Sockets

Introduction

The socket extension implements a low-level interface to the socket communication functions based on the popular BSD sockets, providing the possibility to act as a socket server as well as a client.

For a more generic client-side socket interface, see stream_socket_client(), stream_socket_server(), fsockopen(), and pfsockopen().

When using these functions, it is important to remember that while many of them have identical names to their C counterparts, they often have different declarations. Please be sure to read the descriptions to avoid confusion.

Those unfamiliar with socket programming can find a lot of useful material in the appropriate Unix man pages, and there is a great deal of tutorial information on socket programming in C on the web, much of which can be applied, with slight modifications, to socket programming in PHP. The » Unix Socket FAQ might be a good start.

Note
This extension has been moved to the » PECL repository and is no longer bundled with PHP as of PHP 5.3.0.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Installation

The socket functions described here are part of an extension to PHP which must be enabled at compile time by giving the *--enable-sockets* option to *configure*.

Note
IPv6 Support was added in PHP 5.0.0.

Runtime Configuration

This extension has no configuration directives defined in *php.ini*.

Resource Types

socket_accept(), socket_create_listen() and socket_create() return socket resources.

Predefined Constants

The constants below are defined by this extension, and will only be available when the extension has either been compiled into PHP or dynamically loaded at runtime.

AF_UNIX (integer)

AF_INET (integer)

AF_INET6 (integer)

SOCK_STREAM (integer)

SOCK_DGRAM (integer)

SOCK_RAW (integer)

SOCK_SEQPACKET (integer)

SOCK_RDM (integer)

MSG_OOB (integer)

MSG_WAITALL (integer)

MSG_PEEK (integer)

MSG_DONTROUTE (integer)

MSG_EOR (integer)

MSG_EOF (integer)

SO_DEBUG (integer)

SO_REUSEADDR (integer)

SO_KEEPALIVE (integer)

SO_DONTROUTE (integer)

SO_LINGER (integer)

SO_BROADCAST (integer)

SO_OOBINLINE (integer)

SO_SNDBUF (integer)

SO_RCVBUF (integer)

SO_SNDLOWAT (integer)

SO_RCVLOWAT (integer)

SO_SNDTIMEO (integer)

SO_RCVTIMEO (integer)

SO_TYPE (integer)

SO_ERROR (integer)

SOL_SOCKET (integer)

PHP_NORMAL_READ (integer)

PHP_BINARY_READ (integer)

SOL_TCP (integer)

SOL_UDP (integer)

Examples

Example #1 - Socket example: Simple TCP/IP server

This example shows a simple talkback server. Change the *address* and *port* variables to suit your setup and execute. You may then connect to the server with a command similar to: *telnet 192.168.1.53 10000* (where the address and port match your setup). Anything you type will then be output on the server side, and echoed back to you. To disconnect, enter 'quit'.

```
#!/usr/local/bin/php -q
<?php
error_reporting(E_ALL);

/* Allow the script to hang around waiting for connections. */
set_time_limit(0);

/* Turn on implicit output flushing so we see what we're getting
* as it comes in. */
ob_implicit_flush();

$address = '192.168.1.53';
$port = 10000;

if (($sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP)) === false) {
   echo "socket_create() failed: reason: " .
socket_strerror(socket_last_error()) . "\n";
}

if (socket_bind($sock, $address, $port) === false) {
   echo "socket_bind() failed: reason: " .
socket_strerror(socket_last_error($sock)) . "\n";
}

if (socket_listen($sock, 5) === false) {
   echo "socket_listen() failed: reason: " .
socket_strerror(socket_last_error($sock)) . "\n";
}

do {
   if (($msgsock = socket_accept($sock)) === false) {
       echo "socket_accept() failed: reason: " .
socket_strerror(socket_last_error($sock)) . "\n";
       break;
   }
   /* Send instructions. */
   $msg = "\nWelcome to the PHP Test Server. \n" .
       "To quit, type 'quit'. To shut down the server type 'shutdown'.\n";
   socket_write($msgsock, $msg, strlen($msg));

   do {
       if (false === ($buf = socket_read($msgsock, 2048, PHP_NORMAL_READ)))
{
           echo "socket_read() failed: reason: " .
```

```
socket_strerror(socket_last_error($msgsock)) . "\n";
            break 2;
        }
        if (!$buf = trim($buf)) {
            continue;
        }
        if ($buf == 'quit') {
            break;
        }
        if ($buf == 'shutdown') {
            socket_close($msgsock);
            break 2;
        }
        $talkback = "PHP: You said '$buf'.\n";
        socket_write($msgsock, $talkback, strlen($talkback));
        echo "$buf\n";
    } while (true);
    socket_close($msgsock);
} while (true);

socket_close($sock);
?>
```

Example #2 - Socket example: Simple TCP/IP client

This example shows a simple, one-shot HTTP client. It simply connects to a page, submits a HEAD request, echoes the reply, and exits.

```
<?php
error_reporting(E_ALL);

echo "<h2>TCP/IP Connection</h2>\n";

/* Get the port for the WWW service. */
$service_port = getservbyname('www', 'tcp');

/* Get the IP address for the target host. */
$address = gethostbyname('www.example.com');

/* Create a TCP/IP socket. */
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);
if ($socket === false) {
    echo "socket_create() failed: reason: " .
socket_strerror(socket_last_error()) . "\n";
} else {
    echo "OK.\n";
}

echo "Attempting to connect to '$address' on port '$service_port'...";
$result = socket_connect($socket, $address, $service_port);
if ($result === false) {
    echo "socket_connect() failed.\nReason: ($result) " .
socket_strerror(socket_last_error($socket)) . "\n";
} else {
    echo "OK.\n";
```

```
}

$in = "HEAD / HTTP/1.1\r\n";
$in .= "Host: www.example.com\r\n";
$in .= "Connection: Close\r\n\r\n";
$out = '';

echo "Sending HTTP HEAD request...";
socket_write($socket, $in, strlen($in));
echo "OK.\n";

echo "Reading response:\n\n";
while ($out = socket_read($socket, 2048)) {
   echo $out;
}

echo "Closing socket...";
socket_close($socket);
echo "OK.\n\n";
?>
```

