See **odbc_result_all**.

ada_rollback

Name

`ada_rollback` — rollback a transaction

Description

See **odbc_rollback**.

Reference 2. Array Functions

array

Name

`array` — Create an array

Description

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

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

Note that **array** really 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**.

arsort

Name

`arsort` — Sort an array in reverse order and maintain index association

Description

```
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

Name

`asort` — Sort an array and maintain index association

Description

```
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**.

count

Name

`count` — count elements in array

Description

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

Returns the number of elements in the array.

Returns 0 is the variable is not set.

Returns 1 if the variable is not an array.

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

current

Name

`current` — return the current element in an array

Description

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

Each array variable has an internal pointer that points to one of its elements. In addition, all of the elements in the array are linked by a bidirectional linked list for traversing purposes. The internal pointer points to the first element that was inserted to the array until you run one of the functions that modify that pointer on that 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.

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

end

Name

`end` — set internal pointer of array to last element

Description

```
end(array array);
```

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

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

key

Name

`key` — fetch a key from an associative array

Description

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

key returns the index element of the current array position.

See also: **current next**

ksort

Name

`ksort` — Sort an array by key.

Description

```
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

Name

`list` — assign variables as if they were an array

Description

```
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: **array**.

next

Name

`next` — advance the internal array pointer

Description

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

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

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

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

pos

Name

`pos` — return the current element in an array

Description

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

This is an alias for **current**.

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

prev

Name

`prev` — rewind internal array pointer

Description

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

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

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**

reset

Name

`reset` — set internal pointer of array to first element

Description

```
reset(array array);
```

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

See also: **current**, **next prev** and **reset**

rsort

Name

`rsort` — Sort an array in reverse order

Description

```
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); $key = key($fruits); next($fruits)) {
    echo "fruits[$key] = ".$fruits[$key]."\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**, and **sort**.

sizeof

Name

`sizeof` — get size of array

Description

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

Returns the number of elements in the array.

See also: **count**

sort

Name

`sort` — Sort an array

Description

```
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**, and **rsort**.

Reference 3. BC (Arbitrary Precision) Functions

These BC functions are only available if PHP was compiled with the --enable-bcmath configure option.

bcadd

Name

`bcadd` — Add two arbitrary precision numbers.

Description

```
string bcadd(string left operand, string right operand, int scale);
```

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

See also **bcsub**.

bccomp

Name

`bccomp` — Compare two arbitrary precision numbers.

Description

```
int bccomp(string left operand, string right operand, int scale);
```

Compares the *`left operand`* to the *`right operand`* and returns the result as an integer. The optional *`scale`* parameter is used to set the number of digits after the decimal place which will be used in the 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

Name

`bcdiv` — Divide two arbitrary precision numbers.

Description

```
string bcdiv(string left operand, string right operand, int scale);
```

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

See also **bcmul**.

bcmod

Name

`bcmod` — Get modulus of an arbitrary precision number.

Description

```
string bcmod(string left operand, string modulus, int scale);
```

Get the modulus of the *`left operand`* using *`modulus`*. The *`scale`* parameter sets the number of digits after the decimal place in the result.

See also **bcdiv**.

bcmul

Name

`bcmul` — Multiply two arbitrary precision number.

Description

```
string bcmul(string left operand, string right operand, int scale);
```

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

See also **bcdiv**.

bcpow

Name

`bcpow` — Raise an arbitrary precision number to another.

Description

```
string bcpow(string x, string y, int scale);
```

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

See also **bcsqrt**.

bcscale

Name

`bcscale` — Set default scale parameter for all bc math functions.

Description

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

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

bcsqrt

Name

`bcsqrt` — Get the square root of an arbitray precision number.

Description

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

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

See also **bcpow**.

bcsub

Name

`bcsub` — Subtract one arbitrary precision number from another.

Description

```
string bcsub(string left operand, string right operand, int scale);
```

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

See also **bcadd**.

Reference 4. Calendar Functions

The Calendar extension in PHP 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

Name

`JDToGregorian` — Converts Julian Day Count to Gregorian date

Description

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

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

GregorianToJD

Name

`GregorianToJD` — Converts a Gregorian date to Julian Day Count

Description

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

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

Although this 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

Name

`JDToJulian` — Converts a Julian Calendar date to Julian Day Count

Description

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

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

JulianToJD

Name

`JulianToJD` — Converts a Julian Calendar date to Julian Day Count

Description

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

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

Although this 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

Name

`JDToJewish` — Converts a Julian Day Count to the Jewish Calendar

Description

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

Converts a Julian Day Count the the Jewish Calendar.

JewishToJD

Name

`JewishToJD` — Converts a date in the Jewish Calendar to Julian Day Count

Description

```
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

Name

`JDToFrench` — Converts a Julian Day Count to the French Republican Calendar

Description

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

Converts a Julian Day Count to the French Republican Calendar.

FrenchToJD

Name

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

Description

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

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

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

JDMonthName

Name

`JDMonthName` — Returns a month name

Description

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

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

Table 1. Calendar modes

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

JDDayOfWeek

Name

`JDDayOfWeek` — Returns the day of the week

Description

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

Returns the day of the week. Can return a string or an 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 abreviated day of week (english-gregorian)

Reference 5. Date/Time Functions

checkdate

Name

`checkdate` — validate a date/time

Description

```
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 1900 and 32767 inclusive
- month is between 1 and 12 inclusive
- day is within the allowed number of days for the given month. Leap years are taken into consideration.

date

Name

`date` — format a local time/date

Description

```
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:

- U - seconds since the epoch
- Y - year, numeric, 4 digits
- y - year, numeric, 2 digits
- F - month, textual, long; i.e. "January"
- M - month, textual, 3 letters; i.e. "Jan"
- m - month, numeric
- z - day of the year, numeric; i.e. "299"
- d - day of the month, numeric
- l (lowercase 'L') - day of the week, textual, long; i.e. "Friday"
- D - day of the week, textual, 3 letters; i.e. "Fri"
- w - day of the week, numeric, 1 digit
- H - hour, numeric, 24 hour format
- h - hour, numeric, 12 hour format
- i - minutes, numeric
- s - seconds, numeric
- A - "AM" or "PM"
- a - "am" or "pm"
- S - English ordinal suffix, textual, 2 characters; i.e. "th", "nd"

Unrecognized characters in the format string will be printed as-is.

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)));
```

See also **gmdate** and **mktime**.

getdate

Name

`getdate` — get date/time information

Description

```
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"

gmdate

Name

`gmdate` — format a GMT/CUT date/time

Description

```
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** and **mktime**.

mktime

Name

`mktime` — get UNIX timestamp for a date

Description

```
int mktime(int hour, int minute, int second, int month, int day, int year);
```

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.

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) );
```

See also **date** and **time**.

time

Name

`time` — return current UNIX timestamp

Description

```
int time(void);
```

Returns the current time measured in the number of seconds since the Unix Epoch (January 1, 1970).

See also **date**.

microtime

Name

`microtime` — return current UNIX timestamp with microseconds

Description

```
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**.

set_time_limit

Name

`set_time_limit` — limit execution time

Description

```
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 php3.ini. 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.

Reference 6. dBase Functions

These functions allow you to access records stored in dBase-format (dbf) databases.

There is no support for indexes or memo fields.

dbase_create

Name

`dbase_create` — creates a dBase database

Description

```
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.

dbase_open

Name

`dbase_open` — opens a dBase database

Description

```
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

Name

`dbase_close` — close a dBase database

Description

```
bool dbase_close(int dbase_identifier);
```

Closes the database associated with *`dbase_identifier`*.

dbase_pack

Name

`dbase_pack` — packs a dBase database

Description

```
bool dbase_pack(int dbase_identifier);
```

Packs the specified database (permanently deleting all records marked for deletion using **dbase_delete_record**.

dbase_add_record

Name

`dbase_add_record` — add a record to a dBase database

Description

```
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_delete_record

Name

`dbase_delete_record` — deletes a record from a dBase database

Description

```
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

Name

`dbase_get_record` — gets a record from a dBase database

Description

```
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_numfields

Name

`dbase_numfields` — find out how many fields are in a dBase database

Description

```
int dbase_numfields(int dbase_identifier);
```

Returns the number of fields (columns) in the specified database.

dbase_numrecords

Name

`dbase_numrecords` — find out how many records are in a dBase database

Description

```
int dbase_numrecords(int dbase_identifier);
```

Returns the number of records (rows) in the specified database.

Reference 7. 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).

dbmopen

Name

`dbmopen` — opens a dbm database

Description

```
int dbmopen(string filename, int 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" or "w" for read, new (implies write) and 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

Name

`dbmclose` — closes a dbm database

Description

```
bool dbmclose(int dbm_identifier);
```

Unlocks and closes the specified database.

dbmexists

Name

`dbmexists` — tells if a value exists for a key in a dbm database

Description

```
bool dbmexists(int dbm_identifier, string key);
```

Returns true if there is a value associated with the *`key`*.

dbmfetch

Name

`dbmfetch` — fetches a value for a key from a dbm database

Description

```
string dbmfetch(int dbm_identifier, string key);
```

Returns the value associated with *`key`*.

dbminsert

Name

`dbminsert` — inserts a value for a key in a dbm database

Description

```
bool dbminsert(int dbm_identifier, string key, string value);
```

Adds the value to the database with the specified key.

Returns false if the specified key already exists. (To replace the value, use **dbmreplace**.)

dbmreplace

Name

`dbmreplace` — replaces the value for a key in a dbm database

Description

```
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

Name

`dbmdelete` — deletes the value for a key from a dbm database

Description

```
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

Name

`dbmfirstkey` — retrieves the first key from a dbm database

Description

```
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

Name

`dbmnextkey` — retrieves the next key from a dbm database

Description

```
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

Name

`dblist` — describes the dbm-compatible library being used

Description

```
string dblist(void);
```

Reference 8. Directory Functions

chdir

Name

`chdir` — change directory

Description

```
int chdir(string directory);
```

Changes PHP's current directory to *`directory`*. Returns FALSE if unable to change directory, TRUE otherwise.

dir

Name

`dir` — directory class

Description

```
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

Name

`closedir` — close directory handle

Description

```
void closedir(int dir_handle);
```

Closes the directory stream indicated by *`dir_handle`*. The stream must have previously been opened by **opendir**.

opendir

Name

`opendir` — open directory handle

Description

```
int opendir(string path);
```

Returns a directory handle to be used in subsequent **closedir**, **readdir**, and **rewinddir** calls.

readdir

Name

`readdir` — read entry from directory handle

Description

```
string readdir(int dir_handle);
```

Returns the filename of the next file from the directory. The filenames are not returned in any particular order.

rewinddir

Name

`rewinddir` — rewind directory handle

Description

```
void rewinddir(int dir_handle);
```

Resets the directory stream indicated by *`dir_handle`* to the beginning of the directory.

Reference 9. Dynamic Loading Functions

dl

Name

`dl` — load a PHP extension at runtime

Description

```
int dl(string library);
```

Loads the PHP extension defined in *`library`*. See also the extension_dir configuration directive.

Reference 10. Program Execution Functions

escapeshellcmd

Name

`escapeshellcmd` — escape shell metacharacters

Description

```
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. A standard use would be:

```
system(EscapeShellCmd($cmd))
```

exec

Name

`exec` — Execute an external program

Description

```
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** and **EscapeShellCmd**.

system

Name

`system` — Execute an external program and display output

Description

```
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.

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 the **exec** and **popen** functions.

passthru

Name

`passthru` — Execute an external program and display raw output

Description

```
string 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** and **fpassthru**.

virtual

Name

`virtual` — Perform an Apache sub-request

Description

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

virtual is an Apache-specific function which is equivalent to <!--#include virtual...--> in mod_include. It performs an Apache sub-request. It is useful for including CGI scripts or .shtml files, or anything else that you would parse through Apache. Note that for a CGI script, the script must generate valid CGI headers. At the minimum that means it must generate a Content-type header. For PHP files, you should use **include** or **require**.

Reference 11. filePro Functions

These functions allow read-only access to data stored in filePro databases.

filePro is a registered trademark by Fiserv, Inc. You can find more information about filePro at http://www.fileproplus.com/.

filepro

Name

`filepro` — read and verify the map file

Description

```
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

Name

`filepro_fieldname` — gets the name of a field

Description

```
string filepro_fieldname(int field_number);
```

Returns the name of the field corresponding to *`field_number`*.

filepro_fieldtype

Name

`filepro_fieldtype` — gets the type of a field

Description

```
string filepro_fieldtype(int field_number);
```

Returns the edit type of the field corresponding to *`field_number`*.

filepro_fieldwidth

Name

`filepro_fieldwidth` — gets the width of a field

Description

```
int filepro_fieldwidth(int field_number);
```

Returns the width of the field corresponding to *`field_number`*.

filepro_retrieve

Name

`filepro_retrieve` — retrieves data from a filePro database

Description

```
string filepro_retrieve(int row_number, int field_number);
```

Returns the data from the specified location in the database.

filepro_fieldcount

Name

`filepro_fieldcount` — find out how many fields are in a filePro database

Description

```
int filepro_fieldcount(void);
```

Returns the number of fields (columns) in the opened filePro database.

See also **filepro**.

filepro_rowcount

Name

`filepro_rowcount` — find out how many rows are in a filePro database

Description

```
int filepro_rowcount(void);
```

Returns the number of rows in the opened filePro database.

See also **filepro**.

Reference 12. Filesystem Functions

chgrp

Name

`chgrp` — change file group

Description

```
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.

