Session Handling

Introduction

Session support in PHP consists of a way to preserve certain data across subsequent accesses. This enables you to build more customized applications and increase the appeal of your web site.

A visitor accessing your web site is assigned a unique id, the so-called session id. This is either stored in a cookie on the user side or is propagated in the URL.

The session support allows you to register arbitrary numbers of variables to be preserved across requests. When a visitor accesses your site, PHP will check automatically (if session.auto_start is set to 1) or on your request (explicitly through session_start() or implicitly through session_register()) whether a specific session id has been sent with the request. If this is the case, the prior saved environment is recreated.

Caution
If you do turn on session.auto_start then you cannot put objects into your sessions since the class definition has to be loaded before starting the session in order to recreate the objects in your session.

All registered variables are serialized after the request finishes. Registered variables which are undefined are marked as being not defined. On subsequent accesses, these are not defined by the session module unless the user defines them later.

Warning
Some types of data can not be serialized thus stored in sessions. It includes resource variables or objects with circular references (i.e. objects which passes a reference to itself to another object).

Note
Session handling was added in PHP 4.0.0.

Note
Please note when working with sessions that a record of a session is not created until a variable has been registered using the session_register() function or by adding a new key to the *$_SESSION* superglobal array. This holds true regardless of if a session has been started using the session_start() function.

Installing/Configuring

Requirements

No external libraries are needed to build this extension.

Note
Optionally you can use shared memory allocation (mm), developed by Ralf S. Engelschall, for session storage. You have to download » mm and install it. This option is not available for Windows platforms. Note that the session storage module for mm does not guarantee that concurrent accesses to the same session are properly locked. It might be more appropriate to use a shared memory based filesystem (such as tmpfs on Solaris/Linux, or /dev/md on BSD) to store sessions in files, because they are properly locked. Session data is stored in memory thus web server restart deletes it.

Installation

Session support is enabled in PHP by default. If you would not like to build your PHP with session support, you should specify the *--disable-session* option to configure. To use shared memory allocation (mm) for session storage configure PHP *--with-mm[=DIR]*.

The Windows version of PHP has built-in support for this extension. You do not need to load any additional extensions in order to use these functions.

Note
By default, all data related to a particular session will be stored in a file in the directory specified by the session.save_path INI option. A file for each session (regardless of if any data is associated with that session) will be created. This is due to the fact that a session is opened (a file is created) but no data is even written to that file. Note that this behavior is a side-effect of the limitations of working with the file system and it is possible that a custom session handler (such as one which uses a database) does not keep track of sessions which store no data.

Runtime Configuration

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

Session configuration options

Name	Default	Changeable	Changelog
session.save_path	""	PHP_INI_ALL	
session.name	"PHPSESSID"	PHP_INI_ALL	
session.save_handler	"files"	PHP_INI_ALL	
session.auto_start	"0"	PHP_INI_ALL	
session.gc_probability	"1"	PHP_INI_ALL	
session.gc_divisor	"100"	PHP_INI_ALL	Available since PHP 4.3.2.
session.gc_maxlifetime	"1440"	PHP_INI_ALL	
session.serialize_handler	"php"	PHP_INI_ALL	
session.cookie_lifetime	"0"	PHP_INI_ALL	
session.cookie_path	"/"	PHP_INI_ALL	
session.cookie_domain	""	PHP_INI_ALL	
session.cookie_secure	""	PHP_INI_ALL	Available since PHP 4.0.4.
session.cookie_httponly	""	PHP_INI_ALL	Available since PHP 5.2.0.
session.use_cookies	"1"	PHP_INI_ALL	
session.use_only_cookies	"1"	PHP_INI_ALL	Available since PHP 4.3.0.
session.referer_check	""	PHP_INI_ALL	
session.entropy_file	""	PHP_INI_ALL	
session.entropy_length	"0"	PHP_INI_ALL	
session.cache_limiter	"nocache"	PHP_INI_ALL	
session.cache_expire	"180"	PHP_INI_ALL	

session.use_trans_sid	"0"	PHP_INI_ALL	PHP_INI_ALL in PHP <= 4.2.3. PHP_INI_PERDIR in PHP < 5. Available since PHP 4.0.3.
session.bug_compat_42	"1"	PHP_INI_ALL	Available since PHP 4.3.0. Removed in PHP 6.0.0.
session.bug_compat_warn	"1"	PHP_INI_ALL	Available since PHP 4.3.0. Removed in PHP 6.0.0.
session.hash_function	"0"	PHP_INI_ALL	Available since PHP 5.0.0.
session.hash_bits_per_character	"4"	PHP_INI_ALL	Available since PHP 5.0.0.
url_rewriter.tags	"a=href,area=href,frame=src,form=,fieldset="	PHP_INI_ALL	Available since PHP 4.0.4.

