session-set-save-handler

session_set_save_handler

Sets user-level session storage functions

ENTITY

bool

session_set_save_handler

callable

open

callable

close

callable

read

callable

write

callable

destroy

callable

gc

callable

create_sid

callable

validate_sid

callable

update_timestamp

Alternative signature (not supported with named arguments):

bool

session_set_save_handler

object

sessionhandler

bool

register_shutdown

ENTITY

`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, e.g. storing the session data in a local database.

ENTITY

This function has two prototypes.

`sessionhandler`
An instance of a class implementing SessionHandlerInterface, and optionally SessionIdInterface and/or SessionUpdateTimestampHandlerInterface, such as `SessionHandler`, to register as the session handler.

`register_shutdown`
Register `session_write_close` as a `register_shutdown_function` function.

or

`open`
A callable with the following signature: `bool`<open> `string``savePath` `string``sessionName`

The open callback works like a constructor in classes and is executed when the session is being opened. It is the first callback function executed when the session is started automatically or manually with `session_start`. Return value is ENTITY for success, ENTITY for failure.

`close`
A callable with the following signature: `bool`<close>

The close callback works like a destructor in classes and is executed after the session write callback has been called. It is also invoked when `session_write_close` is called. Return value should be ENTITY for success, ENTITY for failure.

`read`
A callable with the following signature: `stringfalse`<read> `string``sessionId`

The `read` callback must always return a session encoded (serialized) string, or an empty string if there is no data to read. It must return ENTITY to report a failure: `session_start` then emits an `E_WARNING` and returns ENTITY.

This callback is called internally by PHP when the session starts or when `session_start` is called. Before this callback is invoked PHP will invoke the `open` callback.

The value this callback returns must be in exactly the same serialized format that was originally passed for storage to the `write` callback. The value returned will be unserialized automatically by PHP and used to populate the `$_SESSION` superglobal. While the data looks similar to `serialize` please note it is a different format which is specified in the session.serialize_handler ini setting.

`write`
A callable with the following signature: `bool`<write> `string``sessionId` `string``data`

The `write` callback is called when the session needs to be saved and closed. This callback receives the current session ID a serialized version the `$_SESSION` superglobal. The serialization method used internally by PHP is specified in the session.serialize_handler ini setting.

The serialized session data passed to this callback should be stored against the passed session ID. When retrieving this data, the `read` callback must return the exact value that was originally passed to the `write` callback.

This callback is invoked when PHP shuts down or explicitly when `session_write_close` is called. Note that after executing this function PHP will internally execute the `close` callback.

> [!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`
A callable with the following signature: `bool`<destroy> `string``sessionId`

This callback is executed when a session is destroyed with `session_destroy` or with `session_regenerate_id` with the destroy parameter set to ENTITY. Return value should be ENTITY for success, ENTITY for failure.

`gc`
A callable with the following signature: `bool`<gc> `int``lifetime`

The garbage collector callback is invoked internally by PHP periodically in order to purge old session data. The frequency is controlled by session.gc_probability and session.gc_divisor. The value of lifetime which is passed to this callback can be set in session.gc_maxlifetime. Return value should be ENTITY for success, ENTITY for failure.

`create_sid`
A callable with the following signature: `string`<create_sid>

This callback is executed when a new session ID is required. No parameters are provided, and the return value should be a string that is a valid session ID for your handler.

`validate_sid`
A callable with the following signature: `bool`<validate_sid> `string``key`

This callback is executed when a session is to be started, a session ID is supplied and session.use_strict_mode is enabled. The `key` is the session ID to validate. A session ID is valid, if a session with that ID already exists. The return value should be ENTITY for success, ENTITY for failure.

`update_timestamp`
A callable with the following signature: `bool`<update_timestamp> `string``key` `string``val`

This callback is executed when a session is updated. `key` is the session ID, `val` is the session data. The return value should be ENTITY for success, ENTITY for failure.

ENTITY

ENTITY

ENTITY

ENTITY	ENTITY
8.4.0	Calling `session_set_save_handler` with more than two arguments is now deprecated. Use the two argument signature instead.

ENTITY

Custom session handler: see full code in `SessionHandlerInterface` synopsis.

We just show the invocation here, the full example can be seen in the `SessionHandlerInterface` synopsis linked above.

Note we use the OOP prototype with `session_set_save_handler` and register the shutdown function using the function's parameter flag. This is generally advised when registering objects as session save handlers.

```
<?php
class MySessionHandler implements SessionHandlerInterface
{
    // implement interfaces here
}

$handler = new MySessionHandler();
session_set_save_handler($handler, true);
session_start();

// proceed to set and retrieve values by key from $_SESSION
```

ENTITY

> [!WARNING] The `write` and `close` handlers are called after object destruction and therefore cannot use objects or throw exceptions. Exceptions are not able to be caught since will not be caught nor will any exception trace be displayed and the execution will just cease unexpectedly. 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 but the most reliable way is to register the shutdown function as described above.

> [!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`.

ENTITY

The session.save_handler configuration directive, The session.serialize_handler configuration directive., The `register_shutdown_function`, The `session_register_shutdown`, Refer to save_handler.inc for a full procedural reference implementation

---

□□□ #2
□ □□□ □□□ 2026-10-31 18:01:14 UTC
□ □□□ □□□ 2026-10-31 18:01:15 UTC