On Windows, does nothing and returns true.

See also **chown** and **chmod**.

chmod

Name

`chmod` — change file mode

Description

```
int chmod(string filename, int mode);
```

Attempts to change the mode of the file specified by filename to that given in mode.

Returns true on success and false otherwise.

See also **chown** and **chgrp**.

chown

Name

`chown` — change file owner

Description

```
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**.

clearstatcache

Name

`clearstatcache` — clear file stat cache

Description

```
void clearstatcache(void);
```

Invoking the stat() 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.

Affected functions include **stat**, **file_exists**, **filectime**, **fileatime**, **fileinode**, **filegroup**, **fileowner**, **filesize**, **filetype**, and **fileperms**.

fclose

Name

`fclose` — close an open file pointer

Description

```
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**.

feof

Name

`feof` — test for end-of-file on a file pointer

Description

```
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** or **popen**.

fgetc

Name

`fgetc` — get character from file pointer

Description

```
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**, or **popen**.

See also **fopen**, **popen**, and **fgets**.

fgets

Name

`fgets` — get line from file pointer

Description

```
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, 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 **fopen**, or **popen**.

See also **fopen**, **popen**, and **fgetc**.

fgetss

Name

`fgetss` — get line from file pointer and strip HTML tags

Description

```
string fgetss(int fp, int length);
```

Identical to **fgets**, except that fgetss attempts to strip any HTML and PHP tags from the text it reads.

See also **fgets**, **fopen**, and **popen**.

file

Name

`file` — read entire file into an array

Description

```
array file(string filename);
```

Identical to **readfile**, except that file() returns the file in an array.

See also **readfile**, **fopen**, and **popen**.

file_exists

Name

`file_exists` — Check whether a file exists.

Description

```
int file_exists(string filename);
```

Returns true if the file specified by *`filename`* exists; false otherwise.

See also **clearstatcache**.

fileatime

Name

`fileatime` — get last access time of file

Description

```
int fileatime(string filename);
```

Returns the time the file was last accessed, or false in case of an error.

filectime

Name

`filectime` — get inode modification time of file

Description

```
int filectime(string filename);
```

Returns the time the file was last changed, or false in case of an error.

filegroup

Name

`filegroup` — get file group

Description

```
int filegroup(string filename);
```

Returns the group ID of the owner of the file, or false in case of an error.

fileinode

Name

`fileinode` — get file inode

Description

```
int fileinode(string filename);
```

Returns the inode number of the file, or false in case of an error.

filemtime

Name

`filemtime` — get file modification time

Description

```
int filemtime(string filename);
```

Returns the time the file was last modified, or false in case of an error.

fileowner

Name

`fileowner` — get file owner

Description

```
int fileowner(string filename);
```

Returns the user ID of the owner of the file, or false in case of an error.

fileperms

Name

`fileperms` — get file permissions

Description

```
int fileperms(string filename);
```

Returns the permissions on the file, or false in case of an error.

filesize

Name

`filesize` — get file size

Description

```
int filesize(string filename);
```

Returns the size of the file, or false in case of an error.

filetype

Name

`filetype` — get file type

Description

```
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.

fileumask

Name

`fileumask` — get file umask

Description

```
int fileumask(int umask);
```

Sets the umask to umask & 0777.

Returns the old umask.

fopen

Name

`fopen` — open file or URL

Description

```
int fopen(string filename, string mode);
```

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.

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.

Example 1. fopen() example

```
$fp = fopen( "/home/rasmus/file.txt", "r" );
$fp = fopen( "http://www.php.net/", "r" );
```

See also **fclose**.

fpassthru

Name

`fpassthru` — output all remaining data on a file pointer

Description

```
int fpassthru(int fp);
```

Reads to EOF on the given file pointer and writes the 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 **fopen** or **popen**.

fputs

Name

`fputs` — write to a file pointer

Description

```
int fputs(int fp, string str);
```

Writes str to the file stream pointed to by fp.

Returns the number of characters written, or false if an error occurs.

The file pointer must be valid, and must point to a file successfully opened by **fopen** or **popen**.

See also **fgets** and **fgetss**.

fseek

Name

`fseek` — seek on a file pointer

Description

```
int fseek(int fp, int offset);
```

Sets the file position indicator for the file referenced by fp to offset bytes into the file stream. Equivalent to calling (in C) `fseek( fp, offset, 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.

See also **ftell** and **rewind**.

ftell

Name

`ftell` — tell file pointer read/write position

Description

```
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**.

link

Name

`link` — Create a hard link

Description

```
int link(string target, string link);
```

Link creates a hard link.

See also the **symlink** to create soft links, and **readlink** along with **linkinfo**.

linkinfo

Name

`linkinfo` — Get information about a link

Description

```
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**.

link

Name

`link` — Create a symbolic link

Description

```
int symlink(string target, string link);
```

Symlink creates a symbolic link.

See also the **link** to create hard links, and **readlink** along with **linkinfo**.

mkdir

Name

`mkdir` — make directory

Description

```
int mkdir(string pathname, int mode);
```

Attempts to create the directory specified by pathname.

Returns true on success and false on failure.

See also **rmdir**.

pclose

Name

`pclose` — close process file pointer

Description

```
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 true if successful and false otherwise.

See also **popen**.

popen

Name

`popen` — open process file pointer

Description

```
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

Name

`readfile` — output a file

Description

```
int readfile(string filename);
```

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.

See also **fpassthru**, **file**, and **fopen**.

readlink

Name

`readlink` — Create a hard link

Description

```
int 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**.

rename

Name

`rename` — rename a file

Description

```
int rename(string oldname, string newname);
```

Attempts to rename *`oldname`* to *`newname`*.

Returns true on success and false on failure.

rewind

Name

`rewind` — rewind the position of a file pointer

Description

```
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

Name

`rmdir` — remove directory

Description

```
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

Name

`stat` — give information about a file

Description

```
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. 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

tempnam

Name

`tempnam` — create unique file name

Description

```
string tempnam(string dir, string prefix);
```

Creates a unique temporary filename.

Returns the new temporary filename, or the null string on failure.

Example 1. tempnam() example

```
$tmpfname = tempnam( "/tmp", "FOO" );
```

touch

Name

`touch` — set modification time of file

Description

```
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.

unlink

Name

`unlink` — Delete a file

Description

```
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.

basename

Name

`basename` — return file name part of path

Description

```
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**

dirname

Name

`dirname` — return file name part of path

Description

```
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**

copy

Name

`copy` — copy file

Description

```
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**

is_dir

Name

`is_dir` — tells whether the filename is a directory

Description

```
bool is_dir(string filename);
```

Returns true if the filename exists and is a directory.

See also **is_file** and **is_link**.

is_executable

Name

`is_executable` — tells whether the filename is executable

Description

```
bool is_executable(string filename);
```

Returns true if the filename exists and is executable.

See also **is_file** and **is_link**.

is_file

Name

`is_file` — tells whether the filename is a regular file

Description

```
bool is_file(string filename);
```

Returns true if the filename exists and is a regular file.

See also **is_dir** and **is_link**.

is_link

Name

`is_link` — tells whether the filename is a symbolic link

Description

```
bool is_link(string filename);
```

Returns true if the filename exists and is a symbolic link.

See also **is_dir** and **is_file**.

is_readable

Name

`is_readable` — tells whether the filename is readable

Description

```
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.

See also **is_writeable**.

is_writeable

Name

`is_writeable` — tells whether the filename is writeable

Description

```
bool is_readable(string filename);
```

Returns true if the filename exists and 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.

See also **is_readable**.

umask

Name

`umask` — changes the current umask

Description

```
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.

Reference 13. Functions related to HTTP

These functions let you manipulate the output sent back to the remote browser right down to the HTTP protocol level.

getallheaders

Name

`getallheaders` — Fetch all HTTP request headers

Description

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

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

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.

header

Name

`header` — Send a raw HTTP header

Description

```
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 1.0 Specification for more information on raw http headers. Remember that the **Header** function must be called before any actual output is sent either by normal HTML tags or from PHP.

```
Header("Location: http://www.php.net");  /* Redirect browser to PHP web site
*/
```

setcookie

Name

`setcookie` — Send a cookie

Description

```
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. 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. 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. ie. to see the contents of our test cookie in a script, simply do:

```
echo $TestCookie;
```

For more information on cookies, see Netscape's cookie specification at http://www.netscape.com/newsref/std/cookie_spec.html.

Reference 14. 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 GIF images.

GetImageSize

Name

`GetImageSize` — get the size of a GIF, JPG or PNG image

Description

```
array getimagesize(string filename);
```

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]; ?>>
```

NOTE: This function does not require the GD image library.

ImageArc

Name

`ImageArc` — draw a partial ellipse

Description

```
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

Name

`ImageChar` — draw a character horizontally

Description

```
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 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**.

ImageCharUp

Name

`ImageCharUp` — draw a character vertically

Description

```
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

Name

`ImageColorAllocate` — allocate a color for an image

Description

```
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.

ImageColorTransparent

Name

`ImageColorTransparent` — define a color as transparent

Description

```
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 the color identifier returned by **imagecolorallocate**.

ImageCopyResized

Name

`ImageCopyResized` — copy and resize part of an image

Description

```
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

Name

`ImageCreate` — create a new image

Description

```
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

Name

`ImageCreateFromGif` — create a new image from file or URL

Description

```
int imagecreatefromgif(string filename);
```

ImageCreateFromGif returns an image identifier representing the image obtained from the given filename.

ImageDashedLine

Name

`ImageDashedLine` — draw a dashed line

Description

```
int imagedashedline(int im, int x1, int y1, int x2, int y2, int col);
```

ImageLine 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

Name

`ImageDestroy` — destroy an image

Description

```
int imagedestroy(int im);
```

ImageDestroy frees any memory associated with image im. im is the image identifier returned by the **imagecreate** function.

ImageFill

Name

`ImageFill` — flood fill

Description

```
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

Name

`ImageFilledPolygon` — draw a filled polygon

Description

```
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

Name

`ImageFilledRectangle` — draw a filled rectangle

Description

```
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

Name

`ImageFillToBorder` — flood fill to specific color

Description

```
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

Name

`ImageFontHeight` — get font height

Description

```
int imagefontheight(int font);
```

Returns the pixel width of a character in font.

See also **imagefontwidth** and **imageloadfont**.

ImageFontWidth

Name

`ImageFontWidth` — get font width

Description

```
int imagefontwidth(int font);
```

Returns the pixel width of a character in font.

See also **imagefontheight** and **imageloadfont**.

ImageGif

Name

`ImageGif` — output image to browser or file

Description

```
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 the header function, you can create a PHP script that outputs GIF images directly.

ImageInterlace

Name

`ImageInterlace` — enable or disable interlace

Description

```
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.

ImageLine

Name

`ImageLine` — draw a line

Description

```
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

Name

`ImageLoadFont` — load a new font

Description

```
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

Name

`ImagePolygon` — draw a polygon

Description

```
int imagepolygon(int im, int 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**.

ImageRectangle

Name

`ImageRectangle` — draw a rectangle

Description

```
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

Name

`ImageSetPixel` — set a single pixel

Description

```
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

Name

`ImageString` — draw a string horizontally

Description

```
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

Name

`ImageStringUp` — draw a string vertically

Description

```
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

Name

`ImageSX` — get image width

Description

```
int imagesx(int im);
```

ImageSX returns the width of the image identified by im.

See also **imagecreate** and **imagesy**.

ImageSY

Name

`ImageSY` — get image height

Description

```
int imagesy(int im);
```

ImageSY returns the height of the image identified by im.

See also **imagecreate** and **imagesx**.

ImageColorAt

Name

`ImageColorAt` — get the index of the color of a pixel

Description

```
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

Name

`ImageColorClosest` — get the index of the closest color to the specified color

Description

```
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

Name

`ImageColorExact` — get the index of the specified color

Description

```
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**.

ImageColorSet

Name

`ImageColorSet` — set the color for the specified palette index

Description

```
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

Name

`ImageColorsForIndex` — get the colors for an index

Description

```
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

Name

`ImageColorsTotal` — find out the number of colors in an image's palette

Description

```
int imagecolorstotal(int im);
```

This returns the number of colors in the specified image's palette.

See also **imagecolorat** and **imagecolorsforindex**.

Reference 15. IMAP Functions

imap_append

Name

`imap_append` — Append a string message to a specified mailbox

Description

```
int imap_append(int imap_stream, string mbox, string message);
```

Returns true on sucess, false on error.

imap_append() function appends a string message to the specified mailbox *`mbox`*.

imap_base64

Name

`imap_base64` — Decode BASE64 encoded text

Description

```
string imap_base64(string text);
```

imap_base64() function decodes a BASE64 encoded text. The decoded message is retunred as a string.

Note: This function uses built in c-client base64 decoding, not an internal routine, so one must load the imap.so library to use this function.

imap_body

Name

`imap_body` — Read the message body

Description

```
string imap_body(int imap_stream, int msg_number);
```

imap_body() returns the body of the message, numbered *`msg_number`* in the current mailbox.

imap_check

Name

`imap_check` — Check current mailbox

Description

```
array imap_check(int imap_stream);
```

Returns false if there are no new messages, else returns information about the current mailbox.

imap_check() function checks the current mailbox status on the server and returns the information in an associative array with following elements.