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

The session management system supports a number of configuration options which you can place in your *php.ini* file. We will give a short overview.

`session.save_handler` string

session.save_handler defines the name of the handler which is used for storing and retrieving data associated with a session. Defaults to *files*. Note that individual extensions may register their own *save_handler* s; registered handlers can be obtained on a per-installation basis by referring to phpinfo(). See also session_set_save_handler().

`session.save_path` string

session.save_path defines the argument which is passed to the save handler. If you choose the default files handler, this is the path where the files are created. See also session_save_path(). There is an optional N argument to this directive that determines the number of directory levels your session files will be spread around in. For example, setting to *'5;/tmp'* may end up creating a session file and location like */tmp/4/b/1/e/3/sess_4b1e384ad74619bd212e236e52a5a174If*. In order to use N you must create all of these directories before use. A small shell script exists in *ext/session* to do this, it's called *mod_files.sh*. Also note that if N is used and greater than 0 then automatic garbage collection will not be performed, see a copy of *php.ini* for further information. Also, if you use N, be sure to surround *session.save_path* in "quotes" because the separator (*;*) is also used for comments in *php.ini*.

Warning
If you leave this set to a world-readable directory, such as */tmp* (the default), other users on the server may be able to hijack sessions by getting the list of files in that

directory.

Note

Prior to PHP 4.3.6, Windows users had to change this variable in order to use PHP's session functions. A valid path must be specified, e.g.: *c:/temp*.

`session.name` string
session.name specifies the name of the session which is used as cookie name. It should only contain alphanumeric characters. Defaults to *PHPSESSID*. See also session_name().

`session.auto_start` boolean
session.auto_start specifies whether the session module starts a session automatically on request startup. Defaults to *0* (disabled).

`session.serialize_handler` string
session.serialize_handler defines the name of the handler which is used to serialize/deserialize data. Currently, a PHP internal format (name *php* or *php_binary*) and WDDX are supported (name *wddx*). WDDX is only available, if PHP is compiled with WDDX support. Defaults to *php*.

`session.gc_probability` integer
session.gc_probability in conjunction with *session.gc_divisor* is used to manage probability that the gc (garbage collection) routine is started. Defaults to *1*. See session.gc_divisor for details.

`session.gc_divisor` integer
session.gc_divisor coupled with *session.gc_probability* defines the probability that the gc (garbage collection) process is started on every session initialization. The probability is calculated by using gc_probability/gc_divisor, e.g. 1/100 means there is a 1% chance that the GC process starts on each request. *session.gc_divisor* defaults to *100*.

`session.gc_maxlifetime` integer
session.gc_maxlifetime specifies the number of seconds after which data will be seen as 'garbage' and cleaned up. Garbage collection occurs during session start.

Note

If different scripts have different values of *session.gc_maxlifetime* but share the same place for storing the session data then the script with the minimum value will be cleaning the data. In this case, use this directive together with session.save_path.

Note

If you are using the default file-based session handler, your filesystem must keep track of access times (atime). Windows FAT does not so you will have to come up with another way to handle garbage collecting your session if you are stuck with a FAT filesystem or any other filesystem where atime tracking is not available. Since PHP 4.2.3 it has used mtime (modified date) instead of atime. So, you won't have problems with filesystems where atime tracking is not available.

`session.referer_check` string
: *session.referer_check* contains the substring you want to check each HTTP Referer for. If the Referer was sent by the client and the substring was not found, the embedded session id will be marked as invalid. Defaults to the empty string.

`session.entropy_file` string
: *session.entropy_file* gives a path to an external resource (file) which will be used as an additional entropy source in the session id creation process. Examples are */dev/random* or */dev/urandom* which are available on many Unix systems.

`session.entropy_length` integer
: *session.entropy_length* specifies the number of bytes which will be read from the file specified above. Defaults to *0* (disabled).

`session.use_cookies` boolean
: *session.use_cookies* specifies whether the module will use cookies to store the session id on the client side. Defaults to *1* (enabled).

`session.use_only_cookies` boolean
: *session.use_only_cookies* specifies whether the module will *only* use cookies to store the session id on the client side. Enabling this setting prevents attacks involved passing session ids in URLs. This setting was added in PHP 4.3.0. Defaults to *1* (enabled) since PHP 6.0.

`session.cookie_lifetime` integer
: *session.cookie_lifetime* specifies the lifetime of the cookie in seconds which is sent to the browser. The value 0 means "until the browser is closed." Defaults to *0*. See also session_get_cookie_params() and session_set_cookie_params().

Note

The expiration timestamp is set relative to the server time, which is not necessarily the same as the time in the client's browser.

