Network

Introduction

Provides various networking functions.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

There is no installation needed to use these functions; they are part of the PHP core.

Runtime Configuration

The behaviour of these functions is affected by settings in *php.ini*.

Network Configuration Options

Name	Default	Changeable	Changelog
define_syslog_variables	"0"	PHP_INI_ALL	

For further details and definitions of the PHP_INI_* constants, see the php.ini directives.

Here's a short explanation of the configuration directives.

`define_syslog_variables` boolean
: Whether or not to define the various syslog variables (e.g. $LOG_PID, $LOG_CRON, etc.). Turning it off is a good idea performance-wise. At runtime, you can define these variables by calling define_syslog_variables().

Resource Types

This extension defines a file pointer resource returned by fsockopen() and pfsockopen().

Predefined Constants

The constants below are always available as part of the PHP core.

openlog() **Options**

Constant	Description
LOG_CONS	if there is an error while sending data to the system logger, write directly to the system console
LOG_NDELAY	open the connection to the logger immediately
LOG_ODELAY	(default) delay opening the connection until the first message is logged
LOG_NOWAIT	
LOG_PERROR	print log message also to standard error
LOG_PID	include PID with each message

openlog() **Facilities**

Constant	Description
LOG_AUTH	security/authorization messages (use LOG_AUTHPRIV instead in systems where that constant is defined)
LOG_AUTHPRIV	security/authorization messages (private)
LOG_CRON	clock daemon (cron and at)
LOG_DAEMON	other system daemons
LOG_KERN	kernel messages
LOG_LOCAL0 ... LOG_LOCAL7	reserved for local use, these are not available in Windows
LOG_LPR	line printer subsystem
LOG_MAIL	mail subsystem

LOG_NEWS	USENET news subsystem
LOG_SYSLOG	messages generated internally by syslogd
LOG_USER	generic user-level messages
LOG_UUCP	UUCP subsystem

syslog() **Priorities (in descending order)**

Constant	**Description**
LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal, but significant, condition
LOG_INFO	informational message
LOG_DEBUG	debug-level message

dns_get_record() **Options**

Constant	**Description**
DNS_A	IPv4 Address Resource
DNS_MX	Mail Exchanger Resource
DNS_CNAME	Alias (Canonical Name) Resource
DNS_NS	Authoritative Name Server Resource
DNS_PTR	Pointer Resource
DNS_HINFO	Host Info Resource (See IANA's » Operating System Names for the meaning of these values)
DNS_SOA	Start of Authority Resource

DNS_TXT	Text Resource
DNS_ANY	Any Resource Record. On most systems this returns all resource records, however it should not be counted upon for critical uses. Try DNS_ALL instead.
DNS_AAAA	IPv6 Address Resource
DNS_ALL	Iteratively query the name server for each available record type.

Network Functions

checkdnsrr

checkdnsrr -- Check DNS records corresponding to a given Internet host name or IP address

Description

bool **checkdnsrr** (string `$host` [, string `$type`])

Searches DNS for records of type *`type`* corresponding to *`host`*.

Parameters

`host`
: *`host`* may either be the IP address in dotted-quad notation or the host name.

`type`
: *`type`* may be any one of: A, MX, NS, SOA, PTR, CNAME, AAAA, A6, SRV, NAPTR, TXT or ANY. The default is MX.

Return Values

Returns **TRUE** if any records are found; returns **FALSE** if no records were found or if an error occurred.

ChangeLog

Version	Description
5.2.4	TXT *`type`* was added.
5.0.0	AAAA *`type`* was added.

Notes

Note
This function is not implemented on Windows platforms. Try the » PEAR class » Net_DNS.

See Also

- dns_get_record()
- getmxrr()
- gethostbyaddr()
- gethostbyname()
- gethostbynamel()
- the named(8) manual page

closelog

closelog -- Close connection to system logger

Description

bool **closelog** (void)

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

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- define_syslog_variables()
- syslog()
- openlog()

define_syslog_variables

define_syslog_variables -- Initializes all syslog related constants

Description

void **define_syslog_variables** (void)

Initializes all constants used in the syslog functions.

Return Values

No value is returned.

See Also

- openlog()
- syslog()
- closelog()

dns_check_record

dns_check_record -- Alias of checkdnsrr()

Description

This function is an alias of: checkdnsrr().

dns_get_mx

dns_get_mx -- Alias of getmxrr()

Description

This function is an alias of: getmxrr().

dns_get_record

dns_get_record -- Fetch DNS Resource Records associated with a hostname

Description

array **dns_get_record** (string `$hostname` [, int `$type`])

array **dns_get_record** (string `$hostname`, int `$type`, array `&$authns`, array `&$addtl`)

Fetch DNS Resource Records associated with the given *`hostname`*.

Parameters

`hostname`

`hostname` should be a valid DNS hostname such as "www.example.com". Reverse lookups can be generated using in-addr.arpa notation, but gethostbyaddr() is more suitable for the majority of reverse lookups.

Note
Per DNS standards, email addresses are given in user.host format (for example: hostmaster.example.com as opposed to hostmaster@example.com), be sure to check this value and modify if necessary before using it with a functions such as mail().

`type`

By default, dns_get_record() will search for any resource records associated with *`hostname`*. To limit the query, specify the optional *`type`* parameter. May be any one of the following: **DNS_A**, **DNS_CNAME**, **DNS_HINFO**, **DNS_MX**, **DNS_NS**, **DNS_PTR**, **DNS_SOA**, **DNS_TXT**, **DNS_AAAA**, **DNS_SRV**, **DNS_NAPTR**, **DNS_A6**, **DNS_ALL** or **DNS_ANY**. The default is *DNS_ANY*.