Socket Errors

The socket extension was written to provide a usable interface to the powerful BSD sockets. Care has been taken that the functions work equally well on Win32 and Unix implementations. Almost all of the sockets functions may fail under certain conditions and therefore emit an **E_WARNING** message describing the error. Sometimes this doesn't happen to the desire of the developer. For example the function socket_read() may suddenly emit an **E_WARNING** message because the connection broke unexpectedly. It's common to suppress the warning with the @ -operator and catch the error code within the application with the socket_last_error() function. You may call the socket_strerror() function with this error code to retrieve a string describing the error. See their description for more information.

Note

The **E_WARNING** messages generated by the socket extension are in English though the retrieved error message will appear depending on the current locale (**LC_MESSAGES**):

```
Warning - socket_bind() unable to bind address [98]: Die Adresse wird
bereits verwendet
```

Socket Functions

socket_accept

socket_accept -- Accepts a connection on a socket

Description

resource **socket_accept** (resource `$socket`)

After the socket *`socket`* has been created using socket_create(), bound to a name with socket_bind(), and told to listen for connections with socket_listen(), this function will accept incoming connections on that socket. Once a successful connection is made, a new socket resource is returned, which may be used for communication. If there are multiple connections queued on the socket, the first will be used. If there are no pending connections, socket_accept() will block until a connection becomes present. If *`socket`* has been made non-blocking using socket_set_blocking() or socket_set_nonblock(), **FALSE** will be returned.

The socket resource returned by socket_accept() may not be used to accept new connections. The original listening socket *`socket`*, however, remains open and may be reused.

Parameters

`socket`
A valid socket resource created with socket_create().

Return Values

Returns a new socket resource on success, or **FALSE** on error. The actual error code can be retrieved by calling socket_last_error(). This error code may be passed to socket_strerror() to get a textual explanation of the error.

See Also

- socket_connect()
- socket_listen()
- socket_create()
- socket_bind()
- socket_strerror()

socket_bind

socket_bind -- Binds a name to a socket

Description

bool **socket_bind** (resource `$socket`, string `$address` [, int `$port`])

Binds the name given in *`address`* to the socket described by *`socket`*. This has to be done before a connection is be established using socket_connect() or socket_listen().

Parameters

`socket`
A valid socket resource created with socket_create().

`address`
If the socket is of the **AF_INET** family, the *`address`* is an IP in dotted-quad notation (e.g. *127.0.0.1*). If the socket is of the **AF_UNIX** family, the *`address`* is the path of a Unix-domain socket (e.g. */tmp/my.sock*).

`port` (Optional)
The *`port`* parameter is only used when connecting to an **AF_INET** socket, and designates the port on the remote host to which a connection should be made.

Return Values

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

The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual explanation of the error.

Examples

Example #3 - Using socket_bind() to set the source address

```
<?php
// Create a new socket
$sock = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);

// An example list of IP addresses owned by the computer
$sourceips['kevin']    = '127.0.0.1';
$sourceips['madcoder'] = '127.0.0.2';

// Bind the source address
socket_bind($sock, $sourceips['madcoder']);
```

```
// Connect to destination address
socket_connect($sock, '127.0.0.1', 80);

// Write
$request = 'GET / HTTP/1.1' . "\r\n" .
           'Host: example.com' . "\r\n\r\n";
socket_write($sock, $request);

// Close
socket_close($sock);

?>
```

Notes

Note
This function must be used on the socket before socket_connect().

Note
Windows 9x/ME compatibility note: socket_last_error() may return an invalid error code if trying to bind the socket to a wrong address that does not belong to your machine.

See Also

- socket_connect()
- socket_listen()
- socket_create()
- socket_last_error()
- socket_strerror()

socket_clear_error

socket_clear_error -- Clears the error on the socket or the last error code

Description

void **socket_clear_error** ([resource `$socket`])

This function clears the error code on the given socket or the global last socket error if no socket is specified.

This function allows explicitly resetting the error code value either of a socket or of the extension global last error code. This may be useful to detect within a part of the application if an error occurred or not.

Parameters

`socket`
A valid socket resource created with socket_create().

Return Values

No value is returned.

See Also

- socket_last_error()
- socket_strerror()

socket_close

socket_close -- Closes a socket resource

Description

void **socket_close** (resource `$socket`)

socket_close() closes the socket resource given by *`socket`*. This function is specific to sockets and cannot be used on any other type of resources.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

Return Values

No value is returned.

See Also

- socket_bind()
- socket_listen()
- socket_create()
- socket_strerror()

socket_connect

socket_connect -- Initiates a connection on a socket

Description

bool **socket_connect** (resource `$socket`, string `$address` [, int `$port`])

Initiate a connection to *`address`* using the socket resource *`socket`*, which must be a valid socket resource created with socket_create().

Parameters

`socket`

`address`
The *`address`* parameter is either an IPv4 address in dotted-quad notation (e.g. *127.0.0.1*) if *`socket`* is **AF_INET**, a valid IPv6 address (e.g.*::1*) if IPv6 support is enabled and *`socket`* is **AF_INET6** or the pathname of a Unix domain socket, if the socket family is **AF_UNIX**.