```
Date : date of the message
Driver : driver
Mailbox : name of the mailbox
Nmsgs : number of messages
Recent : number of recent messages
```

imap_close

Name

`imap_close` — Close an IMAP stream

Description

```
int imap_close(int imap_stream);
```

Returns true on success and false on error.

Close the imap stream.

imap_createmailbox

Name

`imap_createmailbox` — Create a new mailbox

Description

```
int imap_createmailbox(int imap_stream, string mbox);
```

Returns true on success and false on error.

imap_createmailbox() creates a new mailbox specified by mbox.

imap_delete

Name

`imap_delete` — Mark a messge for deletion from current mailbox

Description

```
int imap_delete(int imap_stream, int msg_number);
```

Returns true on success and false on error.

imap_delete() function marks message pointed by msg_number for deletion. Actual deletion of the messages is done by **imap_expunge** .

imap_deletemailbox

Name

`imap_deletemailbox` — Delete a mailbox

Description

```
int imap_deletemailbox(int imap_stream, string mbox);
```

Returns true on success and false on error.

imap_deletemailbox() deteles the specified mailbox.

imap_expunge

Name

`imap_expunge` — Delete all messages marked for deletion

Description

```
int imap_expunge(int imap_stream);
```

Returns true on success and false on error.

imap_expunge() deletes all the messages marked for deletion by **imap_delete**.

imap_fetchbody

Name

`imap_fetchbody` — Fetch a particular section of the body of the message

Description

```
string imap_fetchbody(int imap_stream, int msg_number, int part_number);
```

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.

imap_fetchstructure

Name

`imap_fetchstructure` — Read the structure of a particular message

Description

```
array imap_fetchstructure(int imap_stream, int msg_number);
```

This function causes a fetch of all the structured information for the given msg_number. The returned value is an object with following elements.

```
type, encoding, ifsubtype, subtype, ifdescription, description, ifid,
 id, lines, bytes, ifparameters
```

It also returns an array of objects called parameters[]. This object has following properties.

```
attribute, value
```

In case of multipart, it also returns an array of objects of all the properties, called parts[].

imap_header

Name

`imap_header` — Read the header of the message

Description

```
array imap_header(int imap_stream, int msg_number);
```

This function returns an associative array of various header elements of the specified message.

```
Date, From, From2, Subject, To, cc, ReplyTo, Recent, Unseen,
      Flagged, Deleted, Msgno, MailDate, Size
```

imap_headerinfo

Name

`imap_headerinfo` — Read the header of the messsage

Description

```
array imap_headerinfo(int imap_stream, int msg_number);
```

Same as **imap_header**

imap_headers

Name

`imap_headers` — Returns headers for all messages in a mailbox

Description

```
array imap_headers(int imap_stream);
```

Returns an array of string formatted with header info. One element per mail message.

imap_listmailbox

Name

`imap_listmailbox` — Read the list of mailboxes

Description

```
array imap_listmailbox(int imap_stream);
```

Returns an array containing the names of the mailboxes.

imap_listsubscribed

Name

`imap_listsubscribed` — List all the subscribed mailboxes

Description

```
array imap_listsubscribed(int imap_stream);
```

Returns an array of all the mailboxes that you have subscribed.

imap_mail_copy

Name

`imap_mail_copy` — Copy specified messages to a mailbox

Description

```
int imap_mail_copy(int imap_stream, string mbox, string msglist);
```

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.

imap_mail_move

Name

`imap_mail_move` — Move specified messages to a mailbox

Description

```
int imap_mail_move(int imap_stream, string mbox, string msglist);
```

Returns true on success and false on error.

Moves mail messages specified by *`msglist`* to specified mailbox. msglist is a range not just message numbers.

imap_num_msg

Name

`imap_num_msg` — Gives the number of messages in the current mailbox

Description

```
int imap_num_msg(void);
```

Return the number of messages in the current mailbox.

imap_num_recent

Name

`imap_num_recent` — Gives the number of recent messages in current mailbox

Description

```
int imap_num_recent(int imap_stream);
```

Returns the number of recent messages in the current mailbox.

imap_open

Name

`imap_open` — Open an IMAP stream to a mailbox

Description

```
int imap_open(string username, string password);
```

Returns an IMAP stream on success and false on error.

imap_ping

Name

`imap_ping` — Check if the IMAP stream is still active

Description

```
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.

imap_renamemailbox

Name

`imap_renamemailbox` — Rename an old mailbox to new mailbox

Description

```
int imap_renamemailbox(int imap_stream, string old_mbox, string new_mbox);
```

Returns true on success and false on error.

This function renames on old mailbox to new mailbox.

imap_reopen

Name

`imap_reopen` — Reopen IMAP stream to new mailbox

Description

```
int imap_reopen(string mailbox, string username, string password);
```

Returns true on success and false on error.

This function reopens the specified stream to new mailbox.

imap_subscribe

Name

`imap_subscribe` — Subscribe to a mailbox

Description

```
int imap_subscribe(int imap_stream, string mbox);
```

Returns true on success and false on error.

Subscribe to a new mailbox.

imap_undelete

Name

`imap_undelete` — Unmark the message which is marked deleted

Description

```
int imap_undelete(int imap_stream, int msg_number);
```

Returns true on success and false on error.

This function removes the deletion flag for a specified message, which is set by **imap_delete**.

imap_unsubscribe

Name

`imap_unsubscribe` — Unsubscribe from a mailbox

Description

```
int imap_unsubscribe(int imap_stream, string mbox);
```

Returns true on success and false on error.

Unsubscribe from a specified mailbox.

Reference 16. PHP options & information

error_log

Name

`error_log` — send an error message somewhere

Description

```
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

Name

`error_reporting` — set which PHP errors are reported

Description

```
int error_reporting(int level);
```

Sets PHP's error reporting level. This 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

getenv

Name

`getenv` — Get the value of an environment variable.

Description

```
string getenv(string varname);
```

Returns the value of the environment variable *`varname`*, or false on an error.

get_cfg_var

Name

`get_cfg_var` — Get the value of a PHP configuration option.

Description

```
string get_cfg_var(string varname);
```

Returns the current value of the PHP configuration variable specified by *`varname`*, or false if an error occurs.

get_current_user

Name

`get_current_user` — Get the name of the owner of the current PHP script.

Description

```
string get_current_user(void);
```

Returns the name of the owner of the current PHP script.

See also **getmyuid**, **getmypid**, **getmyinode**, and **getlastmod**.

getlastmod

Name

`getlastmod` — Get time of last page modification.

Description

```
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

Name

`getmyinode` — Get the inode of the current script.

Description

```
int getmyinode(void);
```

Returns the current script's inode, or false on error.

See also **getmyuid**, **get_current_user**, **getmypid**, and **getlastmod**.

getmypid

Name

`getmypid` — Get PHP's process ID.

Description

```
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

Name

`getmyuid` — Get PHP script owner's UID.

Description

```
int getmyuid(void);
```

Returns the user ID of the current script, or false on error.

See also **getmypid**, **get_current_user**, **getmyinode**, and **getlastmod**.

phpinfo

Name

`phpinfo` — Output lots of PHP information.

Description

```
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

Name

`phpversion` — Get the current PHP version.

Description

```
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**.

putenv

Name

`putenv` — Set the value of an environment variable.

Description

```
void putenv(string setting);
```

Adds *`setting`* to the environment.

Example 1. Setting an Environment Variable

```
putenv("UNIQID=$uniqid");
```

Reference 17. LDAP Functions

ldap_add

Name

`ldap_add` — Add entries to LDAP directory

Description

```
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
```

ldap_bind

Name

`ldap_bind` — Bind to LDAP directory

Description

```
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

Name

`ldap_close` — Close link to LDAP server

Description

```
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.

ldap_connect

Name

`ldap_connect` — Connect to an LDAP server

Description

```
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

Name

`ldap_count_entries` — Count the number of entries in a search

Description

```
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

Name

`ldap_delete` — Delete an entry in a directory

Description

```
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

Name

`ldap_dn2ufn` — Convert DN to User Friendly Naming format

Description

```
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_first_attribute

Name

`ldap_first_attribute` — Return first attribute

Description

```
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

Name

`ldap_first_entry` — Return first result id

Description

```
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 e ntries from the result.

see also **ldap_get_entries**.

ldap_free_entry

Name

`ldap_free_entry` — Free entry result memory

Description

```
int ldap_free_entry(int result_entry_identifier);
```

Returns true on success and false on error.

ldap_free_entry() deallocates memory used to store the entries in LDAP result. All memory allocated for entries is automatically freed when the script terminates.

ldap_free_result

Name

`ldap_free_result` — Free result memory

Description

```
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.

ldap_get_attributes

Name

`ldap_get_attributes` — Get attributes from a search result entry

Description

```
array ldap_get_attributes(int link_identifier, int result_entry_identifier);
```

Returns a comlete 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.

```
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
```

see also **ldap_first_attribute** and **ldap_next_attribute**

ldap_get_dn

Name

`ldap_get_dn` — Get the DN of a result entry

Description

```
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

Name

`ldap_get_entries` — Get all result entries

Description

```
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

Name

`ldap_get_values` — Get all value from a result entry

Description

```
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.

```
    return_value["count"] = number of values for attribute
    return_value[0] = first value of attribute
```

```
        return_value[i] = ith value of attribute
```

ldap_list

Name

`ldap_list` — Single-level search

Description

```
int ldap_list(int link_identifier, string base_dn, string filter);
```

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_modify

Name

`ldap_modify` — Modify an LDAP entry

Description

```
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

Name

`ldap_next_attribute` — Get the next attribute in result

Description

```
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

Name

`ldap_next_entry` — Get next result entry

Description

```
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

Name

`ldap_read` — Read an entry

Description

```
int ldap_read(int link_identifier, string base_dn, string filter);
```

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.

ldap_search

Name

`ldap_search` — Search LDAP tree

Description

```
int ldap_search(int link_identifier, string base_dn, string filter);
```

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.

ldap_unbind

Name

`ldap_unbind` — Unbind from LDAP directory

Description

```
int ldap_unbind(int link_identifier);
```

Returns true on success and false on error.

ldap_unbind() function unbinds from the LDAP directory.

Reference 18. Mail Functions

mail

Name

`mail` — send mail

Description

```
void 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 space 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.

Example 2. Sending mail with extra headers.

```
mail("ssb@guardian.no", "the subject", $message, "X-Mailer: PHP/" +
phpversion());
```

Reference 19. Mathematical Functions

Abs

Name

`Abs` — absolute value

Description

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

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

Acos

Name

`Acos` — arc cosine

Description

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

Returns the arc cosine of arg in radians.

See also **asin** and **atan**.

Asin

Name

`Asin` — arc sine

Description

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

Returns the arc sine of arg in radians.

See also **acos** and **atan**.

Atan

Name

`Atan` — arc tangent

Description

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

Returns the arc tangent of arg in radians.

See also **acos** and **atan**.

BinDec

Name

`BinDec` — binary to decimal

Description

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

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

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

See also the **decbin** function.

Ceil

Name

`Ceil` — round fractions up

Description

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

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

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

See also **floor** and **round**.

Cos

Name

`Cos` — cosine

Description

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

Returns the cosine of arg in radians.

See also **sin** and **tan**.

DecBin

Name

`DecBin` — decimal to binary

Description

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

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

See also the **bindec** function.

DecHex

Name

`DecHex` — decimal to hexadecimal

Description

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

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

See also the **hexdec** function.

DecOct

Name

`DecOct` — decimal to octal

Description

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

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

Exp

Name

`Exp` — e to the power of...

Description

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

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

See also **pow**.

Floor

Name

`Floor` — round fractions down

Description

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

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

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

See also **ceil** and **round**.

getrandmax

Name

`getrandmax` — show largest possible random value

Description

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

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

See also **rand** and **srand**.

HexDec

Name

`HexDec` — hexadecimal to decimal

Description

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

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

See also the **dechex** function.

Log

Name

`Log` — natural logarithm

Description

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

Returns the natural logarithm of arg.

Log10

Name

`Log10` — base-10 logarithm

Description

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

Returns the base-10 logarithm of arg.

max

Name

`max` — find highest value

Description

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

max returns the numerically highest of the parameter values.

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

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

min

Name

`min` — find lowest value

Description

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

min returns the numerically lowest of the parameter values.

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

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

OctDec

Name

`OctDec` — octal to decimal

Description

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

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

See also **decoct**.

pi

Name

`pi` — get value of pi

Description

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

Returns an approximation of pi.

pow

Name

`pow` — exponential expression

Description

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

Returns base raised to the power of exp.

See also **exp**.

rand

Name

`rand` — generate a random value

Description

```
int rand(void );
```

Returns a pseudo-random value between 0 and RAND_MAX.

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

See also **srand** and **getrandmax**.

round

Name

`round` — Rounds a float.

Description

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

Returns the rounded value of *`val`*.

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

See also **ceil** and **floor**.

Sin

Name

`Sin` — sine

Description

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

Returns the sine of arg in radians.

See also **cos** and **tan**.

Sqrt

Name

`Sqrt` — square root

Description

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

Returns the square root of arg.

srand

Name

`srand` — seed the random number generator

Description

```
void srand(int seed);
```

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

```
srand( mktime() );  // seed with # of seconds since the Unix epoch
$randval = rand();
```

See also **rand** and **getrandmax**.

Tan

Name

`Tan` — tangent

Description

```
float tan(float arg);
```

Returns the tangent of arg in radians.

See also **sin** and **cos**.

Reference 20. Miscellaneous Functions

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

sleep

Name

`sleep` — Delay execution

Description

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

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

See also **usleep**.

usleep

Name

`usleep` — Delay execution in microseconds

Description

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

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

See also **sleep**.

uniqid

Name

`uniqid` — generate a unique id

Description

```
int uniqid(string prefix);
```

Uniqid returns a prefixed unique identifier based on 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. The prefix can be up to 114 characters long.

leak

Name

`leak` — Leak memory

Description

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

Leak leaks the specified amount of memory.

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

Reference 21. mSQL Functions

msql

Name

`msql` — send mSQL query

Description

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

Returns a positive mSQL result 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_close

Name

`msql_close` — close mSQL connection

Description

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

Returns true on success, false on error.

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

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

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

See also: **msql_connect** and **msql_pconnect**.

msql_connect

Name

`msql_connect` — open mSQL connection

Description

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

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

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

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

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

See also **msql_pconnect**, **msql_close**.

msql_create_db

Name

`msql_create_db` — create mSQL database

Description

```
int msql_create_db(void);
```

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

See also: **msql_drop_db**.

msql_createdb

Name

`msql_createdb` — create mSQL database

Description

```
int msql_createdb(void);
```

Identical to **msql_create_db**.

msql_data_seek

Name

`msql_data_seek` — move internal row pointer

Description

```
int msql_data_seek(int result_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 result identifier to pointer to the specifyed row number. The next call to **msql_fetch_row** would return that row.