Note
Because of eccentricities in the performance of libresolv between platforms, **DNS_ANY** will not always return every record, the slower **DNS_ALL** will collect all records more reliably.

`authns`

Passed by reference and, if given, will be populated with Resource Records for the *Authoritative Name Servers*.

`addtl`

Passed by reference and, if given, will be populated with any *Additional Records*.

Return Values

This function returns an array of associative arrays. Each associative array contains *at minimum* the following keys:

Basic DNS attributes

Attribute	Meaning
host	The record in the DNS namespace to which the rest of the associated data refers.
class	dns_get_record() only returns Internet class records and as such this parameter will always return *IN*.
type	String containing the record type. Additional attributes will also be contained in the resulting array dependant on the value of type. See table below.
ttl	Time To Live remaining for this record. This will *not* equal the record's original ttl, but will rather equal the original ttl minus whatever length of time has passed since the authoritative name server was queried.

Other keys in associative arrays dependant on 'type'

Type	Extra Columns
A	*ip*: An IPv4 addresses in dotted decimal notation.
MX	*pri*: Priority of mail exchanger. Lower numbers indicate greater priority. *target*: FQDN of the mail exchanger. See also dns_get_mx().
CNAME	*target*: FQDN of location in DNS namespace to which the record is aliased.
NS	*target*: FQDN of the name server which is authoritative for this hostname.

PTR	*target*: Location within the DNS namespace to which this record points.
TXT	*txt*: Arbitrary string data associated with this record.
HINFO	*cpu*: IANA number designating the CPU of the machine referenced by this record. *os*: IANA number designating the Operating System on the machine referenced by this record. See IANA's » Operating System Names for the meaning of these values.
SOA	*mname*: FQDN of the machine from which the resource records originated. *rname*: Email address of the administrative contain for this domain. *serial*: Serial # of this revision of the requested domain. *refresh*: Refresh interval (seconds) secondary name servers should use when updating remote copies of this domain. *retry*: Length of time (seconds) to wait after a failed refresh before making a second attempt. *expire*: Maximum length of time (seconds) a secondary DNS server should retain remote copies of the zone data without a successful refresh before discarding. *minimum-ttl*: Minimum length of time (seconds) a client can continue to use a DNS resolution before it should request a new resolution from the server. Can be overridden by individual resource records.
AAAA	*ipv6*: IPv6 address
A6 (PHP >= 5.1.0)	*masklen*: Length (in bits) to inherit from the target specified by `chain`. *ipv6*: Address for this specific record to merge with `chain`. *chain*: Parent record to merge with `ipv6` data.
SRV	*pri*: (Priority) lowest priorities should be used first. *weight*: Ranking to weight which of commonly prioritized `targets` should be chosen at random. *target* and *port*: hostname and port where the requested service can be found. For additional information see: » RFC 2782
NAPTR	*order* and *pref*: Equivalent to `pri` and `weight` above. *flags*, *services*, *regex*, and *replacement*: Parameters as defined by

	» RFC 2915.

Examples

Example #1 - Using dns_get_record()

```
<?php
$result = dns_get_record("php.net");
print_r($result);
?>
```

The above example will output something similar to:

```
Array
(
   [0] => Array
       (
           [host] => php.net
           [type] => MX
           [pri] => 5
           [target] => pair2.php.net
           [class] => IN
           [ttl] => 6765
       )

   [1] => Array
       (
           [host] => php.net
           [type] => A
           [ip] => 64.246.30.37
           [class] => IN
           [ttl] => 8125
       )

)
```

Example #2 - Using dns_get_record() and DNS_ANY

Since it's very common to want the IP address of a mail server once the MX record has been resolved, dns_get_record() also returns an array in *addtl* which contains associate records. *authns* is returned as well containing a list of authoritative name servers.

```
<?php
/* Request "ANY" record for php.net,
  and create $authns and $addtl arrays
  containing list of name servers and
  any additional records which go with
  them */
$result = dns_get_record("php.net", DNS_ANY, $authns, $addtl);
echo "Result = ";
```

```
print_r($result);
echo "Auth NS = ";
print_r($authns);
echo "Additional = ";
print_r($addtl);
?>
```

The above example will output something similar to:

```
Result = Array
(
   [0] => Array
       (
           [host] => php.net
           [type] => MX
           [pri] => 5
           [target] => pair2.php.net
           [class] => IN
           [ttl] => 6765
       )

   [1] => Array
       (
           [host] => php.net
           [type] => A
           [ip] => 64.246.30.37
           [class] => IN
           [ttl] => 8125
       )

)
Auth NS = Array
(
   [0] => Array
       (
           [host] => php.net
           [type] => NS
           [target] => remote1.easydns.com
           [class] => IN
           [ttl] => 10722
       )

   [1] => Array
       (
           [host] => php.net
           [type] => NS
           [target] => remote2.easydns.com
           [class] => IN
           [ttl] => 10722
       )

   [2] => Array
       (
           [host] => php.net
           [type] => NS
           [target] => ns1.easydns.com
           [class] => IN
           [ttl] => 10722
       )
```

```
    [3] => Array
        (
            [host] => php.net
            [type] => NS
            [target] => ns2.easydns.com
            [class] => IN
            [ttl] => 10722
        )

)
Additional = Array
(
    [0] => Array
        (
            [host] => pair2.php.net
            [type] => A
            [ip] => 216.92.131.5
            [class] => IN
            [ttl] => 6766
        )

    [1] => Array
        (
            [host] => remote1.easydns.com
            [type] => A
            [ip] => 64.39.29.212
            [class] => IN
            [ttl] => 100384
        )

    [2] => Array
        (
            [host] => remote2.easydns.com
            [type] => A
            [ip] => 212.100.224.80
            [class] => IN
            [ttl] => 81241
        )

    [3] => Array
        (
            [host] => ns1.easydns.com
            [type] => A
            [ip] => 216.220.40.243
            [class] => IN
            [ttl] => 81241
        )

    [4] => Array
        (
            [host] => ns2.easydns.com
            [type] => A
            [ip] => 216.220.40.244
            [class] => IN
            [ttl] => 81241
        )

)
```