`session.cookie_path` string
: *session.cookie_path* specifies path to set in session_cookie. Defaults to /. See also session_get_cookie_params() and session_set_cookie_params().

`session.cookie_domain` string
: *session.cookie_domain* specifies the domain to set in session_cookie. Default is none at all meaning the host name of the server which generated the cookie according to cookies specification. See also session_get_cookie_params() and session_set_cookie_params().

`session.cookie_secure` boolean
: *session.cookie_secure* specifies whether cookies should only be sent over secure connections. Defaults to *off*. This setting was added in PHP 4.0.4. See also session_get_cookie_params() and session_set_cookie_params().

`session.cookie_httponly` boolean
: Marks the cookie as accessible only through the HTTP protocol. This means that the cookie won't be accessible by scripting languages, such as JavaScript. This setting can effectively help to reduce identity theft through XSS attacks (although it is not supported by all browsers).

`session.cache_limiter` string
: *session.cache_limiter* specifies cache control method to use for session pages (none/nocache/private/private_no_expire/public). Defaults to *nocache*. See also session_cache_limiter().

`session.cache_expire` integer
: *session.cache_expire* specifies time-to-live for cached session pages in minutes, this has no effect for nocache limiter. Defaults to *180*. See also session_cache_expire().

`session.use_trans_sid` boolean
: *session.use_trans_sid* whether transparent sid support is enabled or not. Defaults to *0* (disabled).

Note
For PHP 4.1.2 or less, it is enabled by compiling with *--enable-trans-sid*. From PHP 4.2.0, trans-sid feature is always compiled. URL based session management has additional security risks compared to cookie based session management. Users may send a URL that contains an active session ID to their friends by email or users may save a URL that contains a session ID to their bookmarks and access your site with the same session ID always, for example.

`session.bug_compat_42` boolean
: PHP versions 4.2.3 and lower have an undocumented feature/bug that allows you to initialize a session variable in the global scope, albeit register_globals is disabled. PHP 4.3.0 and later will warn you, if this feature is used, and if session.bug_compat_warn is also enabled. This feature/bug can be disabled by disabling this directive.

`session.bug_compat_warn` boolean
: PHP versions 4.2.3 and lower have an undocumented feature/bug that allows you to initialize a session variable in the global scope, albeit register_globals is disabled. PHP

4.3.0 and later will warn you, if this feature is used by enabling both session.bug_compat_42 and session.bug_compat_warn.

`session.hash_function` mixed

session.hash_function allows you to specify the hash algorithm used to generate the session IDs. '0' means MD5 (128 bits) and '1' means SHA-1 (160 bits). Since PHP 6.0.0 it is also possible to specify any of the algorithms provided by the hash extension (if it is available), like *sha512* or *whirlpool*. A complete list of supported algorithms can be obtained with the hash_algos() function.

Note
This was introduced in PHP 5.

`session.hash_bits_per_character` integer

session.hash_bits_per_character allows you to define how many bits are stored in each character when converting the binary hash data to something readable. The possible values are '4' (0-9, a-f), '5' (0-9, a-v), and '6' (0-9, a-z, A-Z, "-", ",").

Note
This was introduced in PHP 5.

`url_rewriter.tags` string

url_rewriter.tags specifies which HTML tags are rewritten to include session id if transparent sid support is enabled. Defaults to *a=href,area=href,frame=src,input=src,form=fakeentry,fieldset=*

Note
If you want HTML/XHTML strict conformity, remove the *form* entry and use the <fieldset> tags around your form fields.

The *track_vars* and *register_globals* configuration settings influence how the session variables get stored and restored.

Note
As of PHP 4.0.3, *track_vars* is always turned on.

Resource Types

This extension has no resource types defined.

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.

SID (string)
Constant containing either the session name and session ID in the form of *"name=ID"* or empty string if session ID was set in an appropriate session cookie.

Examples

Note

As of PHP 4.1.0, *$_SESSION* is available as a global variable just like *$_POST*, *$_GET*, *$_REQUEST* and so on. Unlike *$HTTP_SESSION_VARS*, *$_SESSION* is always global. Therefore, you do not need to use the *global* keyword for *$_SESSION*. Please note that this documentation has been changed to use *$_SESSION* everywhere. You can substitute *$HTTP_SESSION_VARS* for *$_SESSION*, if you prefer the former. Also note that you must start your session using session_start() before use of *$_SESSION* becomes available.

The keys in the *$_SESSION* associative array are subject to the same limitations as regular variable names in PHP, i.e. they cannot start with a number and must start with a letter or underscore. For more details see the section on variables in this manual.

If register_globals is disabled, only members of the global associative array *$_SESSION* can be registered as session variables. The restored session variables will only be available in the array *$_SESSION*.