See also: **msql_fetch_row**.

msql_dbname

Name

`msql_dbname` — get current mSQL database name

Description

```
string msql_dbname(string result, int i);
```

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

msql_drop_db

Name

`msql_drop_db` — drop (delete) mSQL database

Description

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

Returns true on success, false on failure.

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

See also: **msql_create_db**.

msql_dropdb

Name

`msql_dropdb` — drop (delete) mSQL database

Description

See **msql_drop_db**.

msql_error

Name

`msql_error` — returns error message of last msql call

Description

```
string msql_error( );
```

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

msql_fetch_array

Name

`msql_fetch_array` — fetch row as array

Description

```
int msql_fetch_array(int result);
```

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.

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

For further details, also see **msql_fetch_row**

msql_fetch_field

Name

`msql_fetch_field` — get field information

Description

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

Returns an object containing field information

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

The properties of the object are:

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

See also **msql_field_seek**.

msql_fetch_object

Name

`msql_fetch_object` — fetch row as object

Description

```
int msql_fetch_object(int result);
```

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).

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

See also: **msql_fetch_array** and **msql_fetch_row**.

msql_fetch_row

Name

`msql_fetch_row` — get row as enumerated array

Description

```
array msql_fetch_row(int result);
```

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

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

msql_fieldname

Name

`msql_fieldname` — get field name

Description

```
string msql_fieldname(int result, int field);
```

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

msql_field_seek

Name

`msql_field_seek` — set field offset

Description

```
int msql_field_seek(int result, int field_offset);
```

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

See also: **msql_fetch_field**.

msql_fieldtable

Name

`msql_fieldtable` — get table name for field

Description

```
int msql_fieldtable(int result, int field);
```

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

msql_fieldtype

Name

`msql_fieldtype` — get field type

Description

```
string msql_fieldtype(string result, int i);
```

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

msql_fieldflags

Name

`msql_fieldflags` — get field flags

Description

```
string msql_fieldflags(string result, int i);
```

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

msql_fieldlen

Name

`msql_fieldlen` — get field length

Description

```
int msql_fieldlen(string result, int i);
```

msql_fieldlen() returns the length of the specified field.

msql_free_result

Name

`msql_free_result` — free result memory

Description

```
int msql_free_result(int result);
```

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

msql_freeresult

Name

`msql_freeresult` — free result memory

Description

See **msql_free_result**

msql_list_fields

Name

`msql_list_fields` — list result fields

Description

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

msql_list_fields() retrieves information about the given tablename. Arguments are the database name and the table name. A result pointer is returned which can be used with **msql_fieldflags**, **msql_fieldlen**, **msql_fieldname**, and **msql_fieldtype**. A 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 `@msql_list_fields()` then this error string will also be printed out.

See also **msql_error**.

msql_listfields

Name

`msql_listfields` — list result fields

Description

See **msql_list_fields**.

msql_list_dbs

Name

`msql_list_dbs` — list mSQL databases on server

Description

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

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

msql_listdbs

Name

`msql_listdbs` — list mSQL databases on server

Description

See **msql_list_dbs**.

msql_list_tables

Name

`msql_list_tables` — list tables in an mSQL database

Description

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

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

msql_listtables

Name

`msql_listtables` — list tables in an mSQL database

Description

See **msql_list_tables**.

msql_num_fields

Name

`msql_num_fields` — get number of fields in result

Description

```
int msql_num_fields(int result);
```

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

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

msql_num_rows

Name

`msql_num_rows` — get number of rows in result

Description

```
int msql_num_rows(string result);
```

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

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

msql_numfields

Name

`msql_numfields` — get number of fields in result

Description

```
int msql_numfields(void);
```

Identical to **msql_num_fields**.

msql_numrows

Name

`msql_numrows` — get number of rows in result

Description

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

Identical to **msql_num_rows**.

msql_pconnect

Name

`msql_pconnect` — open persistent mSQL connection

Description

```
int msql_pconnect(string hostname);
```

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

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

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

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

This type of links is therefore called 'persistent'.

msql_query

Name

`msql_query` — send mSQL query

Description

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

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

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

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

msql_regcase

Name

`msql_regcase` — make regular expression for case insensitive match

Description

See **sql_regcase**.

msql_result

Name

`msql_result` — get result data

Description

```
int msql_result(int result, int i, mixed field);
```

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

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

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

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

msql_select_db

Name

`msql_select_db` — select mSQL database

Description

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

Returns true on success, false on error.

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

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

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

msql_selectdb

Name

`msql_selectdb` — select mSQL database

Description

See **msql_select_db**.

msql_tablename

Name

`msql_tablename` — get table name of field

Description

```
string msql_tablename(int result, 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++;
}
?>
```

Reference 22. MySQL Functions

mysql_affected_rows

Name

`mysql_affected_rows` — get number of affected rows in last query

Description

```
int mysql_affected_rows(int link_identifier);
```

Returns: The number of affected rows by the last query.

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.

mysql_close

Name

`mysql_close` — close MySQL connection

Description

```
int mysql_close(int link_identifier);
```

Returns: true on success, false on error

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

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

mysql_close() will not close persistent links generated by mysql_pconnect().

See also: **mysql_connect**, **mysql_pconnect**.

mysql_connect

Name

`mysql_connect` — open MySQL server connection

Description

```
int mysql_connect(string hostname, string username, string password);
```

Returns: A positive MySQL link identifier on success, or false on error.

mysql_connect() establishes a connection to a MySQL server. All of the arguments are optional, and if they're missing, defaults are assumed ('localhost', user name of the user that owns the server process, empty password).

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

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

See also **mysql_pconnect**, **mysql_close**.

mysql_create_db

Name

`mysql_create_db` — create MySQL database

Description

```
int mysql_create_db(string database name);
```

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

See also: **mysql_drop_db**. For downwards compatibility **mysql_createdb** can also be used.

mysql_data_seek

Name

`mysql_data_seek` — move internal row pointer

Description

```
int mysql_data_seek(int result_identifier, int row_number);
```

Returns: true on success, false on failure

mysql_data_seek() moves the internal row pointer of the MySQL result associated with the specified result identifier to pointer to the specifyed row number. The next call to **mysql_fetch_row** would return that row.

See also: **mysql_data_seek**.

mysql_dbname

Name

`mysql_dbname` — get current MySQL database name

Description

```
string mysql_dbname(string result, int i);
```

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

mysql_db_query

Name

`mysql_db_query` — send MySQL query

Description

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

Returns: A positive MySQL result identifier to the query result, or false on error.

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

See also **mysql_connect**. For downwards compatibility **mysql** can also be used.

mysql_drop_db

Name

`mysql_drop_db` — drop (delete) MySQL database

Description

```
int mysql_drop_db(string database_name, int link_identifier);
```

Returns: true on success, false on failure.

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

See also: **mysql_create_db** For downward compatibility **mysql_dropdb** can also be used.

mysql_errno

Name

`mysql_errno` — returns error number of last mysql call

Description

```
int mysql_errno();
```

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

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

See also: **mysql_error**

mysql_error

Name

`mysql_error` — returns error message of last mysql call

Description

```
string mysql_error();
```

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

```
<?php
mysql_connect("marliesle");
echo mysql_errno().": ".mysql_error()."<BR>";
mysql_select_db("nonexistentdb");
echo mysql_errno().": ".mysql_error()."<BR>";
$conn = mysql_query("SELECT * FROM nonexistenttable");
echo mysql_errno().": ".mysql_error()."<BR>";
?>
```

See also: **mysql_errno**

mysql_fetch_array

Name

`mysql_fetch_array` — fetch row as array

Description

```
int mysql_fetch_array(int result);
```

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.

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.

For further details, also see **mysql_fetch_row**

mysql_fetch_field

Name

`mysql_fetch_field` — get field information

Description

```
object mysql_fetch_field(int result, int field_offset);
```

Returns an object containing field information.

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

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 it the column is a BLOB
- type - the type of the column
- unsigned - 1 if the column is unsigned
- zerofill - 1 if the column is zero-filled

See also **mysql_field_seek**

mysql_fetch_lengths

Name

`mysql_fetch_lengths` — get max data size of output columns

Description

```
int 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** in an array, starting at offset 0.

See also: **mysql_fetch_row**.

mysql_fetch_object

Name

`mysql_fetch_object` — fetch row as object

Description

```
int mysql_fetch_object(int result);
```

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).

Speed-wise, the function is identical to **mysql_fetch_array**, and almost as quick as **mysql_fetch_row** (the difference is insignificant).

See also: **mysql_fetch_array** and **mysql_fetch_row**.

mysql_fetch_row

Name

`mysql_fetch_row` — get row as enumerated array

Description

```
array mysql_fetch_row(int result);
```

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

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

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

See also: **mysql_fetch_array**, **mysql_fetch_object**, **mysql_data_seek**, **mysql_fetch_lengths**, and **mysql_result**.

mysql_field_name

Name

`mysql_field_name` — get field name

Description

```
string mysql_field_name(string result, int i);
```

mysql_field_name() returns the name of the specified field. Arguments to the function is the result identifier and the field index, ie. `mysql_field_name($result,2);`

Will return the name of the second field in the result associated with the result identifier.

For downwards compatibility **mysql_fieldname** can also be used.

mysql_field_seek

Name

`mysql_field_seek` — set field offset

Description

```
int mysql_field_seek(int result, int field_offset);
```

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

See also: **mysql_fetch_field**.

mysql_field_table

Name

`mysql_field_table` — get table name for field

Description

```
int mysql_field_table(void);
```

Get the table name for field. For downward compatibility **mysql_fildtable** can also be used.

mysql_field_type

Name

`mysql_field_type` — get field type

Description

```
string mysql_field_type(string result, int i);
```

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", or "blob".

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."' table has ".$fields." fields and ".$rows." records
<BR>";
echo "The table has the following fields <BR>";
while ($i < $fields) {
    $type  = mysql_field_type  ($result, $i);
    $name  = mysql_field_name  ($result, $i);
    $len   = mysql_field_len   ($result, $i);
    $flags = mysql_field_flags ($result, $i);
    echo $type." ".$name." ".$len." ".$flags."<BR>";
    $i++;
}
mysql_close();
?>
```

For downward compatibility **mysql_fieldtype** can also be used.

mysql_field_flags

Name

`mysql_field_flags` — get field flags

Description

```
string mysql_field_flags(string result, int i);
```

mysql_field_flags() 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).

For downward compatibility **mysql_fieldflags** can also be used.

mysql_field_len

Name

`mysql_field_len` — get field length

Description

```
int mysql_field_len(string result, int i);
```

mysql_field_len() returns the length of the specified field. For downward compatibility **mysql_fieldlen** can also be used.

mysql_free_result

Name

`mysql_free_result` — free result memory

Description

```
int mysql_free_result(int result);
```

mysql_free_result() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script, you may call mysql_free_result() with the result identifier as an argument and the associated result memory will be freed.

For downward compatibility **mysql_freeresult** can also be used.

mysql_insert_id

Name

`mysql_insert_id` — get generated id from last INSERT

Description

```
int mysql_insert_id(void);
```

mysql_insert_id() returns the ID generated for an AUTO_INCREMENTED field. This function takes no arguments. It will return the auto-generated ID returned by the last INSERT query performed.

mysql_list_fields

Name

`mysql_list_fields` — list result fields

Description

```
int mysql_list_fields(string database, string tablename);
```

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_fieldflags**, **mysql_fieldlen**, **mysql_field_name**, and **mysql_field_type**.

A result identifier is a positive integer. The function returns -1 if a error occurs. A string describing the error will be placed in `$phperrmsg`, and unless the function was called as `@mysql()` then this error string will also be printed out.

For downward compatibility **mysql_listfields** can also be used.

mysql_list_dbs

Name

`mysql_list_dbs` — list MySQL databases on server

Description

```
int mysql_listdbs(void);
```

mysql_listdbs() will return a result pointer containing the databases available from the current mysql daemon. Use the **mysql_dbname** function to traverse this result pointer.

For downward compatibility **mysql_listdbs** can also be used.

mysql_list_tables

Name

`mysql_list_tables` — list tables in a MySQL database

Description

```
int mysql_list_tables(string database);
```

mysql_list_tables() takes a database name and result pointer much like the **mysql_db_query** function. The **mysql_tablename** function should be used to extract the actual table names from the result pointer.