Notes

Note
This function is not implemented on Windows platforms, nor does it (currently) work on *BSD systems (including Mac). Try the » PEAR class » Net_DNS.

See Also

- dns_get_mx()
- dns_check_record()

fsockopen

fsockopen -- Open Internet or Unix domain socket connection

Description

resource **fsockopen** (string `$hostname` [, int `$port` [, int `&$errno` [, string `&$errstr` [, float `$timeout`]]]])

Initiates a socket connection to the resource specified by *`hostname`*.

PHP supports targets in the Internet and Unix domains as described in List of Supported Socket Transports. A list of supported transports can also be retrieved using stream_get_transports().

The socket will by default be opened in blocking mode. You can switch it to non-blocking mode by using stream_set_blocking().

Parameters

`hostname`
: If you have compiled in OpenSSL support, you may prefix the *`hostname`* with either *ssl://* or *tls://* to use an SSL or TLS client connection over TCP/IP to connect to the remote host.

`port`
: The port number.

`errno`
: If provided, holds the system level error number that occurred in the system-level *connect()* call. If the value returned in *`errno`* is *0* and the function returned **FALSE**, it is an indication that the error occurred before the *connect()* call. This is most likely due to a problem initializing the socket.

`errstr`
: The error message as a string.

`timeout`
: The connection timeout, in seconds.

Note
If you need to set a timeout for reading/writing data over the socket, use stream_set_timeout(), as the *`timeout`* parameter to fsockopen() only applies while connecting the socket.

Return Values

fsockopen() returns a file pointer which may be used together with the other file functions (such as fgets(), fgetss(), fwrite(), fclose(), and feof()). If the call fails, it will return **FALSE**

ChangeLog

Version	Description
4.3.0	Added support for the *timeout* parameter on win32.
4.3.0	SSL and TLS over TCP/IP support was added.
4.0.0	UDP support was added.
3.0.9	The *timeout* parameter was added.

Examples

Example #3 - fsockopen() Example

```
<?php
$fp = fsockopen("www.example.com", 80, $errno, $errstr, 30);
if (!$fp) {
   echo "$errstr ($errno)<br />\n";
} else {
   $out = "GET / HTTP/1.1\r\n";
   $out .= "Host: www.example.com\r\n";
   $out .= "Connection: Close\r\n\r\n";

   fwrite($fp, $out);
   while (!feof($fp)) {
       echo fgets($fp, 128);
   }
   fclose($fp);
}
?>
```

Example #4 - Using UDP connection

The example below shows how to retrieve the day and time from the UDP service

"daytime" (port 13) in your own machine.

```
<?php
$fp = fsockopen("udp://127.0.0.1", 13, $errno, $errstr);
if (!$fp) {
   echo "ERROR: $errno - $errstr<br />\n";
} else {
   fwrite($fp, "\n");
   echo fread($fp, 26);
   fclose($fp);
}
?>
```

Notes

Note

Depending on the environment, the Unix domain or the optional connect timeout may not be available.

Warning

UDP sockets will sometimes appear to have opened without an error, even if the remote host is unreachable. The error will only become apparent when you read or write data to/from the socket. The reason for this is because UDP is a "connectionless" protocol, which means that the operating system does not try to establish a link for the socket until it actually needs to send or receive data.

Note

When specifying a numerical IPv6 address (e.g. *fe80::1*), you must enclose the IP in square brackets?for example, *tcp://[fe80::1]:80*.

See Also

- pfsockopen()
- stream_set_blocking()
- stream_set_timeout()
- fgets()
- fgetss()
- fwrite()
- fclose()
- feof()

- The Curl extension

gethostbyaddr

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`*.

Parameters

`ip_address`
The host IP address.

Return Values

Returns the host name or the unmodified *`ip_address`* on failure.

Examples

Example #5 - A simple gethostbyaddr() example

```
<?php
$hostname = gethostbyaddr($_SERVER['REMOTE_ADDR']);

echo $hostname;
?>
```

See Also

- gethostbyname()
- gethostbynamel()

gethostbyname

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`*.

Parameters

`hostname`
The host name.

Return Values

Returns the IP address or a string containing the unmodified *`hostname`* on failure.

Examples

Example #6 - A simple gethostbyname() example

```
<?php
$ip = gethostbyname('www.example.com');

echo $ip;
?>
```

See Also

- gethostbyaddr()
- gethostbynamel()

gethostbynamel

gethostbynamel -- Get a list of IP addresses corresponding to a given Internet host name

Description

array **gethostbynamel** (string `$hostname`)

Returns a list of IP addresses to which the Internet host specified by *`hostname`* resolves.

Parameters

`hostname`
The host name.

Return Values

Returns an array of IP addresses or **FALSE** if *`hostname`* could not be resolved.