Use of *$_SESSION* (or *$HTTP_SESSION_VARS* with PHP 4.0.6 or less) is recommended for improved security and code readability. With *$_SESSION*, there is no need to use the session_register(), session_unregister(), session_is_registered() functions. Session variables are accessible like any other variables.

Example #1 - Registering a variable with *$_SESSION*.

```
<?php
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
if (!isset($_SESSION['count'])) {
 $_SESSION['count'] = 0;
} else {
 $_SESSION['count']++;
}
?>
```

Example #2 - Unregistering a variable with *$_SESSION* and register_globals disabled.

```
<?php
session_start();
// Use $HTTP_SESSION_VARS with PHP 4.0.6 or less
unset($_SESSION['count']);
?>
```

Caution
Do NOT unset the whole *$_SESSION* with *unset($_SESSION)* as this will disable the registering of session variables through the *$_SESSION* superglobal.

Warning
You can't use references in session variables as there is no feasible way to restore a reference to another variable.

If register_globals is enabled, then each global variable can be registered as session variable. Upon a restart of a session, these variables will be restored to corresponding global variables. Since PHP must know which global variables are registered as session variables, users need to register variables with session_register() function. You can avoid this by simply setting entries in *$_SESSION*.

Caution
Before PHP 4.3.0, if you are using *$_SESSION* and you have disabled register_globals, don't use session_register(), session_is_registered() or session_unregister(). Disabling register_globals is recommended for both security and performance reasons.

If register_globals is enabled, then the global variables and the *$_SESSION* entries will automatically reference the same values which were registered in the prior session instance. However, if the variable is registered by *$_SESSION* then the global variable is available since the next request.

There is a defect in PHP 4.2.3 and earlier. If you register a new session variable by using session_register(), the entry in the global scope and the *$_SESSION* entry will not reference the same value until the next session_start(). I.e. a modification to the newly registered global variable will not be reflected by the *$_SESSION* entry. This has been corrected in PHP 4.3.0.

Passing the Session ID

There are two methods to propagate a session id:

- Cookies
- URL parameter

The session module supports both methods. Cookies are optimal, but because they are not always available, we also provide an alternative way. The second method embeds the session id directly into URLs.

PHP is capable of transforming links transparently. Unless you are using PHP 4.2.0 or later, you need to enable it manually when building PHP. Under Unix, pass --enable-trans-sid to configure. If this build option and the run-time option *session.use_trans_sid* are enabled, relative URIs will be changed to contain the session id automatically.

Note

The arg_separator.output *php.ini* directive allows to customize the argument seperator. For full XHTML conformance, specify & there.

Alternatively, you can use the constant *SID* which is defined if the session started. If the client did not send an appropriate session cookie, it has the form *session_name=session_id*. Otherwise, it expands to an empty string. Thus, you can embed it unconditionally into URLs.

The following example demonstrates how to register a variable, and how to link correctly to another page using SID.

Example #3 - Counting the number of hits of a single user

```
<?php

session_start();

if (empty($_SESSION['count'])) {
$_SESSION['count'] = 1;
} else {
$_SESSION['count']++;
}
?>

<p>
Hello visitor, you have seen this page <?php echo $_SESSION['count']; ?>
times.
</p>

<p>
To continue, <a href="nextpage.php?<?php echo htmlspecialchars(SID);
?>">click
here</a>.
</p>
```

The htmlspecialchars() may be used when printing the SID in order to prevent XSS related attacks.

Printing the SID, like shown above, is not necessary if --enable-trans-sid was used to

compile PHP.

Note
Non-relative URLs are assumed to point to external sites and hence don't append the SID, as it would be a security risk to leak the SID to a different server.

Custom Session Handlers

To implement database storage, or any other storage method, you will need to use session_set_save_handler() to create a set of user-level storage functions.

Sessions and security

External links: » Session fixation

The session module cannot guarantee that the information you store in a session is only viewed by the user who created the session. You need to take additional measures to actively protect the integrity of the session, depending on the value associated with it.

Assess the importance of the data carried by your sessions and deploy additional protections -- this usually comes at a price, reduced convenience for the user. For example, if you want to protect users from simple social engineering tactics, you need to enable *session.use_only_cookies*. In that case, cookies must be enabled unconditionally on the user side, or sessions will not work.

There are several ways to leak an existing session id to third parties. A leaked session id enables the third party to access all resources which are associated with a specific id. First, URLs carrying session ids. If you link to an external site, the URL including the session id might be stored in the external site's referrer logs. Second, a more active attacker might listen to your network traffic. If it is not encrypted, session ids will flow in plain text over the network. The solution here is to implement SSL on your server and make it mandatory for users.