For downward compatibility **mysql_listtables** can also be used.

mysql_num_fields

Name

`mysql_num_fields` — get number of fields in result

Description

```
int mysql_num_fields(int result);
```

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

See also: **mysql_db_query**, **mysql_query**, **mysql_fetch_field**, **mysql_num_rows**.

For downward compatibility **mysql_numfields** can also be used.

mysql_num_rows

Name

`mysql_num_rows` — get number of rows in result

Description

```
int mysql_num_rows(string result);
```

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

See also: **mysql_db_query**, **mysql_query** and, **mysql_fetch_row**.

For downward compatibility **mysql_numrows** can also be used.

mysql_pconnect

Name

`mysql_pconnect` — open persistent MySQL connection

Description

```
int mysql_pconnect(string hostname, string username, string password);
```

Returns: A positive MySQL persistent link identifier on success, or false on error

mysql_pconnect() acts very much like **mysql_connect** with two major differences.

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

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

This type of links is therefore called 'persistent'.

mysql_query

Name

`mysql_query` — send MySQL query

Description

```
int mysql_query(string query, int link_identifier);
```

Returns: A positive MySQL result identifier on success, or false on error.

mysql_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 **mysql_connect** was called, and use it.

See also: **mysql_db_query**, **mysql_select_db**, and **mysql_connect**.

mysql_result

Name

`mysql_result` — get result data

Description

```
int mysql_result(int result, int i, mixed field);
```

Returns: The contents of the cell at the row and offset in the specified MySQL result set.

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.

Recommended high-performance alternatives: **mysql_fetch_row**, **mysql_fetch_array**, and **mysql_fetch_object**.

mysql_select_db

Name

`mysql_select_db` — select MySQL database

Description

```
int mysql_select_db(string database_name, int link_identifier);
```

Returns: true on success, false on error

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

Every subsequent call to **mysql_query** will be made on the active database.

See also: **mysql_connect**, **mysql_pconnect**, and **mysql_query**

For downward compatibility **mysql_selectdb** can also be used.

mysql_tablename

Name

`mysql_tablename` — get table name of field

Description

```
string mysql_tablename(int result, int i);
```

mysql_tablename() takes a result pointer returned by the **mysql_list_tables** function as well as an integer index and returns the name of a table. The **mysql_num_rows** function may be used to determine the number of tables in the result pointer.

Example 1. mysql_tablename() example

```
<?php
mysql_connect ("localhost:3306");
$result = mysql_listtables ("wisconsin");
$i = 0;
while ($i < mysql_num_rows ($result)) {
    $tb_names[$i] = mysql_tablename ($result, $i);
    echo $tb_names[$i] . "<BR>";
    $i++;
}
?>
```

Reference 23. Sybase Functions

sybase_close

Name

`sybase_close` — close Sybase connection

Description

```
int sybase_close(int link_identifier);
```

Returns: true on success, false on error

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

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

sybase_close() will not close persistent links generated by sybase_pconnect().

See also: **sybase_connect**, **sybase_pconnect**.

sybase_connect

Name

`sybase_connect` — open Sybase server connection

Description

```
int sybase_connect(string servername, string username, string password);
```

Returns: A positive Sybase link identifier on success, or false on error.

sybase_connect() establishes a connection to a Sybase server. 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).

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

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

See also **sybase_pconnect**, **sybase_close**.

sybase_data_seek

Name

`sybase_data_seek` — move internal row pointer

Description

```
int sybase_data_seek(int result_identifier, int row_number);
```

Returns: true on success, false on failure

sybase_data_seek() moves the internal row pointer of the Sybase result associated with the specified result identifier to pointer to the specifyed row number. The next call to **sybase_fetch_row** would return that row.

See also: **sybase_data_seek**.

sybase_fetch_array

Name

`sybase_fetch_array` — fetch row as array

Description

```
int sybase_fetch_array(int result);
```

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

sybase_fetch_array() is an extended version of **sybase_fetch_row**. In addition to storing the data in the numeric indices of the result array, it also stores the data in associative indices, using the field names as keys.

An important thing to note is that using sybase_fetch_array() is NOT significantly slower than using sybase_fetch_row(), while it provides a significant added value.

For further details, also see **sybase_fetch_row**

sybase_fetch_field

Name

`sybase_fetch_field` — get field information

Description

```
object sybase_fetch_field(int result, int field_offset);
```

Returns an object containing field information.

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

The properties of the object are:

- name - column name
- 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 it 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 **sybase_field_seek**

sybase_fetch_object

Name

`sybase_fetch_object` — fetch row as object

Description

```
int sybase_fetch_object(int result);
```

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

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

Speed-wise, the function is identical to **sybase_fetch_array**, and almost as quick as **sybase_fetch_row** (the difference is insignificant).

See also: **sybase_fetch-array** and **sybase_fetch-row**.

sybase_fetch_row

Name

`sybase_fetch_row` — get row as enumerated array

Description

```
array sybase_fetch_row(int result);
```

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

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

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

See also: **sybase_fetch_array**, **sybase_fetch_object**, **sybase_data_seek**, **sybase_fetch_lengths**, and **sybase_result**.

sybase_field_seek

Name

`sybase_field_seek` — set field offset

Description

```
int sybase_field_seek(int result, int field_offset);
```

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

See also: **sybase_fetch_field**.

sybase_freeresult

Name

`sybase_freeresult` — free result memory

Description

```
int sybase_freeresult(int result);
```

sybase_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, you may call sybase_freeresult() with the result identifier as an argument and the associated result memory will be freed.

sybase_num_fields

Name

`sybase_num_fields` — get number of fields in result

Description

```
int sybase_num_fields(int result);
```

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

See also: **sybase_db_query**, **sybase_query**, **sybase_fetch_field**, **sybase_num_rows**.

sybase_num_rows

Name

`sybase_num_rows` — get number of rows in result

Description

```
int sybase_num_rows(string result);
```

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

See also: **sybase_db_query**, **sybase_query** and, **sybase_fetch_row**.

sybase_pconnect

Name

`sybase_pconnect` — open persistent Sybase connection

Description

```
int sybase_pconnect(string servername, string username, string password);
```

Returns: A positive Sybase persistent link identifier on success, or false on error

sybase_pconnect() acts very much like **sybase_connect** with two major differences.

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

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

This type of links is therefore called 'persistent'.

sybase_query

Name

`sybase_query` — send Sybase query

Description

```
int sybase_query(string query, int link_identifier);
```

Returns: A positive Sybase result identifier on success, or false on error.

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

See also: **sybase_db_query**, **sybase_select_db**, and **sybase_connect**.

sybase_result

Name

`sybase_result` — get result data

Description

```
int sybase_result(int result, int i, mixed field);
```

Returns: The contents of the cell at the row and offset in the specified Sybase result set.

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

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

Recommended high-performance alternatives: **sybase_fetch_row**, **sybase_fetch_array**, and **sybase_fetch_object**.

sybase_select_db

Name

`sybase_select_db` — select Sybase database

Description

```
int sybase_select_db(string database_name, int link_identifier);
```

Returns: true on success, false on error

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

Every subsequent call to **sybase_query** will be made on the active database.

See also: **sybase_connect**, **sybase_pconnect**, and **sybase_query**

Reference 24. Network Functions

fsockopen

Name

`fsockopen` — Open Internet or Unix domain socket connection.

Description

```
string fsockopen(string hostname, int port);
```

Opens an Internet domain socket connection to *`hostname`* on port *`port`* and returns a file pointer, which may be used by **fgets**, **fgetss**, **fputs**, and **fclose**.

If *`port`* is 0 and the operating system supports Unix domain sockets, *`hostname`* will be used as the filename of a Unix domain socket to connect to.

gethostbyaddr

Name

`gethostbyaddr` — Get the Internet host name corresponding to a given IP address.

Description

```
string gethostbyaddr(string ip_address);
```

Returns the host name of the Internet host specified by *`ip_address`*. If an error occurs, returns *`ip_address`*.

See also **gethostbyname**.

gethostbyname

Name

`gethostbyname` — Get the IP address corresponding to a given Internet host name.

Description

```
string gethostbyname(string hostname);
```

Returns the IP address of the Internet host specified by *`hostname`*.

See also **gethostbyaddr**.

openlog

Name

`openlog` — open connection to system logger

Description

```
void 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**.

syslog

Name

`syslog` — generate a system log message

Description

```
void 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.

closelog

Name

`closelog` — close connection to system logger

Description

```
string passthru(void);
```

closelog closes the descriptor being used to write to the system logger. The use of **closelog** is optional.

debugger_on

Name

`debugger_on` — enable internal PHP debugger

Description

```
void debugger_on(string address);
```

Enables the internal PHP debugger, connecting it to *`address`*. The debugger is still under development.

debugger_off

Name

`debugger_off` — disable internal PHP debugger

Description

```
void debugger_off(void);
```

Disables the internal PHP debugger. The debugger is still under development.

Reference 25. ODBC Functions

odbc_autocommit

Name

`odbc_autocommit` — Toggle autocommit behaviour

Description

```
int odbc_autocommit(int connection_id, int OnOff);
```

Returns `true` on success, `false` on failure.

By default, autocommit is on for a connection.

See also **odbc_commit** and **odbc_rollback**.

odbc_binmode

Name

`odbc_binmode` — handling of binary column data

Description

```
int odbc_binmode(int result_id, int mode);
```

(ODBC SQL types affected: BINARY, VARBINARY, LONGVARBINARY)

- 0: Passthru BINARY data
- 1: Return as is
- 2: Return and convert to char

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
0	0	passthru
1	0	passthru
2	0	passthru
0	0	passthru
0	>0	passthru
1	>0	return as is
2	>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 both longreadlen and binmode is 0. This means that columns of type binary and/or LONG get passedthru. Handling of binary Long columns is also affected by **odbc_longreadlen**

odbc_close

Name

`odbc_close` — Close an ODBC connection

Description

```
void odbc_close(int connection_id);
```

odbc_close will close down the connection to the database server associated with the given connection identifier.

> **NOTE:** This function will fail if there are open transactions on this connection. The connection will remain open in this case.

odbc_close_all

Name

`odbc_close_all` — Close all ODBC connections

Description

```
void odbc_close_all(void);
```

odbc_close_all will close down all connections to database server(s).

> **NOTE:** This function will fail if there are open transactions on a connection. This connection will remain open in this case.

odbc_commit

Name

`odbc_commit` — Commit an ODBC transaction

Description

```
int odbc_commit(int connection_id);
```

Returns: `true` on success, `false` on failure. All pending transactions on *`connection_id`* are committed.

odbc_connect

Name

`odbc_connect` — Connect to a datasource

Description

```
int odbc_connect(string dsn, string user, string password);
```

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. For persistent connections see **odbc_pconnect**.

odbc_cursor

Name

`odbc_cursor` — Get cursorname

Description

```
string odbc_cursor(int result_id);
```

odbc_cursor will return a cursorname for the given result_id.

odbc_do

Name

`odbc_do` — synonym for **odbc_exec**

Description

```
string odbc_do(int conn_id, string query);
```

odbc_do will execute a query on the given connection

odbc_exec

Name

`odbc_exec` — Prepare and execute a SQL statement

Description

```
int odbc_exec(int connection_id, string query_string);
```

Returns `false` on error. Returns an ODBC result identifier if the SQL command was executed successfully.

odbc_exec will send an SQL statement to the database server specified by *`connection_id`*. This parameter must be a valid identifier returned by **odbc_connect** or **odbc_pconnect**.

See also: **odbc_prepare** and **odbc_execute** for multiple execution of SQL statements.

odbc_execute

Name

`odbc_execute` — execute a prepated statement

Description

```
int odbc_execute(int result_id, array parameters_array);
```

Executes a statement prepared with **odbc_prepare**. Returns `true` on successful execution, `false` otherwise. The array *`arameters_array`* only needs to be given if you really have parameters in your statement.

odbc_fetch_into

Name

`odbc_fetch_into` — Fetch one result row into array

Description

```
int odbc_fetch_into(int result_id, int rownumber, array result_array);
```

Returns the number of columns in the result; `false` on error. *`result_array`* must be passed by reference, but it can be of any type since it will be converted to type array. The array will contain the column values starting at array index 0.

odbc_fetch_row

Name

`odbc_fetch_row` — Fetch a row

Description

```
int odbc_fetch_row(int result_id, int row_number);
```

If **odbc_fetch_row** was succesful (there was a row), `true` is returned. If there are no more rows, `false` is returned.

odbc_fetch_row fetches a row of the data that was returned by **odbc_do** / **odbc_exec**. After **odbc_fetch_row** is called, the fields of that row can be accessed with **odbc_result**.

If *`row_number`* is not specified, **odbc_fetch_row** will try to fetch the next row in the result set. Calls to **odbc_fetch_row** with and without *`row_number`* can be mixed.

To step through the result more than once, you can call **odbc_fetch_row** with *`row_number`* 1, and then continue doing **odbc_fetch_row** without *`row_number`* to review the result. If a driver doesn't support fetching rows by number, the *`row_number`* parameter is ignored.

odbc_field_name

Name

`odbc_field_name` — Get the columnname

Description

```
string odbc_fieldname(int result_id, int field_number);
```

odbc_field_name will return the name of the field occupying the given column number in the given ODBC result identifier. Field numbering starts at 1. `false` is returned on error.

odbc_field_num

Name

`odbc_field_num` — return column number

Description

```
int odbc_fieldnum(int result_id, string field_name);
```

odbc_field_num will return the number of the column slot that corresponds to the named field in the given ODBC result identifier. Field numbering starts at 1. `false` is returned on error.

odbc_field_type

Name

`odbc_field_type` — datatype of a field

Description

```
string odbc_field_type(int result_id, mixed field);
```

odbc_field_type will return the SQL type of the field referecend by name or number in the given ODBC result identifier. Field numbering runs from 1.

odbc_free_result

Name

`odbc_free_result` — free resources associated with a result

Description

```
int odbc_free_result(int result_id);
```

Always returns `true`.

odbc_free_result only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call **odbc_free_result**, and the memory associated with *`result_id`* will be freed.