`port`
The *`port`* parameter is only used and is mandatory when connecting to an **AF_INET** or an **AF_INET6** socket, and designates the port on the remote host to which a connection should be made.

Return Values

Returns **TRUE** on success or **FALSE** on failure. The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual explanation of the error.

Note
If the socket is non-blocking then this function returns **FALSE** with an error *Operation now in progress*.

See Also

- socket_bind()
- socket_listen()

- socket_create()
- socket_last_error()
- socket_strerror()

socket_create_listen

socket_create_listen -- Opens a socket on port to accept connections

Description

resource **socket_create_listen** (int `$port` [, int `$backlog`])

socket_create_listen() creates a new socket resource of type **AF_INET** listening on *all* local interfaces on the given port waiting for new connections.

This function is meant to ease the task of creating a new socket which only listens to accept new connections.

Parameters

`port`
: The port on which to listen on all interfaces.

`backlog`
: The *`backlog`* parameter defines the maximum length the queue of pending connections may grow to. **SOMAXCONN** may be passed as *`backlog`* parameter, see socket_listen() for more information.

Return Values

socket_create_listen() returns a new socket resource on success or **FALSE** on error. The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual explanation of the error.

Notes

Note
If you want to create a socket which only listens on a certain interface you need to use socket_create(), socket_bind() and socket_listen().

See Also

- socket_create()
- socket_create_pair()
- socket_bind()

- socket_listen()
- socket_last_error()
- socket_strerror()

socket_create_pair

socket_create_pair -- Creates a pair of indistinguishable sockets and stores them in an array

Description

bool **socket_create_pair** (int `$domain`, int `$type`, int `$protocol`, array `&$fd`)

socket_create_pair() creates two connected and indistinguishable sockets, and stores them in *`fd`*. This function is commonly used in IPC (InterProcess Communication).

Parameters

`domain`
: The *`domain`* parameter specifies the protocol family to be used by the socket. See socket_create() for the full list.

`type`
: The *`type`* parameter selects the type of communication to be used by the socket. See socket_create() for the full list.

`protocol`
: The *`protocol`* parameter sets the specific protocol within the specified *`domain`* to be used when communicating on the returned socket. The proper value can be retrieved by name by using getprotobyname(). If the desired protocol is TCP, or UDP the corresponding constants **SOL_TCP**, and **SOL_UDP** can also be used. See socket_create() for the full list of supported protocols.

`fd`
: Reference to an array in which the two socket resources will be inserted.

Return Values

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

Examples

Example #4 - socket_create_pair() example

```
<?php
$sockets = array();
/* Setup socket pair */
if (socket_create_pair(AF_UNIX, SOCK_STREAM, 0, $sockets) === false) {
   echo "socket_create_pair failed. Reason:
```

```
".socket_strerror(socket_last_error());
}
/* Send and Recieve Data */
if (socket_write($sockets[0], "ABCdef123\n", strlen("ABCdef123\n")) ===
false) {
   echo "socket_write() failed. Reason:
".socket_strerror(socket_last_error($sockets[0]));
}
if (($data = socket_read($sockets[1], strlen("ABCdef123\n"),
PHP_BINARY_READ) === false) {
   echo "socket_read() failed. Reason:
".socket_strerror(socket_last_error($sockets[1]));
}
var_dump($data);

/* Close sockets */
socket_close($sockets[0]);
socket_close($sockets[1]);
?>
```

Example #5 - socket_create_pair() **IPC example**

```
<?php
$ary = array();
$strone = 'Message From Parent.';
$strtwo = 'Message From Child.';
if (socket_create_pair(AF_UNIX, SOCK_STREAM, 0, $ary) === false) {
   echo "socket_create_pair() failed. Reason:
".socket_strerror(socket_last_error());
}
$pid = pcntl_fork();
if ($pid == -1) {
   echo 'Could not fork Process.';
} elseif ($pid) {
   /*parent*/
   socket_close($ary[0]);
   if (socket_write($ary[1], $strone, strlen($strone)) === false) {
       echo "socket_write() failed. Reason:
".socket_strerror(socket_last_error($ary[1]));
   }
   if (socket_read($ary[1], strlen($strtwo), PHP_BINARY_READ) == $strtwo) {
       echo "Recieved $strtwo\n";
   }
   socket_close($ary[1]);
} else {
   /*child*/
   socket_close($ary[1]);
   if (socket_write($ary[0], $strtwo, strlen($strtwo)) === false) {
       echo "socket_write() failed. Reason:
".socket_strerror(socket_last_error($ary[0]));
   }
   if (socket_read($ary[0], strlen($strone), PHP_BINARY_READ) == $strone) {
       echo "Recieved $strone\n";
   }
   socket_close($ary[0]);
```

```
}
?>
```

See Also

- socket_create()
- socket_create_listen()
- socket_bind()
- socket_listen()
- socket_last_error()
- socket_strerror()

socket_create

socket_create -- Create a socket (endpoint for communication)

Description

resource **socket_create** (int `$domain`, int `$type`, int `$protocol`)

Creates and returns a socket resource, also referred to as an endpoint of communication. A typical network connection is made up of 2 sockets, one performing the role of the client, and another performing the role of the server.

Parameters

`domain`

The *`domain`* parameter specifies the protocol family to be used by the socket.