Session Functions

session_cache_expire

session_cache_expire -- Return current cache expire

Description

int **session_cache_expire** ([int `$new_cache_expire`])

session_cache_expire() returns the current setting of *session.cache_expire*.

The cache expire is reset to the default value of 180 stored in *session.cache_limiter* at request startup time. Thus, you need to call session_cache_expire() for every request (and before session_start() is called).

Parameters

`new_cache_expire`
If *`new_cache_expire`* is given, the current cache expire is replaced with *`new_cache_expire`*.

Note
Setting *`new_cache_expire`* is of value only, if *session.cache_limiter* is set to a value *different* from *nocache*.

Return Values

Returns the current setting of *session.cache_expire*. The value returned should be read in minutes, defaults to 180.

Examples

Example #4 - session_cache_expire() example

```
<?php

/* set the cache limiter to 'private' */

session_cache_limiter('private');
$cache_limiter = session_cache_limiter();

/* set the cache expire to 30 minutes */
session_cache_expire(30);
$cache_expire = session_cache_expire();
```

```
/* start the session */

session_start();

echo "The cache limiter is now set to $cache_limiter<br />";
echo "The cached session pages expire after $cache_expire minutes";
?>
```

See Also

- session.cache_expire
- session.cache_limiter
- session_cache_limiter()

session_cache_limiter

session_cache_limiter -- Get and/or set the current cache limiter

Description

string **session_cache_limiter** ([string `$cache_limiter`])

session_cache_limiter() returns the name of the current cache limiter.

The cache limiter defines which cache control HTTP headers are sent to the client. These headers determine the rules by which the page content may be cached by the client and intermediate proxies. Setting the cache limiter to *nocache* disallows any client/proxy caching. A value of *public* permits caching by proxies and the client, whereas *private* disallows caching by proxies and permits the client to cache the contents.

In *private* mode, the Expire header sent to the client may cause confusion for some browsers, including Mozilla. You can avoid this problem by using *private_no_expire* mode. The expire header is never sent to the client in this mode.

The cache limiter is reset to the default value stored in session.cache_limiter at request startup time. Thus, you need to call session_cache_limiter() for every request (and before session_start() is called).

Parameters

`cache_limiter`
: If *`cache_limiter`* is specified, the name of the current cache limiter is changed to the new value.

Return Values

Returns the name of the current cache limiter.

ChangeLog

Version	Description
4.2.0	The *private_no_expire* cache limiter was added.

Examples

Example #5 - session_cache_limiter() example

```
<?php

/* set the cache limiter to 'private' */

session_cache_limiter('private');
$cache_limiter = session_cache_limiter();

echo "The cache limiter is now set to $cache_limiter<br />";
?>
```

See Also

- session.cache_limiter

session_commit

session_commit -- Alias of session_write_close()

Description

This function is an alias of: session_write_close().

session_decode

session_decode -- Decodes session data from a string

Description

bool **session_decode** (string `$data`)

session_decode() decodes the session data in *`data`*, setting variables stored in the session.

Parameters

`data`
: The encoded data to be stored.

Return Values

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

See Also

- session_encode()

session_destroy

session_destroy -- Destroys all data registered to a session

Description

bool **session_destroy** (void)

session_destroy() destroys all of the data associated with the current session. It does not unset any of the global variables associated with the session, or unset the session cookie.

In order to kill the session altogether, like to log the user out, the session id must also be unset. If a cookie is used to propagate the session id (default behavior), then the session cookie must be deleted. setcookie() may be used for that.

Return Values

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

Examples

Example #6 - Destroying a session with *$_SESSION*

```
<?php
// Initialize the session.
// If you are using session_name("something"), don't forget it now!
session_start();

// Unset all of the session variables.
$_SESSION = array();

// If it's desired to kill the session, also delete the session cookie.
// Note: This will destroy the session, and not just the session data!
if (isset($_COOKIE[session_name()])) {
   setcookie(session_name(), '', time()-42000, '/');
}

// Finally, destroy the session.
session_destroy();
?>
```

Notes

Note

Only use session_unset() for older deprecated code that does not use *$_SESSION*.

See Also

- unset()
- setcookie()

session_encode

session_encode -- Encodes the current session data as a string

Description

string **session_encode** (void)

session_encode() returns a string with the contents of the current session encoded within.

Return Values

Returns the contents of the current session encoded.

See Also

- session_decode()

session_get_cookie_params

session_get_cookie_params -- Get the session cookie parameters

Description

array **session_get_cookie_params** (void)

Gets the session cookie parameters.

Return Values

Returns an array with the current session cookie information, the array contains the following items:

- "lifetime" - The lifetime of the cookie in seconds.
- "path" - The path where information is stored.
- "domain" - The domain of the cookie.
- "secure" - The cookie should only be sent over secure connections.
- "httponly" - The cookie can only be accessed through the HTTP protocol.