NOTE: If auto-commit is disabled (see **odbc_autocommit**) and you call **odbc_free_result** before commiting, all pending transactions are rolled back.

odbc_longreadlen

Name

`odbc_longreadlen` — handling of LONG columns

Description

```
int odbc_longreadlen(int result_id, int length);
```

(ODBC SQL types affected: LONG, LONGVARBINARY) The number of bytes returned to PHP is controled by the parameter length. If it is set to 0, Long column data is passed thru to the client.

NOTE: Handling of LONGVARBINARY columns is also affected by **odbc_binmode**

odbc_num_fields

Name

`odbc_num_fields` — number of columns in a result

Description

```
int odbc_num_fields(int result_id);
```

odbc_num_fields will return the number of fields (columns) in an ODBC result. This function will return -1 on error. The argument is a valid result identifier returned by **odbc_exec**.

odbc_pconnect

Name

`odbc_pconnect` — Open a persistent database connection

Description

```
int odbc_pconnect(string dsn, string user, string password);
```

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 more information on persistent connections, refer to the PHP3 FAQ.

odbc_prepare

Name

`odbc_prepare` — Prepares a statement for execution

Description

```
int odbc_prepare(int connection_id, string query_string);
```

Returns `false` on error.

Returns an ODBC result identifier if the SQL command was prepared successfully. The result identifier can be used later to execute the statement with **odbc_execute**.

odbc_num_rows

Name

`odbc_num_rows` — Number of rows in a result

Description

```
int odbc_num_rows(int result_id);
```

odbc_num_rows will return the number of rows in an ODBC result. This function will return -1 on error. For INSERT, UPDATE and DELETE statements **odbc_num_rows** returns the number of rows affected. For a SELECT clause this `can` be the number of rows available.

Note: Using **odbc_num_rows** to determine the number of rows available after a SELECT will return -1 with many drivers.

odbc_result

Name

`odbc_result` — get result data

Description

```
string odbc_result(int result_id, mixed field);
```

Returns the contents of the field.

Field indices start from 1. Regarding the way binary or long column data is returned refer to **odbc_binmode** and **odbc_longreadlen**.

odbc_result_all

Name

`odbc_result_all` — Print result as HTML table

Description

```
int odbc_result_all(int result_id, string format);
```

Returns the number of rows in the result or `false` on error.

odbc_result_all will print all rows from a result identifier produced by **odbc_exec**. The result is printed in HTML table format. With the optional string argument *`format`*, additional overall table formatting can be done.

odbc_rollback

Name

`odbc_rollback` — Rollback a transaction

Description

```
int odbc_rollback(int connection_id);
```

Rolls back all pending statements on *`connection_id`*. Returns `true` on success, `false` on failure.

Reference 26. Oracle functions

Ora_Close

Name

`Ora_Close` — close an Oracle cursor

Description

```
int ora_close(int cursor);
```

Returns true if the close succeeds, otherwise false. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

This function closes a data cursor opened with **ora_open**.

Ora_ColumnName

Name

`Ora_ColumnName` — get name of Oracle result column

Description

```
string Ora_ColumnName(int cursor, int column);
```

Returns the name of the field/column *`column`* on the cursor *`cursor`*. The returned name is in all uppercase letters.

Ora_ColumnType

Name

`Ora_ColumnType` — get type of Oracle result column

Description

```
string Ora_ColumnType(int cursor, int column);
```

Returns the Oracle data type name of the field/column *`column`* on the cursor *`cursor`*. The returned type will be one of the following:

```
"VARCHAR2"
"VARCHAR"
"CHAR"
"NUMBER"
"LONG"
"LONG RAW"
"ROWID"
"DATE"
"CURSOR"
```

Ora_Commit

Name

`Ora_Commit` — commit an Oracle transaction

Description

```
int ora_commit(int conn);
```

Returns true on success, false on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions. This function commits an Oracle transaction. A transaction is defined as all the changes on a given connection since the last commit/rollback, autocommit was turned off or when the connection was established.

Ora_CommitOff

Name

`Ora_CommitOff` — disable automatic commit

Description

```
int ora_commitoff(int conn);
```

Returns true on success, false on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

This function turns off automatic commit after each **ora_exec**.

Ora_CommitOn

Name

`Ora_CommitOn` — enable automatic commit

Description

```
int ora_commiton(int conn);
```

This function turns on automatic commit after each **ora_exec** on the given connection.

Returns true on success, false on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

Ora_Error

Name

`Ora_Error` — get Oracle error message

Description

```
string Ora_Error(int cursor);
```

Returns an error message of the form *`XXX`*-*`NNNNN`* where *`XXX`* is where the error comes from and *`NNNNN`* identifies the error message.

On UNIX versions of Oracle, you can find details about an error message like this: `$` **`oerr ora 00001`** `00001, 00000, "unique constraint (%s.%s) violated" // *Cause: An update or insert statement attempted to insert a duplicate key // For Trusted ORACLE configured in DBMS MAC mode, you may see // this message if a duplicate entry exists at a different level. // *Action: Either remove the unique restriction or do not insert the key`

Ora_ErrorCode

Name

`Ora_ErrorCode` — get Oracle error code

Description

```
int Ora_ErrorCode(int cursor);
```

Returns the numeric error code of the last executed statement on the specified cursor.

Ora_Exec

Name

`Ora_Exec` — execute parsed statement on an Oracle cursor

Description

```
int ora_exec(int cursor);
```

Returns true on success, false on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

Ora_Fetch

Name

`Ora_Fetch` — fetch a row of data from a cursor

Description

```
int ora_fetch(int cursor);
```

Returns true (a row was fetched) or false (no more rows, or an error occured). If an error occured, details can be retrieved using the **ora_error** and **ora_errorcode** functions. If there was no error, **ora_errorcode** will return 0. Retrieves a row of data from the specified cursor.

Ora_GetColumn

Name

`Ora_GetColumn` — get data from a fetched row

Description

```
mixed ora_getcolumn(int cursor, mixed column);
```

Returns the column data. If an error occurs, False is returned and **ora_errorcode** will return a non-zero value. Note, however, that a test for False on the results from this function may be true in cases where there is not error as well (NULL result, empty string, the number 0, the string "0"). Fetches the data for a column or function result.

Ora_Logoff

Name

`Ora_Logoff` — close an Oracle connection

Description

```
int ora_logoff(int connection);
```

Returns true on success, False on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions. Logs out the user and disconnects from the server.

Ora_Logon

Name

`Ora_Logon` — open an Oracle connection

Description

```
int ora_logon(string user, string password);
```

Establishes a connection between PHP and an Oracle database with the given username and password.

Returns a connection index on success, or False on failure. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

Ora_Open

Name

`Ora_Open` — open an Oracle cursor

Description

```
int ora_open(int connection);
```

Opens an Oracle cursor associated with connection.

Returns a cursor index or False on failure. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

Ora_Parse

Name

`Ora_Parse` — parse an SQL statement

Description

```
int ora_parse(int cursor_ind, string sql_statement, int defer);
```

This function parses an SQL statement or a PL/SQL block and associates it with the given cursor. Returns 0 on success or -1 on error.

Ora_Rollback

Name

`Ora_Rollback` — roll back transaction

Description

```
int ora_rollback(int connection);
```

This function undoes an Oracle transaction. (See **ora_commit** for the definition of a transaction.)

Returns true on success, false on error. Details about the error can be retrieved usign the **ora_error** and **ora_errorcode** functions.

Reference 27. 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 6.3.1 is available at www.postgreSQL.org.

Since version 6.3 (03/02/1998) PostgreSQL use unix domain sockets, a table is given to this new possibilities. This socket will be found in `/tmp/.s.PGSQL.5432`. This option can be enabled with the '-i' flag to **postmaster** and it's meaning is: "listen on TCP/IP sockets as well as Unix domain socket".

Table 1. Postmaster and PHP

Postmaster	PHP	Status
postmaster &	pg_connect("", "", "", "", "dbname");	OK
postmaster -i &	pg_connect("", "", "", "", "dbname");	OK
postmaster &	pg_connect("localhost", "", "", "", "dbname");	Unable to connect to PostgreSQL server: connectDB() failed: Is the postmaster running and accepting TCP/IP (with -i) connection at 'localhost' on port '5432'? in /path/to/file.php3 on line 20.
postmaster -i &	pg_connect("localhost", "", "", "", "dbname");	OK

pg_Close

Name

`pg_Close` — closes a PostgreSQL connection

Description

```
void pg_close(int connection);
```

Returns false if connection is not a valid connection index, true otherwise. Closes down the connection to a PostgreSQL database associated with the given connection index.

pg_cmdTuples

Name

`pg_cmdTuples` — returns number of affected tuples

Description

```
int pg_cmdtuples(int result_id);
```

pg_cmdTuples() returns the number of tuples (instances) affected by INSERT, UPDATE, and DELETE queries. If no tuple is affected the function will return 0.

Example 1. pg_cmdtuples

```
<?php
$result = pg_exec($conn, "INSERT INTO verlag VALUES ('Autor')");
$cmdtuples = pg_cmdtuples($result);
echo $cmdtuples . " <- cmdtuples affected.";
?>
```

pg_Connect

Name

`pg_Connect` — opens a connection

Description

```
int pg_connect(string host, string port, string options, string tty, string dbname);
```

Returns a connection index on success, or false if the connection could not be made. Opens a connection to a PostgreSQL database. Each of the arguments should be a quoted string, including the port number. The options and tty arguments are optional and can be left out. This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple connections open at once.

See also **pg_pConnect**.

pg_DBname

Name

`pg_DBname` — database name

Description

```
string pg_dbname(int connection);
```

Returns the name of the database that the given PostgreSQL connection index is connected to, or false if connection is not a valid connection index.

pg_ErrorMessage

Name

`pg_ErrorMessage` — error message

Description

```
string pg_errormessage(int connection);
```

Returns a string containing the error message, false on failure. Details about the error probably cannot be retrieved using the pg_errormessage() function if an error occured on the last database action for which a valid connection exists, this function will return a string containing the error message generated by the backend server.

pg_Exec

Name

`pg_Exec` — execute a query

Description

```
int pg_exec(int connection, string query);
```

Returns a result index if query could be executed, false on failure or if connection is not a valid connection index. Details about the error can be retrieved using the **pg_ErrorMessage** function if connection is valid. Sends an SQL statement to the PostgreSQL database specified by the connection index. The connection must be a valid index that was returned by **pg_Connect**. The return value of this function is an index to be used to access the results from other PostgreSQL functions.

NOTE: PHP2 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 PHP3.

pg_FieldIsNull

Name

`pg_FieldIsNull` — Test if a field is NULL

Description

```
int pg_fieldisnull(int result_id, string fieldname);
```

Test if a field is NULL or not.

pg_FieldName

Name

`pg_FieldName` — Returns the name of a field

Description

```
string pg_fieldname(int result_id, int field_number);
```

pg_FieldName() will return the name of the field occupying the given column number in the given PostgreSQL result identifier. Field numbering starts from 0.

pg_FieldNum

Name

`pg_FieldNum` — Returns the number of a columns

Description

```
string pg_fieldnum(int result_id, int field_name);
```

pg_FieldNum() will return the number of the column slot that corresponds to the named field in the given PosgreSQL result identifier. Field numbering starts at 0. This function will return -1 on error.

pg_FieldPrtLen

Name

`pg_FieldPrtLen` — Returns the printed length

Description

```
int pg_fieldprtlen(int result_id, int row_number, string field_name);
```

pg_FieldPrtLen() will return the actual printed length (number of characters) of a specific value in a PostgreSQL result. Row numbering starts at 0. This function will return -1 on an error.

pg_FieldSize

Name

`pg_FieldSize` — Returns the internal storage size of the named field

Description

```
int pg_fieldsize(int result_id, string field_name);
```

pg_FieldSize() will return the internal storage size (in bytes) of the named field in the given PostgreSQL result. A field size of 0 indicates a variable length field. This function will return -1 on error.

pg_FieldType

Name

`pg_FieldType` — Returns the type name for the corresponding field number

Description

```
int pg_fieldtype(int result_id, int field_number);
```

pg_FieldType() will return a string containing the type name of the given field in the given PostgreSQL result identifier. Field numbering starts at 0.

pg_FreeResult

Name

`pg_FreeResult` — Frees up memory

Description

```
int pg_freeresult(int result_id);
```

pg_FreeResult() only needs to be called if you are worried about using too much memory while your script is running. All result memory will automatically be freed when the script is finished. But, if you are sure you are not going to need the result data anymore in a script, you may call pg_FreeResult() with the result identifier as an argument and the associated result memory will be freed.

pg_GetLastOid

Name

`pg_GetLastOid` — Returns the last object identifier

Description

```
int pg_getlastoid(void);
```

pg_GetLastOid() can be used to retrieve the Oid assigned to an inserted tuple if the last command sent via **pg_Exec** was an SQL INSERT. This function will return a positive integer if there was a valid Oid. It will return -1 if an error occured or the last command sent via **pg_Exec** was not an INSERT.