Available address/protocol families

Domain	Description
AF_INET	IPv4 Internet based protocols. TCP and UDP are common protocols of this protocol family.
AF_INET6	IPv6 Internet based protocols. TCP and UDP are common protocols of this protocol family. Support added in PHP 5.0.0.
AF_UNIX	Local communication protocol family. High efficiency and low overhead make it a great form of IPC (Interprocess Communication).

`type`

The *`type`* parameter selects the type of communication to be used by the socket.

Available socket types

Type	Description
SOCK_STREAM	Provides sequenced, reliable, full-duplex, connection-based byte streams. An out-of-band data transmission mechanism may be supported. The TCP protocol is based on this socket type.

SOCK_DGRAM	Supports datagrams (connectionless, unreliable messages of a fixed maximum length). The UDP protocol is based on this socket type.
SOCK_SEQPACKET	Provides a sequenced, reliable, two-way connection-based data transmission path for datagrams of fixed maximum length; a consumer is required to read an entire packet with each read call.
SOCK_RAW	Provides raw network protocol access. This special type of socket can be used to manually construct any type of protocol. A common use for this socket type is to perform ICMP requests (like ping, traceroute, etc).
SOCK_RDM	Provides a reliable datagram layer that does not guarantee ordering. This is most likely not implemented on your operating system.

protocol

The *protocol* parameter sets the specific protocol within the specified *domain* to be used when communicating on the returned socket. The proper value can be retrieved by name by using getprotobyname(). If the desired protocol is TCP, or UDP the corresponding constants **SOL_TCP**, and **SOL_UDP** can also be used.

Common protocols

Name	**Description**
icmp	The Internet Control Message Protocol is used primarily by gateways and hosts to report errors in datagram communication. The "ping" command (present in most modern operating systems) is an example application of ICMP.
udp	The User Datagram Protocol is a connectionless, unreliable, protocol with fixed record lengths. Due to these aspects, UDP requires a minimum amount of protocol overhead.
tcp	The Transmission Control Protocol is a reliable, connection based, stream oriented, full duplex protocol. TCP guarantees that all data packets will be received in the order in which they were sent. If any packet is

	somehow lost during communication, TCP will automatically retransmit the packet until the destination host acknowledges that packet. For reliability and performance reasons, the TCP implementation itself decides the appropriate octet boundaries of the underlying datagram communication layer. Therefore, TCP applications must allow for the possibility of partial record transmission.

Return Values

socket_create() returns a socket resource on success, or **FALSE** on error. The actual error code can be retrieved by calling socket_last_error(). This error code may be passed to socket_strerror() to get a textual explanation of the error.

Errors/Exceptions

If an invalid *domain* or *type* is given, socket_create() defaults to **AF_INET** and **SOCK_STREAM** respectively and additionally emits an **E_WARNING** message.

See Also

- socket_accept()
- socket_bind()
- socket_connect()
- socket_listen()
- socket_last_error()
- socket_strerror()

socket_get_option

socket_get_option -- Gets socket options for the socket

Description

mixed **socket_get_option** (resource `$socket`, int `$level`, int `$optname`)

The socket_get_option() function retrieves the value for the option specified by the *`optname`* parameter for the specified *`socket`*.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`level`
The *`level`* parameter specifies the protocol level at which the option resides. For example, to retrieve options at the socket level, a *`level`* parameter of SOL_SOCKET would be used. Other levels, such as TCP, can be used by specifying the protocol number of that level. Protocol numbers can be found by using the getprotobyname() function.

`optname`

Available Socket Options

Option	Description	Type
SO_DEBUG	Reports whether debugging information is being recorded.	int
SO_BROADCAST	Reports whether transmission of broadcast messages is supported.	int
SO_REUSEADDR	Reports whether local addresses can be reused.	int
SO_KEEPALIVE	Reports whether connections are kept active with periodic transmission of messages. If the connected socket fails to respond to these messages, the connection is broken and processes writing to that socket are notified with a	int

	SIGPIPE signal.	
SO_LINGER	Reports whether the *socket* lingers on socket_close() if data is present. By default, when the socket is closed, it attempts to send all unsent data. In the case of a connection-oriented socket, socket_close() will wait for its peer to acknowledge the data. If *l_onoff* is non-zero and *l_linger* is zero, all the unsent data will be discarded and RST (reset) is sent to the peer in the case of a connection-oriented socket. On the other hand, if *l_onoff* is non-zero and *l_linger* is non-zero, socket_close() will block until all the data is sent or the time specified in *l_linger* elapses. If the socket is non-blocking, socket_close() will fail and return an error.	array. The array will contain two keys: *l_onoff* and *l_linger* .
SO_OOBINLINE	Reports whether the *socket* leaves out-of-band data inline.	int
SO_SNDBUF	Reports the size of the send buffer.	int
SO_RCVBUF	Reports the size of the receive buffer.	int
SO_ERROR	Reports information about error status and clears it.	int (cannot be set by socket_set_option())
SO_TYPE	Reports the *socket* type (e.g. **SOCK_STREAM**).	int (cannot be set by socket_set_option())
SO_DONTROUTE	Reports whether outgoing messages bypass the standard routing facilities.	int

SO_RCVLOWAT	Reports the minimum number of bytes to process for *socket* input operations.	int
SO_RCVTIMEO	Reports the timeout value for input operations.	array. The array will contain two keys: *sec* which is the seconds part on the timeout value and *usec* which is the microsecond part of the timeout value.
SO_SNDTIMEO	Reports the timeout value specifying the amount of time that an output function blocks because flow control prevents data from being sent.	array. The array will contain two keys: *sec* which is the seconds part on the timeout value and *usec* which is the microsecond part of the timeout value.
SO_SNDLOWAT	Reports the minimum number of bytes to process for *socket* output operations.	int

Return Values

Returns the value of the given option, or **FALSE** on errors.