Examples

Example #7 - gethostbynamel() example

```
<?php
$hosts = gethostbynamel('www.example.com');
print_r($hosts);
?>
```

The above example will output:

```
Array
(
   [0] => 192.0.34.166
)
```

See Also

- gethostbyname()
- gethostbyaddr()
- checkdnsrr()
- getmxrr()
- the *named(8)* manual page

getmxrr

getmxrr -- Get MX records corresponding to a given Internet host name

Description

bool **getmxrr** (string `$hostname`, array `&$mxhosts` [, array `&$weight`])

Searches DNS for MX records corresponding to *`hostname`*.

Parameters

`hostname`
: The Internet host name.

`mxhosts`
: A list of the MX records found is placed into the array *`mxhosts`*.

`weight`
: If the *`weight`* array is given, it will be filled with the weight information gathered.

Return Values

Returns **TRUE** if any records are found; returns **FALSE** if no records were found or if an error occurred.

Notes

Note
This function should not be used for the purposes of address verification. Only the mailexchangers found in DNS are returned, however, according to » RFC 2821 when no mail exchangers are listed, *`hostname`* itself should be used as the only mail exchanger with a priority of *0*.

Note
This function is not implemented on Windows platforms. Try the » PEAR class » Net_DNS.

See Also

- checkdnsrr()
- dns_get_record()
- gethostbyname()
- gethostbynamel()
- gethostbyaddr()
- the *named(8)* manual page

getprotobyname

getprotobyname -- Get protocol number associated with protocol name

Description

int **getprotobyname** (string `$name`)

getprotobyname() returns the protocol number associated with the protocol *`name`* as per */etc/protocols*.

Parameters

`name`
The protocol name.

Return Values

Returns the protocol number or -1 if the protocol is not found.

Examples

Example #8 - getprotobyname() example

```
<?php
$protocol = 'tcp';
$get_prot = getprotobyname($protocol);
if ($get_prot == -1) {
   echo 'Invalid Protocol';
} else {
   echo 'Protocol #' . $get_prot;
}
?>
```

See Also

- getprotobynumber()

getprotobynumber

getprotobynumber -- Get protocol name associated with protocol number

Description

string **getprotobynumber** (int `$number`)

getprotobynumber() returns the protocol name associated with protocol *`number`* as per */etc/protocols*.

Parameters

`number`
The protocol number.

Return Values

Returns the protocol name as a string.

See Also

- getprotobyname()

getservbyname

getservbyname -- Get port number associated with an Internet service and protocol

Description

int **getservbyname** (string `$service`, string `$protocol`)

getservbyname() returns the Internet port which corresponds to *`service`* for the specified *`protocol`* as per */etc/services*.

Parameters

`service`
The Internet service name, as a string.

`protocol`
`protocol` is either *"tcp"* or *"udp"* (in lowercase).

Return Values

Returns the port number, or **FALSE** if *`service`* or *`protocol`* is not found.

Examples

Example #9 - getservbyname() example

```
<?php
$services = array('http', 'ftp', 'ssh', 'telnet', 'imap',
'smtp', 'nicname', 'gopher', 'finger', 'pop3', 'www');

foreach ($services as $service) {
   $port = getservbyname($service, 'tcp');
   echo $service . ": " . $port . "<br />\n";
}
?>
```

See Also

- getservbyport()
- » http://www.iana.org/assignments/port-numbers for a complete list of port numbers.

getservbyport

getservbyport -- Get Internet service which corresponds to port and protocol

Description

string **getservbyport** (int `$port`, string `$protocol`)

getservbyport() returns the Internet service associated with *`port`* for the specified *`protocol`* as per */etc/services*.

Parameters

`port`
: The port number.

`protocol`
: *`protocol`* is either *"tcp"* or *"udp"* (in lowercase).

Return Values

Returns the Internet service name as a string.

See Also

- getservbyname()

header

header -- Send a raw HTTP header

Description

void **header** (string `$string` [, bool `$replace` [, int `$http_response_code`]])

header() is used to send a raw HTTP header. See the » HTTP/1.1 specification for more information on HTTP headers.

Remember that header() must be called before any actual output is sent, either by normal HTML tags, blank lines in a file, or from PHP. It is a very common error to read code with **include()**, or **require()**, functions, or another file access function, and have spaces or empty lines that are output before header() is called. The same problem exists when using a single PHP/HTML file.

```
<html>
<?php
/* This will give an error. Note the output
* above, which is before the header() call */
header('Location: http://www.example.com/');
?>
```

Parameters

`string`
The header string. There are two special-case header calls. The first is a header that starts with the string " *HTTP/* " (case is not significant), which will be used to figure out the HTTP status code to send. For example, if you have configured Apache to use a PHP script to handle requests for missing files (using the *ErrorDocument* directive), you may want to make sure that your script generates the proper status code.

```
<?php
header("HTTP/1.0 404 Not Found");
?>
```

The second special case is the "Location:" header. Not only does it send this header back to the browser, but it also returns a *REDIRECT* (302) status code to the browser unless some *3xx* status code has already been set.

```
<?php
header("Location: http://www.example.com/"); /* Redirect browser */

/* Make sure that code below does not get executed when we redirect. */
exit;
?>
```

replace
The optional *replace* parameter indicates whether the header should replace a previous similar header, or add a second header of the same type. By default it will replace, but if you pass in **FALSE** as the second argument you can force multiple headers of the same type. For example:

```
<?php
header('WWW-Authenticate: Negotiate');
header('WWW-Authenticate: NTLM', false);
?>
```

http_response_code
Forces the HTTP response code to the specified value.