pg_Host

Name

`pg_Host` — Returns the host name

Description

```
string pg_host(int connection_id);
```

pg_Host() will return the host name of the given PostgreSQL connection identifier is connected to.

pg_loclose

Name

`pg_loclose` — close a large object

Description

```
void pg_loclose(int fd);
```

pg_loclose closes an Inversion Large Object. *`fd`* is a file descriptor for the large object from **pg_loopen**.

pg_locreate

Name

`pg_locreate` — create a large object

Description

```
int pg_locreate(int conn);
```

pg_locreate creates an Inversion Large Object and returns the oid of the large object. *`conn`* specifies a valid database connection. PostgreSQL access modes INV_READ, INV_WRITE, and INV_ARCHIVE are not supported, the object is created always with both read and write access. INV_ARCHIVE has been removed from PostgreSQL itself (version 6.3 and above).

pg_loopen

Name

`pg_loopen` — open a large object

Description

```
int pg_loopen(int conn, int objoid, string mode);
```

pg_loopen open an Inversion Large Object and returns file descriptor of the large object. The file descriptor encapsulates information about the connection. Do not close the connection before closing the large object file descriptor. *`objoid`* specifies a valid large object oid and *`mode`* can be either "r", "w", or "rw".

pg_loread

Name

`pg_loread` — read a large object

Description

```
void pg_loread(int fd, string bufvar, int len);
```

pg_loread reads at most *`len`* bytes from a large object into a variable name *`bufvar`*. *`fd`* specifies a valid large object file descriptor. *`bufvar`* specifies a valid buffer variable to contain the large object segment and *`len`* specifies the maximum allowable size of the large object segment.

pg_loreadall

Name

`pg_loreadall` — read a entire large object

Description

```
void pg_loreadall(int fd);
```

pg_loreadall reads a large object and passes it straight through to the browser after sending all pending headers. Mainly intended for sending binary data like images or sound.

pg_lounlink

Name

`pg_lounlink` — delete a large object

Description

```
void pg_lounlink(int conn, int lobjid);
```

pg_lounlink deletes a large object with the *`lobjid`* identifier for that large object.

pg_lowrite

Name

`pg_lowrite` — write a large object

Description

```
void pg_lowrite(int fd, string buf, int len);
```

pg_lowrite writes at most *`len`* bytes to a large object from a variable *`buf`*. *`fd`* is a file descriptor for the large object from **pg_loopen**.

pg_NumFields

Name

`pg_NumFields` — Returns the number of fields

Description

```
int pg_numfields(int result_id);
```

pg_NumFields() will return the number of fields (columns) in a PostgreSQL result. The argument is a valid result identifier returned by **pg_Exec**. This function will return -1 on error.

pg_NumRows

Name

`pg_NumRows` — Returns the number of rows

Description

```
int pg_numfields(int result_id);
```

pg_NumFields() will return the number of rows in a PostgreSQL result. The argument is a valid result identifier returned by **pg_Exec**. This function will return -1 on error.

pg_Options

Name

`pg_Options` — Returns options

Description

```
string pg_options(int connection_id);
```

pg_Options() will return a string containing the options specified on the given PostgreSQL connection identifier.

pg_pConnect

Name

`pg_pConnect` — make a persistent database connection

Description

```
int pg_pconnect(string host, string port, string options, string tty, string dbname);
```

Returns a connection index on success, or false if the connection could not be made. Opens a persistent connection to a PostgreSQL database. Each of the arguments should be a quoted string, including the port number. The options and tty arguments are optional and can be left out. This function returns a connection index that is needed by other PostgreSQL functions. You can have multiple persistent connections open at once. See also **pg_Connect**.

pg_Port

Name

`pg_Port` — Returns the port number

Description

```
int pg_port(int connection_id);
```

pg_Port() will return the port number that the given PostgreSQL connection identifier is connected to.

pg_Result

Name

`pg_Result` — Returns values from a result identifier

Description

```
mixed pg_result(int result_id, int row_number, mixed fieldname);
```

pg_Result() will return values from a result identifier produced by **pg_Exec**. The *`row_number`* and *`fieldname`* sepcify what cell in the table of results to return. Row numbering starts from 0. Instead of naming the field, you may use the field index as an unquoted number. Field indices start from 0.

PostgreSQL has many built in types and only the basic ones are directly supported here. All forms of integer, boolean and oid types are returned as integer values. All forms of float, and real types are returned as double values. All other types, including arrays are returned as strings formatted in the same default PostgreSQL manner that you would see in the **psql** program.

pg_tty

Name

`pg_tty` — Returns the tty name

Description

```
string pg_tty(int connection_id);
```

pg_tty() will return the tty name that server side debugging output is sent to on the given PostgreSQL connection identifier.

Reference 28. Regular expression functions

ereg

Name

`ereg` — regular expression match

Description

```
int ereg(string pattern, string string, array regs);
```

Searchs *`string`* for matches to the regular expression given in *`pattern`*. If matches are found for parenthesized substrings of pattern and the function is called with the third argument *`regs`*, the matches will be stored in the elements of *`regs`*.

Searching is case sensitive.

Returns true if a match for pattern was found in string, or false if no matches were found or an error occurred.

The following code snippet takes a date in ISO format (YYYY-MM-DD) and prints it in DD.MM.YYYY format:

Example 1. ereg() example

```
if ( ereg( "([0-9]{4})-([0-9]{1,2})-([0-9]{1,2})", $date, $regs ) ) {
    echo "$regs[3].$regs[2].$regs[1]";
} else {
    echo "Invalid date format: $date";
}
```

See also **eregi**, **ereg_replace**, and **eregi_replace**.

ereg_replace

Name

`ereg_replace` — replace regular expression

Description

```
string ereg_replace(string pattern, string replacement, string string);
```

This function scans *`string`* for matches to *`pattern`*, then replaces the matched text with *`replacement`*.

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. For example, the following code snippet prints "This was a test" three times:

Example 1. ereg_replace() example

```
$string = "This is a test";
echo ereg_replace( " is", " was", $string );
echo ereg_replace( "( )is", "\\1was", $string );
echo ereg_replace( "(( )is)", "\\2was", $string );
```

See also **ereg**, **eregi**, and **eregi_replace**.

eregi

Name

`eregi` — case insensitive regular expression match

Description

```
int eregi(string pattern, string string, array regs);
```

This function is identical to **ereg** save that this ignores case distinction when matching alphabetic characters.

See also **ereg**, **ereg_replace**, and **eregi_replace**.

eregi_replace

Name

`eregi_replace` — replace regular expression case insensitive

Description

```
string eregi_replace(string pattern, string replacement, string string);
```

This function is identical to **ereg_replace** save that this ignores case distinction when matching alphabetic characters.

See also **ereg**, **eregi**, and **ereg_replace**.

split

Name

`split` — split string into array by regular expression

Description

```
array split(string pattern, string string, int limit);
```

Returns an array of strings, each of which is a substring of string formed by splitting it on boundaries formed by *`pattern`*. If an error occurs, returns false.

To get the first five fields from a line from `/etc/passwd`:

Example 1. split() example

```
$passwd_list = split( ":", $passwd_line, 5 );
```

sql_regcase

Name

`sql_regcase` — make regular expression for case insensitive match

Description

```
string sql_regcase(string string);
```

Returns a valid regular expression which will match *`string`*, ignoring case. This expression is *`string`* with each character converted to a bracket expression; this bracket expression contains that character's uppercase and lowercase form if applicable, otherwise it contains the original character twice.

Example 1. sql_regcase() example

```
echo sql_regcase( "Foo bar" );
```

prints

```
[Ff][Oo][Oo][  ][Bb][Aa][Rr]
```

.

This can be used to achieve case insensitive pattern matching in products which support only case sensitive regular expressions.

Reference 29. Solid Functions

The Solid functions are deprecated, you probably want to use the Unified ODBC functions instead.

solid_close

Name

`solid_close` — close a Solid connection

Description

See **odbc_close**.

solid_connect

Name

`solid_connect` — connect to a Solid data source

Description

See **odbc_connect**.

solid_exec

Name

`solid_exec` — execute a Solid query

Description

See **odbc_exec**.

solid_fetchrow

Name

`solid_fetchrow` — fetch row of data from Solid query

Descriptio

See **odbc_fetch_row**

solid_fieldname

Name

`solid_fieldname` — get name of column from Solid query

Description

See **odbc_field_name**.

solid_fieldnum

Name

`solid_fieldnum` — get index of column from Solid query

Description

See **odbc_field_num**.

solid_freeresult

Name

`solid_freeresult` — free result memory from Solid query

Description

See **odbc_free_result**.

solid_numfields

Name

`solid_numfields` — get number of fields in Solid result

Description

See **odbc_num_fields**.

solid_numrows

Name

`solid_numrows` — get number of rows in Solid result

Description

See **odbc_num_rows**.

solid_result

Name

`solid_result` — get data from Solid results

Description

See **odbc_result**.

Reference 30. SNMP Functions

These SNMP functions are only available through the optional SNMP dynamically loadable PHP extension. You can find this extension in the dl/ directory in the PHP3 distribution.

snmpget

Name

`snmpget` — Fetch an SNMP object

Description

```
int snmpget(string hostname, string community, string object_id);
```

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.

```
  snmpget("127.0.0.1", "public", "system.SysContact.0")
```

snmpwalk

Name

`snmpwalk` — Fetch all the SNMP objects from an agent

Description

```
int snmpwalk(int hostname, string community, string object_id);
```

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];
}
```

Reference 31. String functions

AddSlashes

Name

`AddSlashes` — quote string with slashes

Description

```
string addslashes(string str);
```

Returns a string with backslashes before characters that need to be quoted in database queries etc. These characters are single quote (`'`), double quote (`"`) and backslash (`\`).

See also **stripslashes** and **quotemeta**.

Chop

Name

`Chop` — remove trailing whitespace

Description

```
string chop(string str);
```

Returns the argument string without trailing whitespace.

Example 1. chop() example

```
$trimmed = Chop($line);
```

Chr

Name

`Chr` — return a specific character

Description

```
string chr(int ascii);
```

Returns a one-character string containing the character specified by ascii.

Example 1. chr() example

```
$str += chr(27); /* add an escape character at the end of $str */
```

This function complements **ord**.

crypt

Name

`crypt` — DES-encrypt a string

Description

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 you do not have a crypt function on your Unix system, you can use Michael Glad's public domain UFC-Crypt package which was developed in Denmark and hence not restricted by US export laws as long as you ftp it from an non-US site.

There is no decrypt function, since **crypt** uses a one-way algorithm.

echo

Name

`echo` — output one or more strings

Description

```
echo(string arg1, string argn...);
```

Outputs all parameters.

See also: **print printf flush**

explode

Name

`explode` — split a string by string

Description

```
array explode(string separator, string string);
```

Returns an array of strings containing the elements separated by *`separator`*.

Example 1. explode() example

```
$pizza = "piece1 piece2 piece3 piece4 piece5 piece6";
$pieces = explode(" ", $pizza);
```

See also **split** and **implode**.

flush

Name

`flush` — flush the output buffer

Description

```
void flush(void);
```

Flushes the output buffers of PHP and whatever backend PHP is using (CGI, a web server, etc.) This effectively tries to push all the output so far to the user's browser.

htmlspecialchars

Name

`htmlspecialchars` — Convert special characters to HTML entities.

Description

```
string htmlspecialchars(string string);
```

Certain characters have special significance in HTML, and should be represented by HTML entities if they are to preserve their meanings. This function returns a string with these conversions made.

At present, the translations that are done are:

- '&' (ampersand) becomes '&amp;'
- '"' (double quote) becomes '&quot;'
- '<' (less than) becomes '&lt;'
- '>' (greater than) becomes '&gt;'

Note that this functions does not translate anything beyond what is listed above. For full entity translation, see **htmlentities**.

See also **htmlentities** and **nl2br**.

htmlentities

Name

`htmlentities` — Convert all applicable characters to HTML entities.

Description

```
string htmlentities(string string);
```

This function is identical to **htmlspecialchars** in all ways, except that all characters which have HTML entity equivalents are translated into these entities.

At present, the ISO-8859-1 character set is used.

See also **htmlspecialchars** and **nl2br**.

implode

Name

`implode` — join array elements with a string

Description

```
string implode(array pieces, string glue);
```

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, ":");
```

join

Name

`join` — join array elements with a string

Description

```
string join(array pieces, string glue);
```

join is an alias to **implode**, and is identical in every way.

nl2br

Name

`nl2br` — Converts newlines to HTML line breaks.

Description

```
string nl2br(string string);
```

Returns *`string`* with '<BR>' inserted before all newlines.

See also **htmlspecialchars** and **htmlentities**.

Ord

Name

`Ord` — return ASCII value of character

Description

```
int ord(string string);
```

Returns the ASCII value of the first character of string. This function complements **chr**.

Example 1. ord() example

```
if (ord($str) == 10) {
    echo("The first character of \$str is a line feed.\n");
}
```

print

Name

`print` — output a string

Description

```
print(string arg);
```

Outputs the parameter.

See also: **echo printf flush**

printf

Name

`printf` — output a formatted string

Description

```
print(string format, mixed args...);
```

Produces output according to *`format`*, which is described in the documentation for **sprintf**.

See also: **print**, **sprintf**, and **flush**.