Examples

Example #6 - socket_set_option() example

```
<?php
$socket = socket_create_listen(1223);

$linger = array('l_linger' => 1, 'l_onoff' => 1);
socket_set_option($socket, SOL_SOCKET, SO_LINGER, $linger);

var_dump(socket_get_option($socket, SOL_SOCKET, SO_REUSEADDR));
?>
```

ChangeLog

Version	Description

4.3.0	The name of this function was changed. It used to be called *socket_getopt()*.

socket_getpeername

socket_getpeername -- Queries the remote side of the given socket which may either result in host/port or in a Unix filesystem path, dependent on its type

Description

bool **socket_getpeername** (resource `$socket`, string `&$address` [, int `&$port`])

Queries the remote side of the given socket which may either result in host/port or in a Unix filesystem path, dependent on its type.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`address`
If the given socket is of type **AF_INET** or **AF_INET6**, socket_getpeername() will return the peers (remote) *IP address* in appropriate notation (e.g. *127.0.0.1* or *fe80::1*) in the *`address`* parameter and, if the optional *`port`* parameter is present, also the associated port. If the given socket is of type **AF_UNIX**, socket_getpeername() will return the Unix filesystem path (e.g. */var/run/daemon.sock*) in the *`address`* parameter.

`port`
If given, this will hold the port associated to *`address`*.

Return Values

Returns **TRUE** on success or **FALSE** on failure. socket_getpeername() may also return **FALSE** if the socket type is not any of **AF_INET**, **AF_INET6**, or **AF_UNIX**, in which case the last socket error code is *not* updated.

Notes

Note
socket_getpeername() should not be used with **AF_UNIX** sockets created with socket_accept(). Only sockets created with socket_connect() or a primary server socket following a call to socket_bind() will return meaningful values.

See Also

- socket_getsockname()
- socket_last_error()
- socket_strerror()

socket_getsockname

socket_getsockname -- Queries the local side of the given socket which may either result in host/port or in a Unix filesystem path, dependent on its type

Description

bool **socket_getsockname** (resource `$socket`, string `&$addr` [, int `&$port`])

Note
socket_getsockname() should not be used with **AF_UNIX** sockets created with socket_connect(). Only sockets created with socket_accept() or a primary server socket following a call to socket_bind() will return meaningful values.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`addr`
If the given socket is of type **AF_INET** or **AF_INET6**, socket_getsockname() will return the local *IP address* in appropriate notation (e.g. *127.0.0.1* or *fe80::1*) in the *`address`* parameter and, if the optional *`port`* parameter is present, also the associated port. If the given socket is of type **AF_UNIX**, socket_getsockname() will return the Unix filesystem path (e.g. */var/run/daemon.sock*) in the *`address`* parameter.

`port`
If provided, this will hold the associated port.

Return Values

Returns **TRUE** on success or **FALSE** on failure. socket_getsockname() may also return **FALSE** if the socket type is not any of **AF_INET**, **AF_INET6**, or **AF_UNIX**, in which case the last socket error code is *not* updated.

See Also

- socket_getpeername()
- socket_last_error()
- socket_strerror()

socket_last_error

socket_last_error -- Returns the last error on the socket

Description

int **socket_last_error** ([resource `$socket`])

If a socket resource is passed to this function, the last error which occurred on this particular socket is returned. If the socket resource is omitted, the error code of the last failed socket function is returned. The latter is particularly helpful for functions like socket_create() which don't return a socket on failure and socket_select() which can fail for reasons not directly tied to a particular socket. The error code is suitable to be fed to socket_strerror() which returns a string describing the given error code.

Parameters

`socket`
A valid socket resource created with socket_create().

Return Values

This function returns a socket error code.

Examples

Example #7 - socket_last_error() example

```
<?php
$socket = @socket_create(AF_INET, SOCK_STREAM, SOL_TCP);

if ($socket === false) {
   $errorcode = socket_last_error();
   $errormsg = socket_strerror($errorcode);

   die("Couldn't create socket: [$errorcode] $errormsg");
}
?>
```

Notes

Note

socket_last_error() does not clear the error code, use socket_clear_error() for this purpose.

socket_listen

socket_listen -- Listens for a connection on a socket

Description

bool **socket_listen** (resource `$socket` [, int `$backlog`])

After the socket *`socket`* has been created using socket_create() and bound to a name with socket_bind(), it may be told to listen for incoming connections on *`socket`*.

socket_listen() is applicable only to sockets of type **SOCK_STREAM** or **SOCK_SEQPACKET**.

Parameters

`socket`
A valid socket resource created with socket_create().

`backlog`
A maximum of *`backlog`* incoming connections will be queued for processing. If a connection request arrives with the queue full the client may receive an error with an indication of *ECONNREFUSED*, or, if the underlying protocol supports retransmission, the request may be ignored so that retries may succeed.

Note
The maximum number passed to the *`backlog`* parameter highly depends on the underlying platform. On Linux, it is silently truncated to **SOMAXCONN**. On win32, if passed **SOMAXCONN**, the underlying service provider responsible for the socket will set the backlog to a maximum *reasonable* value. There is no standard provision to find out the actual backlog value on this platform.

Return Values

Returns **TRUE** on success or **FALSE** on failure. The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual explanation of the error.