Return Values

No value is returned.

ChangeLog

Version	Description
4.4.2 and 5.1.2	This function now prevents more than one header to be sent at once as a protection against header injection attacks.
4.3.0	The *http_response_code* parameter was added.
4.0.4	The *replace* parameter was added.

Examples

Example #10 - Download dialog

If you want the user to be prompted to save the data you are sending, such as a generated PDF file, you can use the » Content-Disposition header to supply a recommended filename and force the browser to display the save dialog.

```
<?php
// We'll be outputting a PDF
header('Content-type: application/pdf');
```

```
// It will be called downloaded.pdf
header('Content-Disposition: attachment; filename="downloaded.pdf"');

// The PDF source is in original.pdf
readfile('original.pdf');
?>
```

Example #11 - Caching directives

PHP scripts often generate dynamic content that must not be cached by the client browser or any proxy caches between the server and the client browser. Many proxies and clients can be forced to disable caching with:

```
<?php
header("Cache-Control: no-cache, must-revalidate"); // HTTP/1.1
header("Expires: Mon, 26 Jul 1997 05:00:00 GMT"); // Date in the past
?>
```

Note

You may find that your pages aren't cached even if you don't output all of the headers above. There are a number of options that users may be able to set for their browser that change its default caching behavior. By sending the headers above, you should override any settings that may otherwise cause the output of your script to be cached.

Additionally, session_cache_limiter() and the *session.cache_limiter* configuration setting can be used to automatically generate the correct caching-related headers when sessions are being used.

Notes

Note

As of PHP 4, you can use output buffering to get around this problem, with the overhead of all of your output to the browser being buffered in the server until you send it. You can do this by calling ob_start() and ob_end_flush() in your script, or setting the *output_buffering* configuration directive on in your *php.ini* or server configuration files.

Note

The HTTP status header line will always be the first sent to the client, regardless of the actual header() call being the first or not. The status may be overridden by calling header() with a new status line at any time unless the HTTP headers have already been sent.

Note

There is a bug in Microsoft Internet Explorer 4.01 that prevents this from working. There is no workaround. There is also a bug in Microsoft Internet Explorer 5.5 that interferes with this, which can be resolved by upgrading to Service Pack 2 or later.

Note

If safe mode is enabled the uid of the script is added to the *realm* part of the *WWW-Authenticate* header if you set this header (used for HTTP Authentication).

Note

HTTP/1.1 requires an absolute URI as argument to » Location: including the scheme, hostname and absolute path, but some clients accept relative URIs. You can usually use *$_SERVER['HTTP_HOST']*, *$_SERVER['PHP_SELF']* and dirname() to make an absolute URI from a relative one yourself:

```
<?php
/* Redirect to a different page in the current directory that was requested
*/
$host  = $_SERVER['HTTP_HOST'];
$uri   = rtrim(dirname($_SERVER['PHP_SELF']), '/\\');
$extra = 'mypage.php';
header("Location: http://$host$uri/$extra");
exit;
?>
```

Note

Session ID is not passed with Location header even if session.use_trans_sid is enabled. It must by passed manually using **SID** constant.

See Also

- headers_sent()
- setcookie()
- The section on HTTP authentication

headers_list

headers_list -- Returns a list of response headers sent (or ready to send)

Description

array **headers_list** (void)

headers_list() will return a list of headers to be sent to the browser / client. To determine whether or not these headers have been sent yet, use headers_sent().

Return Values

Returns a numerically indexed array of headers.

Examples

Example #12 - Examples using headers_list()

```
<?php

/* setcookie() will add a response header on its own */
setcookie('foo', 'bar');

/* Define a custom response header
  This will be ignored by most clients */
header("X-Sample-Test: foo");

/* Specify plain text content in our response */
header('Content-type: text/plain');

/* What headers are going to be sent? */
var_dump(headers_list());

?>
```

The above example will output:

```
array(4) {
 [0]=>
 string(23) "X-Powered-By: PHP/5.1.3"
 [1]=>
 string(19) "Set-Cookie: foo=bar"
 [2]=>
 string(18) "X-Sample-Test: foo"
 [3]=>
 string(24) "Content-type: text/plain"
}
```

See Also

- headers_sent()
- header()
- setcookie()
- apache_response_headers()

headers_sent

headers_sent -- Checks if or where headers have been sent

Description

bool **headers_sent** ([string `&$file` [, int `&$line`]])

Checks if or where headers have been sent.

You can't add any more header lines using the header() function once the header block has already been sent. Using this function you can at least prevent getting HTTP header related error messages. Another option is to use Output Buffering.

Parameters

`file`
If the optional *`file`* and *`line`* parameters are set, headers_sent() will put the PHP source file name and line number where output started in the *`file`* and *`line`* variables.

`line`
The line number where the output started.

Return Values

headers_sent() will return **FALSE** if no HTTP headers have already been sent or **TRUE** otherwise.

ChangeLog

Version	Description
4.3.0	The optional *`file`* and *`line`* parameters were added.

Examples

Example #13 - Examples using headers_sent()

```
<?php
```

```
// If no headers are sent, send one
if (!headers_sent()) {
   header('Location: http://www.example.com/');
   exit;
}

// An example using the optional file and line parameters, as of PHP 4.3.0
// Note that $filename and $linenum are passed in for later use.
// Do not assign them values beforehand.
if (!headers_sent($filename, $linenum)) {
   header('Location: http://www.example.com/');
   exit;

// You would most likely trigger an error here.
} else {

   echo "Headers already sent in $filename on line $linenum\n" .
         "Cannot redirect, for now please click this <a " .
         "href=\"http://www.example.com\">link</a> instead\n";
   exit;
}

?>
```

See Also

- ob_start()
- trigger_error()
- headers_list()
- header() for a more detailed discussion of the matters involved.

inet_ntop

inet_ntop -- Converts a packed internet address to a human readable representation

Description

string **inet_ntop** (string `$in_addr`)

This function converts a 32bit IPv4, or 128bit IPv6 address (if PHP was built with IPv6 support enabled) into an address family appropriate string representation.