ChangeLog

Version	Description
5.2.0	The "httponly" entry was added in the returned array.
4.0.4	The "secure" entry was added in the returned array.

See Also

- session.cookie_lifetime
- session.cookie_path
- session.cookie_domain
- session.cookie_secure
- session.cookie_httponly
- session.cookie_lifetime

- session_set_cookie_params()

session_id

session_id -- Get and/or set the current session id

Description

string **session_id** ([string `$id`])

session_id() is used to get or set the session id for the current session.

The constant `SID` can also be used to retrieve the current name and session id as a string suitable for adding to URLs. See also Session handling.

Parameters

`id`

If *`id`* is specified, it will replace the current session id. session_id() needs to be called before session_start() for that purpose. Depending on the session handler, not all characters are allowed within the session id. For example, the file session handler only allows characters in the range *a-z, A-Z and 0-9* !

Note
When using session cookies, specifying an *`id`* for session_id() will always send a new cookie when session_start() is called, regardless if the current session id is identical to the one being set.

Return Values

session_id() returns the session id for the current session or the empty string ("") if there is no current session (no current session id exists).

See Also

- session_regenerate_id()
- session_start()
- session_set_save_handler()
- session.save_handler

session_is_registered

session_is_registered -- Find out whether a global variable is registered in a session

Description

bool **session_is_registered** (string `$name`)

Finds out whether a global variable is registered in a session.

Parameters

`name`
The variable name.

Return Values

session_is_registered() returns **TRUE** if there is a global variable with the name *`name`* registered in the current session, **FALSE** otherwise.

Notes

Note
If *$_SESSION* (or *$HTTP_SESSION_VARS* for PHP 4.0.6 or less) is used, use isset() to check a variable is registered in *$_SESSION*.

Caution
If you are using *$_SESSION* (or *$HTTP_SESSION_VARS*), do not use session_register(), session_is_registered() and session_unregister().

session_module_name

session_module_name -- Get and/or set the current session module

Description

string **session_module_name** ([string `$module`])

session_module_name() gets the name of the current session module.

Parameters

`module`
: If *`module`* is specified, that module will be used instead.

Return Values

Returns the name of the current session module.

session_name

session_name -- Get and/or set the current session name

Description

string **session_name** ([string `$name`])

session_name() returns the name of the current session.

The session name is reset to the default value stored in *session.name* at request startup time. Thus, you need to call session_name() for every request (and before session_start() or session_register() are called).

Parameters

name
: The session name references the session id in cookies and URLs. It should contain only alphanumeric characters; it should be short and descriptive (i.e. for users with enabled cookie warnings). If *name* is specified, the name of the current session is changed to its value.

Warning
The session name can't consist of digits only, at least one letter must be present. Otherwise a new session id is generated every time.

Return Values

Returns the name of the current session.

Examples

Example #7 - session_name() example

```
<?php

/* set the session name to WebsiteID */

$previous_name = session_name("WebsiteID");

echo "The previous session name was $previous_name<br />";
?>
```

See Also

- The session.name configuration directive

session_regenerate_id

session_regenerate_id -- Update the current session id with a newly generated one

Description

bool **session_regenerate_id** ([bool `$delete_old_session`])

session_regenerate_id() will replace the current session id with a new one, and keep the current session information.

Parameters

`delete_old_session`
Whether to delete the old associated session file or not. Defaults to **FALSE**.

Return Values

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

ChangeLog

Version	Description
4.3.3	Since then, if session cookies are enabled, use of session_regenerate_id() will also submit a new session cookie with the new session id.
5.1.0	Added the *`delete_old_session`* parameter.

Examples

Example #8 - A session_regenerate_id() example

```
<?php
session_start();

$old_sessionid = session_id();

session_regenerate_id();
```

```
$new_sessionid = session_id();

echo "Old Session: $old_sessionid<br />";
echo "New Session: $new_sessionid<br />";

print_r($_SESSION);
?>
```

See Also

- session_id()
- session_start()
- session_name()

session_register

session_register -- Register one or more global variables with the current session

Description

bool **session_register** (mixed `$name` [, mixed `$...`])

session_register() accepts a variable number of arguments, any of which can be either a string holding the name of a variable or an array consisting of variable names or other arrays. For each name, session_register() registers the global variable with that name in the current session.

You can also create a session variable by simply setting the appropriate member of the *$_SESSION* or *$HTTP_SESSION_VARS* (PHP < 4.1.0) array.

```
<?php
// Use of session_register() is deprecated
$barney = "A big purple dinosaur.";
session_register("barney");

// Use of $_SESSION is preferred, as of PHP 4.1.0
$_SESSION["zim"] = "An invader from another planet.";

// The old way was to use $HTTP_SESSION_VARS
$HTTP_SESSION_VARS["spongebob"] = "He's got square pants.";
?>
```

If session_start() was not called before this function is called, an implicit call to session_start() with no parameters will be made. *$_SESSION* does not mimic this behavior and requires session_start() before use.