See Also

- socket_accept()

- socket_bind()
- socket_connect()
- socket_create()
- socket_strerror()

socket_read

socket_read -- Reads a maximum of length bytes from a socket

Description

string **socket_read** (resource `$socket`, int `$length` [, int `$type`])

The function socket_read() reads from the socket resource *`socket`* created by the socket_create() or socket_accept() functions.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`length`
The maximum number of bytes read is specified by the *`length`* parameter. Otherwise you can use \r, \n, or \0 to end reading (depending on the *`type`* parameter, see below).

`type`
Optional *`type`* parameter is a named constant:

- **PHP_BINARY_READ** (Default) - use the system *recv()* function. Safe for reading binary data.
- **PHP_NORMAL_READ** - reading stops at \n or \r.

Return Values

socket_read() returns the data as a string on success, or **FALSE** on error (including if the remote host has closed the connection). The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual representation of the error.

Note
socket_read() returns a zero length string ("") when there is no more data to read.

ChangeLog

Version	Description
4.1.0	The default value for *type* was changed from **PHP_NORMAL_READ** to **PHP_BINARY_READ**

See Also

- socket_accept()
- socket_bind()
- socket_connect()
- socket_listen()
- socket_last_error()
- socket_strerror()
- socket_write()

socket_recv

socket_recv -- Receives data from a connected socket

Description

int **socket_recv** (resource `$socket`, string `&$buf`, int `$len`, int `$flags`)

Warning
This function is currently not documented; only its argument list is available.

socket_recvfrom

socket_recvfrom -- Receives data from a socket whether or not it is connection-oriented

Description

int **socket_recvfrom** (resource `$socket`, string `&$buf`, int `$len`, int `$flags`, string `&$name` [, int `&$port`])

The socket_recvfrom() function receives *`len`* bytes of data in *`buf`* from *`name`* on port *`port`* (if the socket is not of type **AF_UNIX**) using *`socket`*. socket_recvfrom() can be used to gather data from both connected and unconnected sockets. Additionally, one or more flags can be specified to modify the behaviour of the function.

The *`name`* and *`port`* must be passed by reference. If the socket is not connection-oriented, *`name`* will be set to the internet protocol address of the remote host or the path to the UNIX socket. If the socket is connection-oriented, *`name`* is **NULL**. Additionally, the *`port`* will contain the port of the remote host in the case of an unconnected **AF_INET** or **AF_INET6** socket.

Parameters

`socket`
: The *`socket`* must be a socket resource previously created by socket_create().

`buf`
: The data received will be fetched to the variable specified with *`buf`*.

`len`
: Up to *`len`* bytes will be fetched from remote host.

`flags`
: The value of *`flags`* can be any combination of the following flags, joined with the binary OR (|) operator.

Possible values for *`flags`*

Flag	Description
MSG_OOB	Process out-of-band data.
MSG_PEEK	Receive data from the beginning of the receive queue without removing it from the queue.
MSG_WAITALL	Block until at least *`len`* are received. However, if a signal is caught or the remote host disconnects, the function may return

	less data.
MSG_DONTWAIT	With this flag set, the function returns even if it would normally have blocked.

name
: If the socket is of the type **AF_UNIX** type, *name* is the path to the file. Else, for unconnected sockets, *name* is the IP address of, the remote host, or **NULL** if the socket is connection-oriented.

port
: This argument only applies to **AF_INET** and **AF_INET6** sockets, and specifies the remote port from which the data is received. If the socket is connection-oriented, *port* will be **NULL**.

Return Values

socket_recvfrom() returns the number of bytes received, or -1 if there was an error. The actual error code can be retrieved by calling socket_last_error(). This error code may be passed to socket_strerror() to get a textual explanation of the error.

Examples

Example #8 - socket_recvfrom() example

```
<?php
error_reporting(E_ALL | E_STRICT);

$socket = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);
socket_bind($socket, '127.0.0.1', 1223);

$from = "";
$port = 0;
socket_recvfrom($socket, $buf, 12, 0, $from, $port);

echo "Received $buf from remote address $from and remote port $port" . PHP_EOL;
?>
```

This example will initiate an UDP socket on port 1223 of 127.0.0.1 and print at most 12 characters received from a remote host.

ChangeLog

Version	Description

4.3.0	socket_recvfrom() is now binary safe.

See Also

- socket_recv()
- socket_send()
- socket_sendto()
- socket_create()

socket_select

socket_select -- Runs the select() system call on the given arrays of sockets with a specified timeout

Description

int **socket_select** (array `&$read`, array `&$write`, array `&$except`, int `$tv_sec` [, int `$tv_usec`])

socket_select() accepts arrays of sockets and waits for them to change status. Those coming with BSD sockets background will recognize that those socket resource arrays are in fact the so-called file descriptor sets. Three independent arrays of socket resources are watched.

Parameters

`read`
: The sockets listed in the *`read`* array will be watched to see if characters become available for reading (more precisely, to see if a read will not block - in particular, a socket resource is also ready on end-of-file, in which case a socket_read() will return a zero length string).

`write`
: The sockets listed in the *`write`* array will be watched to see if a write will not block.

`except`
: The sockets listed in the *`except`* array will be watched for exceptions.

`tv_sec`
: The *`tv_sec`* and *`tv_usec`* together form the *timeout* parameter. The *timeout* is an upper bound on the amount of time elapsed before socket_select() return. *`tv_sec`* may be zero , causing socket_select() to return immediately. This is useful for polling. If *`tv_sec`* is **NULL** (no timeout), socket_select() can block indefinitely.

`tv_usec`

Warning
On exit, the arrays are modified to indicate which socket resource actually changed status.

You do not need to pass every array to socket_select(). You can leave it out and use an empty array or **NULL** instead. Also do not forget that those arrays are passed *by reference*

and will be modified after socket_select() returns.