Parameters

`in_addr`
A 32bit IPv4, or 128bit IPv6 address.

Return Values

Returns a string representation of the address or **FALSE** on failure.

Examples

Example #14 - inet_ntop() **Example**

```
<?php
$packed = chr(127) . chr(0) . chr(0) . chr(1);
$expanded = inet_ntop($packed);

/* Outputs: 127.0.0.1 */
echo $expanded;

$packed = str_repeat(chr(0), 15) . chr(1);
$expanded = inet_ntop($packed);

/* Outputs: ::1 */
echo $expanded;
?>
```

Notes

ChangeLog

Version	Description
5.3.0	This function is now available on Windows platforms

See Also

- long2ip()
- ip2long()
- inet_pton()

inet_pton

inet_pton -- Converts a human readable IP address to its packed in_addr representation

Description

string **inet_pton** (string `$address`)

This function converts a human readable IPv4 or IPv6 address (if PHP was built with IPv6 support enabled) into an address family appropriate 32bit or 128bit binary structure.

Parameters

`address`
A human readable IPv4 or IPv6 address.

Return Values

Returns the *in_addr* representation of the given *`address`*

Examples

Example #15 - inet_pton() Example

```
<?php
$in_addr = inet_pton('127.0.0.1');

$in6_addr = inet_pton('::1');

?>
```

Notes

ChangeLog

Version	Description
5.3.0	This function is now available on Windows platforms

See Also

- ip2long()
- long2ip()
- inet_ntop()

ip2long

ip2long -- Converts a string containing an (IPv4) Internet Protocol dotted address into a proper address

Description

int **ip2long** (string `$ip_address`)

The function ip2long() generates an IPv4 Internet network address from its Internet standard format (dotted string) representation.

ip2long() will also work with non-complete IP addresses. Read » http://publibn.boulder.ibm.com/doc_link/en_US/a_doc_lib/libs/commtrf2/inet_addr.htm for more info.

Parameters

`ip_address`
A standard format address.

Return Values

Returns the IPv4 address or **FALSE** if *`ip_address`* is invalid.

ChangeLog

Version	Description
5.0.0	Prior to this version, ip2long() returned -1 on failure.

Examples

Example #16 - ip2long() Example

```
<?php
$ip = gethostbyname('www.example.com');
$out = "The following URLs are equivalent:<br />\n";
$out .= 'http://www.example.com/, http://' . $ip . '/, and http://' .
sprintf("%u", ip2long($ip)) . "/<br />\n";
```

```
echo $out;
?>
```

Example #17 - Displaying an IP address

This second example shows how to print a converted address with the printf() function in both PHP 4 and PHP 5:

```
<?php
$ip   = gethostbyname('www.example.com');
$long = ip2long($ip);

if ($long == -1 || $long === FALSE) {
   echo 'Invalid IP, please try again';
} else {
   echo $ip   . "\n";           // 192.0.34.166
   echo $long . "\n";           // -1073732954
   printf("%u\n", ip2long($ip)); // 3221234342
}
?>
```

Example #18 - IP validation

ip2long() should not be used as the sole form of IP validation. Combine it with long2ip() :

```
<?php
// make sure IPs are valid. also converts a non-complete IP into
// a proper dotted quad as explained below.
$ip = long2ip(ip2long("127.0.0.1")); // "127.0.0.1"
$ip = long2ip(ip2long("10.0.0")); // "10.0.0.0"
$ip = long2ip(ip2long("10.0.256")); // "10.0.1.0"
?>
```

Notes

Note

Because PHP's integer type is signed, and many IP addresses will result in negative integers, you need to use the "%u" formatter of sprintf() or printf() to get the string representation of the unsigned IP address.

Note
ip2long() will return **FALSE** for the IP *255.255.255.255* in PHP 5 <= 5.0.2. It was fixed in PHP 5.0.3 where it returns *-1* (same as PHP 4).

See Also

- long2ip()
- sprintf()

long2ip

long2ip -- Converts an (IPv4) Internet network address into a string in Internet standard dotted format

Description

string **long2ip** (int `$proper_address`)

The function long2ip() generates an Internet address in dotted format (i.e.: aaa.bbb.ccc.ddd) from the proper address representation.

Parameters

`proper_address`
A proper address representation.

Return Values

Returns the Internet IP address as a string.

See Also

- ip2long()

openlog

openlog -- Open connection to system logger

Description

bool **openlog** (string `$ident`, int `$option`, int `$facility`)

openlog() opens a connection to the system logger for a program.

The use of openlog() is optional. It will automatically be called by syslog() if necessary, in which case *`ident`* will default to **FALSE**.

Parameters

`ident`
The string *`ident`* is added to each message.