Parameters

name
: A string holding the name of a variable or an array consisting of variable names or other arrays.

...

Return Values

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

Notes

Caution

If you want your script to work regardless of register_globals, you need to instead use the $_SESSION array as *$_SESSION* entries are automatically registered. If your script uses session_register(), it will not work in environments where the PHP directive register_globals is disabled.

Note

register_globals: important note

As of PHP 4.2.0, the default value for the PHP directive register_globals is *off*, and it was completely removed as of PHP 6.0.0. The PHP community discourages developers from relying on this directive, and encourages the use of other means, such as the superglobals.

Caution

This registers a *global* variable. If you want to register a session variable from within a function, you need to make sure to make it global using the *global* keyword or the *$GLOBALS[]* array, or use the special session arrays as noted below.

Caution

If you are using *$_SESSION* (or *$HTTP_SESSION_VARS*), do not use session_register(), session_is_registered(), and session_unregister().

Note

It is currently impossible to register resource variables in a session. For example, you cannot create a connection to a database and store the connection id as a session variable and expect the connection to still be valid the next time the session is restored. PHP functions that return a resource are identified by having a return type of *resource* in their function definition. A list of functions that return resources are available in the resource types appendix.

If *$_SESSION* (or *$HTTP_SESSION_VARS* for PHP 4.0.6 or less) is used, assign values to *$_SESSION*. For example: $_SESSION['var'] = 'ABC';

See Also

- session_is_registered()
- session_unregister()
- $_SESSION

session_save_path

session_save_path -- Get and/or set the current session save path

Description

string **session_save_path** ([string `$path`])

session_save_path() returns the path of the current directory used to save session data.

Parameters

`path`
Session data path. If specified, the path to which data is saved will be changed. session_save_path() needs to be called before session_start() for that purpose.

Note
On some operating systems, you may want to specify a path on a filesystem that handles lots of small files efficiently. For example, on Linux, reiserfs may provide better performance than ext2fs.

Return Values

Returns the path of the current directory used for data storage.

See Also

- The session.save_path configuration directive

session_set_cookie_params

session_set_cookie_params -- Set the session cookie parameters

Description

void **session_set_cookie_params** (int `$lifetime` [, string `$path` [, string `$domain` [, bool `$secure` [, bool `$httponly`]]]])

Set cookie parameters defined in the *php.ini* file. The effect of this function only lasts for the duration of the script. Thus, you need to call session_set_cookie_params() for every request and before session_start() is called.

Parameters

`lifetime`

`path`

`domain`

`secure`

`httponly`

Return Values

No value is returned.

ChangeLog

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

See Also

- session.cookie_lifetime
- session.cookie_domain
- session.cookie_secure
- session.cookie_httponly
- session_get_cookie_params()

session_set_save_handler

session_set_save_handler -- Sets user-level session storage functions

Description

bool **session_set_save_handler** (callback `$open`, callback `$close`, callback `$read`, callback `$write`, callback `$destroy`, callback `$gc`)

session_set_save_handler() sets the user-level session storage functions which are used for storing and retrieving data associated with a session. This is most useful when a storage method other than those supplied by PHP sessions is preferred. i.e. Storing the session data in a local database.

Parameters

open

close

read
Read function must return string value always to make save handler work as expected. Return empty string if there is no data to read. Return values from other handlers are converted to boolean expression. **TRUE** for success, **FALSE** for failure.

write

Note
The "write" handler is not executed until after the output stream is closed. Thus, output from debugging statements in the "write" handler will never be seen in the browser. If debugging output is necessary, it is suggested that the debug output be written to a file instead.

destroy

gc

Return Values

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

Examples

Example #9 - session_set_save_handler() example

The following example provides file based session storage similar to the PHP sessions default save handler *files*. This example could easily be extended to cover database storage using your favorite PHP supported database engine.

```
<?php
function open($save_path, $session_name)
{
 global $sess_save_path;

 $sess_save_path = $save_path;
 return(true);
}

function close()
{
 return(true);
}

function read($id)
{
 global $sess_save_path;

 $sess_file = "$sess_save_path/sess_$id";
 return (string) @file_get_contents($sess_file);
}

function write($id, $sess_data)
{
 global $sess_save_path;

 $sess_file = "$sess_save_path/sess_$id";
 if ($fp = @fopen($sess_file, "w")) {
   $return = fwrite($fp, $sess_data);
   fclose($fp);
   return $return;
 } else {
   return(false);
 }

}

function destroy($id)
{
 global $sess_save_path;

 $sess_file = "$sess_save_path/sess_$id";
 return(@unlink($sess_file));
}

function gc($maxlifetime)
```

```
{
 global $sess_save_path;

 foreach (glob("$sess_save_path/sess_*") as $filename) {
   if (filemtime($filename) + $maxlifetime < time()) {
     @unlink($filename);
   }
 }
 return true;
}

session_set_save_handler("open", "close", "read", "write", "destroy", "gc");

session_start();

// proceed to use sessions normally

?>
```