Note

Due a limitation in the current Zend Engine it is not possible to pass a constant modifier like **NULL** directly as a parameter to a function which expects this parameter to be passed by reference. Instead use a temporary variable or an expression with the leftmost member being a temporary variable:

Example #9 - Using NULL with socket_select()

```
<?php
$e = NULL;
socket_select($r, $w, $e, 0);
?>
```

Return Values

On success socket_select() returns the number of socket resources contained in the modified arrays, which may be zero if the timeout expires before anything interesting happens. On error **FALSE** is returned. The error code can be retrieved with socket_last_error().

Note

Be sure to use the === operator when checking for an error. Since the socket_select() may return 0 the comparison with == would evaluate to **TRUE**:

Example #10 - Understanding socket_select() 's result

```
<?php
$e = NULL;
if (false === socket_select($r, $w, $e, 0)) {
   echo "socket_select() failed, reason: " .
       socket_strerror(socket_last_error()) . "\n";
}
?>
```

Examples

Example #11 - socket_select() example

```
<?php
```

```
/* Prepare the read array */
$read   = array($socket1, $socket2);
$write  = NULL;
$except = NULL;
$num_changed_sockets = socket_select($read, $write, $except, 0);

if ($num_changed_sockets === false) {
   /* Error handling */
} else if ($num_changed_sockets > 0) {
   /* At least at one of the sockets something interesting happened */
}
?>
```

Notes

Note

Be aware that some socket implementations need to be handled very carefully. A few basic rules:

- You should always try to use socket_select() without timeout. Your program should have nothing to do if there is no data available. Code that depends on timeouts is not usually portable and difficult to debug.
- No socket resource must be added to any set if you do not intend to check its result after the socket_select() call, and respond appropriately. After socket_select() returns, all socket resources in all arrays must be checked. Any socket resource that is available for writing must be written to, and any socket resource available for reading must be read from.
- If you read/write to a socket returns in the arrays be aware that they do not necessarily read/write the full amount of data you have requested. Be prepared to even only be able to read/write a single byte.
- It's common to most socket implementations that the only exception caught with the *except* array is out-of-bound data received on a socket.

See Also

- socket_read()
- socket_write()
- socket_last_error()
- socket_strerror()

socket_send

socket_send -- Sends data to a connected socket

Description

int **socket_send** (resource `$socket`, string `$buf`, int `$len`, int `$flags`)

The function socket_send() sends *`len`* bytes to the socket *`socket`* from *`buf`*.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`buf`
A buffer containing the data that will be sent to the remote host.

`len`
The number of bytes that will be sent to the remote host from *`buf`*.

`flags`
The value of *`flags`* can be any combination of the following flags, joined with the binary OR (|) operator.

Possible values for *`flags`*

MSG_OOB	Send OOB (out-of-band) data.
MSG_EOR	Indicate a record mark. The sent data completes the record.
MSG_EOF	Close the sender side of the socket and include an appropriate notification of this at the end of the sent data. The sent data completes the transaction.
MSG_DONTROUTE	Bypass routing, use direct interface.

Return Values

See Also

- socket_sendto()

socket_sendto

socket_sendto -- Sends a message to a socket, whether it is connected or not

Description

int **socket_sendto** (resource `$socket`, string `$buf`, int `$len`, int `$flags`, string `$addr` [, int `$port`])

The function socket_sendto() sends *`len`* bytes from *`buf`* through the socket *`socket`* to the *`port`* at the address *`addr`*.

Parameters

`socket`
A valid socket ressource created using socket_create().

`buf`
The sent data will be taken from buffer *`buf`*.

`len`
`len` bytes from *`buf`* will be sent.

`flags`
The value of *`flags`* can be any combination of the following flags, joined with the binary OR (|) operator.

Possible values for *`flags`*

MSG_OOB	Send OOB (out-of-band) data.
MSG_EOR	Indicate a record mark. The sent data completes the record.
MSG_EOF	Close the sender side of the socket and include an appropriate notification of this at the end of the sent data. The sent data completes the transaction.
MSG_DONTROUTE	Bypass routing, use direct interface.

`addr`
IP address of the remote host.

`port`
`port` is the remote port number at which the data will be sent.

Return Values

socket_sendto() returns the number of bytes sent to the remote host or -1 if an error occured.

Examples

Example #12 - socket_sendto() Example

```
<?php
   $sock = socket_create(AF_INET, SOCK_DGRAM, SOL_UDP);

   $msg = "Ping !";
   $len = strlen($msg);

   socket_sendto($sock, $msg, $len, 0, '127.0.0.1', 1223);
   socket_close($sock);
?>
```

See Also

- socket_send()

socket_set_block

socket_set_block -- Sets blocking mode on a socket resource

Description

bool **socket_set_block** (resource `$socket`)

The socket_set_block() function removes the **O_NONBLOCK** flag on the socket specified by the *`socket`* parameter.

When an operation (e.g. receive, send, connect, accept, ...) is performed on a blocking socket, the script will pause its execution until it receives a signal or it can perform the operation.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

Return Values

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

Examples

Example #13 - socket_set_block() example

```
<?php
$socket = socket_create_listen(1223);
socket_set_block($socket);

socket_accept($socket);
?>
```

This example creates a listening socket on all interfaces on port 1223 and sets the socket to **O_BLOCK** mode. socket_accept() will hang until there is a connection to accept.

See Also

- socket_set_nonblock()
- socket_set_option()

socket_set_nonblock

socket_set_nonblock -- Sets nonblocking mode for file descriptor fd

Description

bool **socket_set_nonblock** (resource `$socket`)

The socket_set_nonblock() function sets the **O_NONBLOCK** flag on the socket specified by the *`socket`* parameter.