`option`
The *`option`* argument is used to indicate what logging options will be used when generating a log message.

openlog() **Options**

Constant	Description
LOG_CONS	if there is an error while sending data to the system logger, write directly to the system console
LOG_NDELAY	open the connection to the logger immediately
LOG_ODELAY	(default) delay opening the connection until the first message is logged
LOG_PERROR	print log message also to standard error
LOG_PID	include PID with each message

You can use one or more of this options. When using multiple options you need to *OR* them, i.e. to open the connection immediately, write to the console and include the PID in each message, you will use: *LOG_CONS* | *LOG_NDELAY* | *LOG_PID*

`facility`
The *`facility`* argument is used to specify what type of program is logging the message. This allows you to specify (in your machine's syslog configuration) how messages coming from different facilities will be handled.

openlog() **Facilities**

Constant	Description
LOG_AUTH	security/authorization messages (use **LOG_AUTHPRIV** instead in systems where that constant is defined)
LOG_AUTHPRIV	security/authorization messages (private)
LOG_CRON	clock daemon (cron and at)
LOG_DAEMON	other system daemons
LOG_KERN	kernel messages
LOG_LOCAL0... **LOG_LOCAL7**	reserved for local use, these are not available in Windows
LOG_LPR	line printer subsystem
LOG_MAIL	mail subsystem
LOG_NEWS	USENET news subsystem
LOG_SYSLOG	messages generated internally by syslogd
LOG_USER	generic user-level messages
LOG_UUCP	UUCP subsystem

Note
LOG_USER is the only valid log type under Windows operating systems

Return Values

Returns **TRUE** on success or **FALSE** on failure.

See Also

- define_syslog_variables()
- syslog()
- closelog()

pfsockopen

pfsockopen -- Open persistent Internet or Unix domain socket connection

Description

resource **pfsockopen** (string `$hostname` [, int `$port` [, int `&$errno` [, string `&$errstr` [, float `$timeout`]]]])

This function behaves exactly as fsockopen() with the difference that the connection is not closed after the script finishes. It is the persistent version of fsockopen().

Parameters

For parameter information, see the fsockopen() documentation.

See Also

- fsockopen()

setcookie

setcookie -- Send a cookie

Description

bool **setcookie** (string `$name` [, string `$value` [, int `$expire` [, string `$path` [, string `$domain` [, bool `$secure` [, bool `$httponly`]]]]]])

setcookie() defines a cookie to be sent along with the rest of the HTTP headers. Like other headers, cookies must be sent *before* any output from your script (this is a protocol restriction). This requires that you place calls to this function prior to any output, including *<html>* and *<head>* tags as well as any whitespace.

Once the cookies have been set, they can be accessed on the next page load with the $_COOKIE or *$HTTP_COOKIE_VARS* arrays. Note, superglobals such as *$_COOKIE* became available in PHP 4.1.0. Cookie values also exist in $_REQUEST.

Parameters

All the arguments except the *`name`* argument are optional. You may also replace an argument with an empty string (*''*) in order to skip that argument. Because the *`expire`* argument is integer, it cannot be skipped with an empty string, use a zero (*0*) instead.

See » Netscape cookie specification for specifics on how each setcookie() parameter works

`name`
: The name of the cookie.

`value`
: The value of the cookie. This value is stored on the clients computer; do not store sensitive information. Assuming the *`name`* is 'cookiename', this value is retrieved through *$_COOKIE['cookiename']*

`expire`
: The time the cookie expires. This is a Unix timestamp so is in number of seconds since the epoch. In other words, you'll most likely set this with the time() function plus the number of seconds before you want it to expire. Or you might use mktime(). *time()+60*60*24*30* will set the cookie to expire in 30 days. If set to 0, or omitted, the cookie will expire at the end of the session (when the browser closes).

Note
You may notice the *`expire`* parameter takes on a Unix timestamp, as opposed to the date format *Wdy, DD-Mon-YYYY HH:MM:SS GMT*, this is because PHP does this conversion internally. *`expire`* is compared to the client's time which can differ from server's time.

path
: The path on the server in which the cookie will be available on. If set to *'/'*, the cookie will be available within the entire *domain*. If set to *'/foo/'*, the cookie will only be available within the */foo/* directory and all sub-directories such as */foo/bar/* of *domain*. The default value is the current directory that the cookie is being set in.

domain
: The domain that the cookie is available. To make the cookie available on all subdomains of example.com then you'd set it to *'.example.com'*. The. is not required but makes it compatible with more browsers. Setting it to *www.example.com* will make the cookie only available in the *www* subdomain. Refer to tail matching in the » spec for details.

secure
: Indicates that the cookie should only be transmitted over a secure HTTPS connection from the client. When set to **TRUE**, the cookie will only be set if a secure connection exists. The default is **FALSE**. On the server-side, it's on the programmer to send this kind of cookie only on secure connection (e.g. with respect to *$_SERVER["HTTPS"]*).

httponly
: When **TRUE** the cookie will be made accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectly help to reduce identity theft through XSS attacks (although it is not supported by all browsers). Added in PHP 5.2.0. **TRUE** or **FALSE**

Return Values

If output exists prior to calling this function, setcookie() will fail and return **FALSE**. If setcookie() successfully runs, it will return **TRUE**. This does not indicate whether the user accepted the cookie.