QuoteMeta

Name

`QuoteMeta` — quote meta characters

Description

```
int quotemeta(string str);
```

Returns a version of str with a backslash character (`\`) before every character that is among these:

```
. \\ + * ? [ ^ ] ( $ )
```

rawurldecode

Name

`rawurldecode` — decode URL-encoded strings

Description

```
string rawurldecode(string str);
```

Returns a string in which the sequences with percent (`%`) signs followed by two hex digits have been replaced with literal characters. For example, the string

```
foo%20bar%40baz
```

decodes into

```
foo bar@baz
```

.

rawurlencode

Name

`rawurlencode` — URL-encode according to RFC1738

Description

```
string rawurlencode(string str);
```

Returns a string in which all non-alphanumeric characters except

```
-_.
```

have been replaced with a percent (`%`) sign followed by two hex digits. This is the encoding described in RFC1738 for protecting literal characters from being interpreted as special URL delimiters, and for protecting URL's from being mangled by transmission media with character conversions (like some email systems). For example, if you want to include a password in a 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'), '">';
```

setlocale

Name

`setlocale` — set locale information

Description

```
string setlocale(string category, string locale);
```

category is a string specifying the category of the functions affected by the locale setting:

- LC_ALL for all of the below
- LC_COLLATE for string comparison - not currently implemented in PHP
- LC_CTYPE for character classification and conversion, for example strtoupper()
- LC_MONETARY for localeconv() - not currently implemented in PHP
- LC_NUMERIC for decimal separator
- LC_TIME for time formats - not currently implemented in PHP

If locale is an 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. Invalid category name also causes a warning message.

sprintf

Name

`sprintf` — return a formatted string

Description

```
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.
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.
   - the argument is treated as an integer, and presented as a binary number.
   - the argument is treated as an integer, and presented as the character with that ASCII value.
   - the argument is treated as an integer, and presented as a decimal number.
   - the argument is treated as a double, and presented as a floating-point number.
   - the argument is treated as an integer, and presented as an octal number.
   - the argument is treated as and presented as a string.
   - the argument is treated as an integer and presented as a hexadecimal number (with lowercase letters).
   - the argument is treated as an integer and presented as a hexadecimal number (with uppercase letters).

See also: **printf**

Examples

Example 1. sprintf: zero-padded integers

```
$isodate = sprintf("%04d-%02d-%02d", $year, $month, $day);
```

strchr

Name

`strchr` — Find the first occurrence of a character.

Description

```
string strchr(string haystack, string needle);
```

This function is an alias for **strstr**, and is identical in every way.

StripSlashes

Name

`StripSlashes` — un-quote string quoted with addslashes

Description

```
string stripslashes(string str);
```

Returns a string with backslashes stripped off. (`\'` becomes `'` and so on.) Double backslashes are made into a single backslash.

strlen

Name

`strlen` — get string length

Description

```
int strlen(string str);
```

Returns the length of *`string`*.

strtok

Name

`strtok` — tokenize string

Description

```
string strtok(string arg1, string arg2);
```

strtok() is used to tokenize a string. That is, if you have a string like "This is an example string" you could tokenize this string into its individual words by using the space character as the token.

Example 1. strtok() example

```
    $string = "This is an example string";
    $tok = strtok($string," ");
    while($tok) {
        echo "Word=$tok<br>";
        $tok = strtok(" ");
    }
```

Note that only the first call to strtok uses the string argument. Every subsequent call to strtok only needs the token to use, as it keeps track of where it is in the current string. To start over, or to tokenize a new string you simply call strtok with the string argument again to initialize it. Note that you may put multiple tokens in the token parameter. The string will be tokenized when any one of the characters in the argument are found.

See also **split** and **explode**.

strrchr

Name

`strrchr` — Find the last occurrence of a character in a string.

Description

```
string strrchr(string haystack, string needle);
```

This function returns the portion of *`haystack`* which starts at the last occurrence of *`needle`* and goes until the end of *`haystack`*.

Returns false if *`needle`* is not found.

If *`needle`* is not found, false is returned.

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** and **strstr**.

strrev

Name

`strrev` — Reverse a string.

Description

```
string strrev(string string);
```

Returns *`string`*, reversed.

strstr

Name

`strstr` — Find first occurrence of a string.

Description

```
string strstr(string haystack, string needle);
```

Returns all of *`haystack`* from the first occurrence of *`needle`* to the end.

If *`needle`* is not found, returns false.

If *`needle`* is not a string, it is converted to an integer and applied as the ordinal value of a character.

See also **strrchr**, **substr**, and **ereg**.

strtolower

Name

`strtolower` — Make a string lowercase.

Description

```
string strtolower(string str);
```

Returns *`string`* with all alphabetic characters converted to lowercase.

Note that 'alphabetic' is determined by the current locale. This means that in i.e. the default "C" locale, characters such as umlaut-A (Ä) will not be converted.

See also **strtoupper** and **ucfirst**.

strtoupper

Name

`strtoupper` — Make a string uppercase.

Description

```
string strtoupper(string string);
```

Returns *`string`* with all alphabetic characters converted to uppercase.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

See also **strtolower** and **ucfirst**.

strtr

Name

`strtr` — Translate certain characters.

Description

```
string strtr(string str, string from, string to);
```

This function operates on *`str`*, translating all occurrences of each character in *`from`* to the corresponding character in *`to`* and returning the result.

If *`from`* and *`to`* are different lengths, the extra characters in the longer of the two are ignored.

Example 1. strtr() example

```
$addr = strtr( "äåö", "aao", $addr );
```

See also **ereg_replace**.

substr

Name

`substr` — Return part of a string.

Description

```
string substr(string string, int start, int length);
```

Substr returns the portion of *`string`* specified by the *`start`* and *`length`* parameters.

If *`start`* is positive, the returned string will start at the *`start`*'th character of *`string`*.

If *`start`* is negative, the returned string will start at the *`start`*'th character from the end of *`string`*.

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, 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`*.

See also **strrchr** and **ereg**.

ucfirst

Name

`ucfirst` — Make a string's first character uppercase.

Description

```
string ucfirst(string str);
```

Capitalizes the first character of *`str`* if that character is alphabetic.

Note that 'alphabetic' is determined by the current locale. For instance, in the default "C" locale characters such as umlaut-a (ä) will not be converted.

See also **strtoupper** and **strtolower**.

md5

Name

`md5` — calculate the md5 hash of a string

Description

```
string md5(string str);
```

Calculates the MD5 hash of *`str`*.

soundex

Name

`soundex` — calculate the soundex key of a string

Description

```
string soundex(string str);
```

Calculates the soundex key of *`str`*.

Soundex keys have the property that words pronounced similarly produce the same soundex key, and can thus be used to simplify searches in databases where you know the pronunciation but not the spelling. This soundex function returns a string 4 characters long, starting with a letter.

This particular soundex function is one described by Donald Knuth in "The Art Of Computer Programming, vol. 3: Sorting And Searching", Addison-Wesley (1973), pp. 391-392.

Example 1. Soundex Examples

```
soundex("Euler") == soundex("Ellery") == 'E460';
soundex("Gauss") == soundex("Ghosh") == 'G200';
soundex("Knuth") == soundex("Kant") == 'H416';
soundex("Lloyd") == soundex("Ladd") == 'L300';
soundex("Lukasiewicz") == soundex("Lissajous") == 'L222';
```

parse_str

Name

`parse_str` — parses the string into variables

Description

```
void soundex(string str);
```

Parses *`str`* as if it were the query string based via a URL and sets variables in the current scope.

Example 1. Using parse_str

```
$str = "first=value&second[]=this+works&second[]=another";
$arr = parse_str($str);
echo $first; /* prints "value" */
echo $second[0]; /* prints "this works" */
echo $second[1]; /* prints "another" */
```

Reference 32. URL functions

parse_url

Name

`parse_url` — parse a query string like PHP does with form data

Description

```
array parse_url(string url);
```

Returns: This function returns an associative array returning any of the various components of the URL that are present. This includes the "scheme", "host", "port", "user", "pass", "path", "query", and "fragment".

urldecode

Name

`urldecode` — decodes URL-encoded string

Description

```
string urldecode(string str);
```

Decodes any `%##` encoding in the given string. The decoded string is returned.

Example 1. urldecode() example

```
$a = split ('&', $querystring);
$i = 0;
while ($i < count ($a)) {
  $b = split ('=', $a [$i]);
  echo 'Value for parameter ', htmlspecialchars (urldecode ($b [0])),
       ' is ', htmlspecialchars (urldecode ($b [1])), "<BR>";
  $i++;
}
```

See also **urlencode**

urlencode

Name

`urlencode` — URL-encodes string

Description

```
string urlencode(string str);
```

Returns a string in which all non-alphanumeric characters except `-_.` have been replaced with a percent (`%`) sign followed by two hex digits and spaces encoded as plus (`+`) signs. It is encoded the same way that the posted data from a WWW form is encoded, that is the same way as in `application/x-www-form-urlencoded` media type. This differs from the RFC1738 encoding (see **rawurlencode**) in that for historical reasons, spaces are encoded as plus (+) signs. This function is convenient when encoding a string to be used in a query part of an URL, as a convinient way to pass variables to the next page:

Example 1. urlencode() example

```
echo '<A HREF="mycgi?foo=', urlencode ($userinput), '">';
```

See also **urldecode**

base64_encode

Name

`base64_encode` — encodes data with MIME base64

Description

```
string base64_encode(string data);
```

base64_encode returns *`data`* encoded with base64. This encoding is designed to make binary data survive transport through transport layers that are not 8-bit clean, such as mail bodies.

Base64-encoded data takes about 33% more space than the original data.

See also: **base64_decode**, RFC-2045 section 6.8.

base64_decode

Name

`base64_decode` — decodes data encoded with MIME base64

Description

```
string base64_decode(string encoded_data);
```

base64_decode decodes *`encoded_data`* and returns the original data. The returned data may be binary.

See also: **base64_encode**, RFC-2045 section 6.8.

Reference 33. Variable functions

gettype

Name

`gettype` — Get the type of a variable.

Description

```
string gettype(mixed var);
```

Returns the type of the PHP variable *`var`*.

Possibles values for the returned string are:

- "integer"
- "double"
- "string"
- "array"
- "class"
- "object"
- "unknown type"

See also **settype**.

intval

Name

`intval` — Get integer value of a variable.

Description

```
int intval(mixed var, int base);
```

Returns the integer value of *`var`*, using the specified base for the conversion (the default is base 10).

`var` may be any scalar type. You cannot use **intval** on arrays or objects.

See also **doubleval**, **strval**, **settype** and Type juggling.

doubleval

Name

`doubleval` — Get double value of a variable.

Description

```
double doubleval(mixed var);
```

Returns the double (floating point) value of *`var`*.

`var` may be any scalar type. You cannot use **doubleval** on arrays or objects.

See also **intval**, **strval**, **settype** and Type juggling.

strval

Name

`strval` — Get string value of a variable.

Description

```
string strval(mixed var);
```

Returns the string value of *`var`*.

`var` may be any scalar type. You cannot use **strval** on arrays or objects.

See also **doubleval**, **intval**, **settype** and Type juggling.

is_array

Name

`is_array` — Finds whether a variable is an array.

Description

```
int is_array(mixed var);
```

Returns true if *`var`* is an array, false otherwise.

See also **is_double**, **is_string**, **is_long**, and **is_object**.

is_double

Name

`is_double` — Finds whether a variable is a double.

Description

```
int is_double(mixed var);
```

Returns true if `var` is a double, false otherwise.

See also **is_array**, **is_real**, **is_string**, **is_long**, and **is_object**.

is_integer

Name

`is_integer` — Find whether a variable is an integer.

Description

```
int is_integer(mixed var);
```

This function is an alias for **is_long**.

See also **is_double**, **is_string**, **is_real**, **is_array**, and **is_long**.

is_long

Name

`is_long` — Finds whether a variable is an integer.

Description

```
int is_long(mixed var);
```

Returns true if `var` is an integer (long), false otherwise.

See also **is_double**, **is_real**, **is_string**, **is_array**, and **is_integer**.

is_object

Name

`is_object` — Finds whether a variable is an object.

Description

```
int is_object(mixed var);
```

Returns true if *`var`* is an object, false otherwise.

See also **is_double**, **is_string**, **is_long**, and **is_object**.

is_real

Name

`is_real` — Finds whether a variable is a real.

Description

```
int is_real(mixed var);
```

This function is an alias for **is_double**.

See also **is_double**, **is_string**, **is_integer**, **is_array**, and **is_long**.

is_string

Name

`is_string` — Finds whether a variable is a string.

Description

```
int is_string(mixed var);
```

Returns true if *`var`* is a string, false otherwise.

See also **is_double**, **is_real**, **is_array**, **is_long**, and **is_object**.

settype

Name

`settype` — Set the type of a variable.

Description

```
int settype(string var, string type);
```

Set the type of variable *`var`* to *`type`*.

Possibles values of *`type`* are:

- "integer"
- "double"
- "string"
- "array"
- "object"

Returns true if successful; otherwise returns false.

See also **gettype**.

III. 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

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

E_CORE_WARNING

Hitchhiker's guide to PHP internals

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 php3.ini (debugger.port) and enable it (debugger.enabled).
2. Set up a TCP listener on that port somewhere (for example **socket -l -s 1400** on UNIX).
3. In your code, run "debugger_on(*`host`*)", where *`host`* is the IP number or name of the host running the TCP listener.

Now, all warnings, notices etc. will show up on that listener socket, *even if you them turned off with* ***error_reporting***.

Debugger Protocol

The debugger protocol is line-based. Each line has a *type*, and several lines comprise a *message*. Each message starts with 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 addressof 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.

Name	Meaning
`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
```