When an operation (e.g. receive, send, connect, accept, ...) is performed on a non-blocking socket, the script not pause its execution until it receives a signal or it can perform the operation. Rather, if the operation would result in a block, the called function will fail.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

Return Values

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

Examples

Example #14 - socket_set_nonblock() example

```
<?php
$socket = socket_create_listen(1223);
socket_set_nonblock($socket);

socket_accept($socket);
?>
```

This example creates a listening socket on all interfaces on port 1223 and sets the socket to **O_NONBLOCK** mode. socket_accept() will immediately fail unless there is a pending connection exactly at this moment.

See Also

- socket_set_block()
- socket_set_option()

socket_set_option

socket_set_option -- Sets socket options for the socket

Description

bool **socket_set_option** (resource `$socket`, int `$level`, int `$optname`, mixed `$optval`)

The socket_set_option() function sets the option specified by the *`optname`* parameter, at the specified protocol *`level`*, to the value pointed to by the *`optval`* parameter for the *`socket`*.

Parameters

`socket`
A valid socket resource created with socket_create() or socket_accept().

`level`
The *`level`* parameter specifies the protocol level at which the option resides. For example, to retrieve options at the socket level, a *`level`* parameter of **SOL_SOCKET** would be used. Other levels, such as TCP, can be used by specifying the protocol number of that level. Protocol numbers can be found by using the getprotobyname() function.

`optname`
The available socket options are the same as those for the socket_get_option() function.

`optval`
The option value.

Return Values

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

Examples

Example #15 - socket_set_option() example

```
<?php
$socket = socket_create(AF_INET, SOCK_STREAM, SOL_TCP);

if (!is_resource($socket)) {
   echo 'Unable to create socket: '. socket_strerror(socket_last_error()) .
PHP_EOL;
}

if (!socket_set_option($socket, SOL_SOCKET, SO_REUSEADDR, 1)) {
   echo 'Unable to set option on socket: '. socket_strerror(socket_last_error())
```

```
. PHP_EOL;
}

if (!socket_bind($socket, '127.0.0.1', 1223)) {
   echo 'Unable to bind socket: '. socket_strerror(socket_last_error()) .
PHP_EOL;
}

$rval = socket_get_option($socket, SOL_SOCKET, SO_REUSEADDR);

if ($rval === false) {
   echo 'Unable to get socket option: '. socket_strerror(socket_last_error()) .
PHP_EOL;
} else if ($rval !== 0) {
   echo 'SO_REUSEADDR is set on socket !' . PHP_EOL;
}
?>
```

ChangeLog

Version	Description
4.3.0	This function was renamed. It used to be called **socket_setopt()**.

socket_shutdown

socket_shutdown -- Shuts down a socket for receiving, sending, or both

Description

bool **socket_shutdown** (resource `$socket` [, int `$how`])

The socket_shutdown() function allows you to stop incoming, outgoing or all data (the default) from being sent through the *`socket`*

Parameters

`socket`
A valid socket resource created with socket_create().

how
The value of *`how`* can be one of the following:

possible values for *`how`*

0	Shutdown socket reading
1	Shutdown socket writing
2	Shutdown socket reading and writing

Return Values

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

socket_strerror

socket_strerror -- Return a string describing a socket error

Description

string **socket_strerror** (int `$errno`)

socket_strerror() takes as its *errno* parameter a socket error code as returned by socket_last_error() and returns the corresponding explanatory text.

Note
Although the error messages generated by the socket extension are in English, the system messages retrieved with this function will appear depending on the current locale (**LC_MESSAGES**).

Parameters

errno
A valid socket error number, likely produced by socket_last_error().

Return Values

Returns the error message associated with the *errno* parameter.

Examples

Example #16 - socket_strerror() example

```
<?php
if (false == ($socket = @socket_create(AF_INET, SOCK_STREAM, SOL_TCP))) {
  echo "socket_create() failed: reason: " . socket_strerror(socket_last_error())
. "\n";
}

if (false == (@socket_bind($socket, '127.0.0.1', 80))) {
  echo "socket_bind() failed: reason: " .
socket_strerror(socket_last_error($socket)) . "\n";
}
?>
```

The expected output from the above example (assuming the script is not run with root privileges):

```
socket_bind() failed: reason: Permission denied
```

See Also

- socket_accept()
- socket_bind()
- socket_connect()
- socket_listen()
- socket_create()

socket_write

socket_write -- Write to a socket

Description

int **socket_write** (resource `$socket`, string `$buffer` [, int `$length`])

The function socket_write() writes to the *`socket`* from the given *`buffer`*.

Parameters

`socket`

`buffer`
The buffer to be written.

`length`
The optional parameter *`length`* can specify an alternate length of bytes written to the socket. If this length is greater then the buffer length, it is silently truncated to the length of the buffer.

Return Values

Returns the number of bytes successfully written to the socket or **FALSE** one error. The error code can be retrieved with socket_last_error(). This code may be passed to socket_strerror() to get a textual explanation of the error.

Note
It is perfectly valid for socket_write() to return zero which means no bytes have been written. Be sure to use the === operator to check for **FALSE** in case of an error.

Notes

Note
socket_write() does not necessarily write all bytes from the given buffer. It's valid that, depending on the network buffers etc., only a certain amount of data, even one byte, is written though your buffer is greater. You have to watch out so you don't unintentionally forget to transmit the rest of your data.

See Also

- socket_accept()
- socket_bind()
- socket_connect()
- socket_listen()
- socket_read()
- socket_strerror()