Examples

Some examples follow how to send cookies:

Example #19 - setcookie() send example

```
<?php
$value = 'something from somewhere';

setcookie("TestCookie", $value);
setcookie("TestCookie", $value, time()+3600);  /* expire in 1 hour */
setcookie("TestCookie", $value, time()+3600, "/~rasmus/", ".example.com", 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. If you don't want this, you can use setrawcookie() instead if you are using PHP 5. To see the contents of our test cookie in a script, simply use one of the following examples:

```php
<?php
// Print an individual cookie
echo $_COOKIE["TestCookie"];
echo $HTTP_COOKIE_VARS["TestCookie"];

// Another way to debug/test is to view all cookies
print_r($_COOKIE);
?>
```

Example #20 - setcookie() delete example

When deleting a cookie you should assure that the expiration date is in the past, to trigger the removal mechanism in your browser. Examples follow how to delete cookies sent in previous example:

```php
<?php
// set the expiration date to one hour ago
setcookie ("TestCookie", "", time() - 3600);
setcookie ("TestCookie", "", time() - 3600, "/~rasmus/", ".example.com", 1);
?>
```

Example #21 - setcookie() and arrays

You may also set array cookies by using array notation in the cookie name. This has the effect of setting as many cookies as you have array elements, but when the cookie is received by your script, the values are all placed in an array with the cookie's name:

```php
<?php
// set the cookies
setcookie("cookie[three]", "cookiethree");
setcookie("cookie[two]", "cookietwo");
setcookie("cookie[one]", "cookieone");

// after the page reloads, print them out
if (isset($_COOKIE['cookie'])) {
   foreach ($_COOKIE['cookie'] as $name => $value) {
       echo "$name : $value <br />\n";
   }
}
?>
```

The above example will output:

```
three : cookiethree
two : cookietwo
one : cookieone
```

Notes

Note
As of PHP 4, you can use output buffering to send output prior to the call of this function, with the overhead of all of your output to the browser being buffered in the server until you send it. You can do this by calling ob_start() and ob_end_flush() in your script, or setting the *output_buffering* configuration directive on in your *php.ini* or server configuration files.

Note
If the PHP directive register_globals is set to *on* then cookie values will also be made into variables. In our examples below, *$TestCookie* will exist. It's recommended to use *$_COOKIE*.

Common Pitfalls:

- Cookies will not become visible until the next loading of a page that the cookie should be visible for. To test if a cookie was successfully set, check for the cookie on a next loading page before the cookie expires. Expire time is set via the `expire` parameter. A nice way to debug the existence of cookies is by simply calling *print_r($_COOKIE);*.
- Cookies must be deleted with the same parameters as they were set with. If the value argument is an empty string, or **FALSE**, and all other arguments match a previous call to setcookie, then the cookie with the specified name will be deleted from the remote client.
- Because setting a cookie with a value of **FALSE** will try to delete the cookie, you should not use boolean values. Instead, use *0* for **FALSE** and *1* for **TRUE**.
- Cookies names can be set as array names and will be available to your PHP scripts as arrays but separate cookies are stored on the users system. Consider explode() to set one cookie with multiple names and values. It is not recommended to use serialize() for this purpose, because it can result in security holes.

Multiple calls to setcookie() are performed in the order called.

See Also

- header()
- setrawcookie()
- cookies section
- » RFC 2109
- » RFC 2965

setrawcookie

setrawcookie -- Send a cookie without urlencoding the cookie value

Description

bool **setrawcookie** (string `$name` [, string `$value` [, int `$expire` [, string `$path` [, string `$domain` [, bool `$secure` [, bool `$httponly`]]]]]])

setrawcookie() is exactly the same as setcookie() except that the cookie value will not be automatically urlencoded when sent to the browser.

Parameters

For parameter information, see the setcookie() documentation.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

ChangeLog

Version	Description
5.2.0	The *`httponly`* parameter was added.

See Also

- setcookie()

socket_get_status

socket_get_status -- Alias of stream_get_meta_data()

Description

This function is an alias of: stream_get_meta_data().

socket_set_blocking

socket_set_blocking -- Alias of stream_set_blocking()

Description

This function is an alias of: stream_set_blocking().

socket_set_timeout

socket_set_timeout -- Alias of stream_set_timeout()

Description

This function is an alias of: stream_set_timeout().

syslog

syslog -- Generate a system log message

Description

bool **syslog** (int `$priority`, string `$message`)

syslog() generates a log message that will be distributed by the system logger.

For information on setting up a user defined log handler, see the **syslog.conf**) 5) Unix manual page. More information on the syslog facilities and option can be found in the man pages for **syslog**) 3) on Unix machines.

Parameters

`priority`
: *`priority`* is a combination of the facility and the level. Possible values are:

  syslog() **Priorities (in descending order)**

Constant	Description
LOG_EMERG	system is unusable
LOG_ALERT	action must be taken immediately
LOG_CRIT	critical conditions
LOG_ERR	error conditions
LOG_WARNING	warning conditions
LOG_NOTICE	normal, but significant, condition
LOG_INFO	informational message
LOG_DEBUG	debug-level message

`message`
: 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.

Return Values

Returns **TRUE** on success or **FALSE** on failure.

Examples

Example #22 - Using syslog()

```
<?php
define_syslog_variables();
// open syslog, include the process ID and also send
// the log to standard error, and use a user defined
// logging mechanism
openlog("myScriptLog", LOG_PID | LOG_PERROR, LOG_LOCAL0);

// some code

if (authorized_client()) {
   // do something
} else {
   // unauthorized client!
   // log the attempt
   $access = date("Y/m/d H:i:s");
   syslog(LOG_WARNING, "Unauthorized client: $access {$_SERVER['REMOTE_ADDR']}
({$_SERVER['HTTP_USER_AGENT']})");
}

closelog();
?>
```

Notes

On Windows NT, the syslog service is emulated using the Event Log.

Note

Use of *LOG_LOCAL0* through *LOG_LOCAL7* for the `facility` parameter of openlog() is not available in Windows.

See Also

- define_syslog_variables()
- openlog()
- closelog()