Notes

Warning

As of PHP 5.0.5 the *write* and *close* handlers are called after object destruction and therefore cannot use objects or throw exceptions. The object destructors can however use sessions.

It is possible to call session_write_close() from the destructor to solve this chicken and egg problem.

Warning

Current working directory is changed with some SAPIs if session is closed in the script termination. It is possible to close the session earlier with session_write_close().

See Also

- The session.save_handler configuration directive

session_start

session_start -- Initialize session data

Description

bool **session_start** (void)

session_start() creates a session or resumes the current one based on the current session id that's being passed via a request, such as GET, POST, or a cookie.

If you want to use a named session, you must call session_name() before calling session_start().

session_start() will register internal output handler for URL rewriting when *trans-sid* is enabled. If a user uses *ob_gzhandler* or like with ob_start(), the order of output handler is important for proper output. For example, user must register *ob_gzhandler* before session start.

Return Values

This function always returns **TRUE**.

ChangeLog

Version	Description
4.3.3	As of now, calling session_start() while the session has already been started will result in an error of level **E_NOTICE**. Also, the second session start will simply be ignored.

Examples

Example #10 - A session example: *page1.php*

```
<?php
// page1.php

session_start();

echo 'Welcome to page #1';

$_SESSION['favcolor'] = 'green';
```

```
$_SESSION['animal']   = 'cat';
$_SESSION['time']     = time();

// Works if session cookie was accepted
echo '<br /><a href="page2.php">page 2</a>';

// Or maybe pass along the session id, if needed
echo '<br /><a href="page2.php?' . SID . '">page 2</a>';
?>
```

After viewing *page1.php*, the second page *page2.php* will magically contain the session data. Read the session reference for information on propagating session ids as it, for example, explains what the constant **SID** is all about.

Example #11 - A session example: *page2.php*

```
<?php
// page2.php

session_start();

echo 'Welcome to page #2<br />';

echo $_SESSION['favcolor']; // green
echo $_SESSION['animal'];   // cat
echo date('Y m d H:i:s', $_SESSION['time']);

// You may want to use SID here, like we did in page1.php
echo '<br /><a href="page1.php">page 1</a>';
?>
```

Notes

Note

If you are using cookie-based sessions, you must call session_start() before anything is outputted to the browser.

Note

Use of zlib.output_compression is recommended rather than ob_gzhandler()

See Also

- $_SESSION
- The session.auto_start configuration directive
- session_id()

session_unregister

session_unregister -- Unregister a global variable from the current session

Description

bool **session_unregister** (string `$name`)

session_unregister() unregisters the global variable named *name* from the current session.

Parameters

name
The variable name.

Return Values

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

Notes

Note
If *$_SESSION* (or *$HTTP_SESSION_VARS* for PHP 4.0.6 or less) is used, use unset() to unregister a session variable. Do not unset() *$_SESSION* itself as this will disable the special function of the *$_SESSION* superglobal.

Caution
This function does not unset the corresponding global variable for *name*, it only prevents the variable from being saved as part of the session. You must call unset() to remove the corresponding global variable.

Caution
If you are using *$_SESSION* (or *$HTTP_SESSION_VARS*), do not use session_register(), session_is_registered() and session_unregister().

session_unset

session_unset -- Free all session variables

Description

void **session_unset** (void)

The session_unset() function frees all session variables currently registered.

Return Values

No value is returned.

Notes

Note
If *$_SESSION* (or *$HTTP_SESSION_VARS* for PHP 4.0.6 or less) is used, use unset() to unregister a session variable, i.e. *unset ($_SESSION['varname']);*.

Caution
Do NOT unset the whole *$_SESSION* with *unset($_SESSION)* as this will disable the registering of session variables through the *$_SESSION* superglobal.

session_write_close

session_write_close -- Write session data and end session

Description

void **session_write_close** (void)

End the current session and store session data.

Session data is usually stored after your script terminated without the need to call session_write_close(), but as session data is locked to prevent concurrent writes only one script may operate on a session at any time. When using framesets together with sessions you will experience the frames loading one by one due to this locking. You can reduce the time needed to load all the frames by ending the session as soon as all changes to session variables are done.

Return Values

No value is returned.