**Session Library**

The Session class permits you maintain a user’s “state” and track their activity while they browse your site.

CodeIgniter comes with a few session storage drivers:

* files (default; file-system based)
* database
* redis
* memcached

In addition, you may create your own, custom session drivers based on other kinds of storage, while still taking advantage of the features of the Session class.

* [Using the Session Class](https://codeigniter.com/user_guide/libraries/sessions.html#using-the-session-class)
  + [Initializing a Session](https://codeigniter.com/user_guide/libraries/sessions.html#initializing-a-session)
  + [How do Sessions work?](https://codeigniter.com/user_guide/libraries/sessions.html#how-do-sessions-work)
    - [A note about concurrency](https://codeigniter.com/user_guide/libraries/sessions.html#a-note-about-concurrency)
  + [What is Session Data?](https://codeigniter.com/user_guide/libraries/sessions.html#what-is-session-data)
  + [Retrieving Session Data](https://codeigniter.com/user_guide/libraries/sessions.html#retrieving-session-data)
  + [Adding Session Data](https://codeigniter.com/user_guide/libraries/sessions.html#adding-session-data)
  + [Removing Session Data](https://codeigniter.com/user_guide/libraries/sessions.html#removing-session-data)
  + [Flashdata](https://codeigniter.com/user_guide/libraries/sessions.html#flashdata)
  + [Tempdata](https://codeigniter.com/user_guide/libraries/sessions.html#tempdata)
  + [Destroying a Session](https://codeigniter.com/user_guide/libraries/sessions.html#destroying-a-session)
  + [Accessing session metadata](https://codeigniter.com/user_guide/libraries/sessions.html#accessing-session-metadata)
  + [Session Preferences](https://codeigniter.com/user_guide/libraries/sessions.html#session-preferences)
  + [Session Drivers](https://codeigniter.com/user_guide/libraries/sessions.html#session-drivers)
    - [Files Driver](https://codeigniter.com/user_guide/libraries/sessions.html#files-driver)
      * [Bonus Tip](https://codeigniter.com/user_guide/libraries/sessions.html#bonus-tip)
    - [Database Driver](https://codeigniter.com/user_guide/libraries/sessions.html#database-driver)
    - [Redis Driver](https://codeigniter.com/user_guide/libraries/sessions.html#redis-driver)
    - [Memcached Driver](https://codeigniter.com/user_guide/libraries/sessions.html#memcached-driver)
      * [Bonus Tip](https://codeigniter.com/user_guide/libraries/sessions.html#id1)
    - [Custom Drivers](https://codeigniter.com/user_guide/libraries/sessions.html#custom-drivers)
* [Class Reference](https://codeigniter.com/user_guide/libraries/sessions.html#class-reference)

[**Using the Session Class**](https://codeigniter.com/user_guide/libraries/sessions.html#id3)

[**Initializing a Session**](https://codeigniter.com/user_guide/libraries/sessions.html#id4)

Sessions will typically run globally with each page load, so the Session class should either be initialized in your [controller](https://codeigniter.com/user_guide/general/controllers.html) constructors, or it can be [auto-loaded](https://codeigniter.com/user_guide/general/autoloader.html) by the system. For the most part the session class will run unattended in the background, so simply initializing the class will cause it to read, create, and update sessions when necessary.

To initialize the Session class manually in your controller constructor, use the $this->load->library() method:

$this**->**load**->**library('session');

Once loaded, the Sessions library object will be available using:

$this**->**session

**Important**

Because the [Loader Class](https://codeigniter.com/user_guide/libraries/loader.html) is instantiated by CodeIgniter’s base controller, make sure to call parent::\_\_construct() before trying to load a library from inside a controller constructor.

[**How do Sessions work?**](https://codeigniter.com/user_guide/libraries/sessions.html#id5)

When a page is loaded, the session class will check to see if valid session cookie is sent by the user’s browser. If a sessions cookie does **not** exist (or if it doesn’t match one stored on the server or has expired) a new session will be created and saved.

If a valid session does exist, its information will be updated. With each update, the session ID may be regenerated if configured to do so.

It’s important for you to understand that once initialized, the Session class runs automatically. There is nothing you need to do to cause the above behavior to happen. You can, as you’ll see below, work with session data, but the process of reading, writing, and updating a session is automatic.

**Note**

Under CLI, the Session library will automatically halt itself, as this is a concept based entirely on the HTTP protocol.

[**A note about concurrency**](https://codeigniter.com/user_guide/libraries/sessions.html#id6)

Unless you’re developing a website with heavy AJAX usage, you can skip this section. If you are, however, and if you’re experiencing performance issues, then this note is exactly what you’re looking for.

Sessions in previous versions of CodeIgniter didn’t implement locking, which meant that two HTTP requests using the same session could run exactly at the same time. To use a more appropriate technical term - requests were non-blocking.

However, non-blocking requests in the context of sessions also means unsafe, because modifications to session data (or session ID regeneration) in one request can interfere with the execution of a second, concurrent request. This detail was at the root of many issues and the main reason why CodeIgniter 3.0 has a completely re-written Session library.

Why are we telling you this? Because it is likely that after trying to find the reason for your performance issues, you may conclude that locking is the issue and therefore look into how to remove the locks …

DO NOT DO THAT! Removing locks would be **wrong** and it will cause you more problems!

Locking is not the issue, it is a solution. Your issue is that you still have the session open, while you’ve already processed it and therefore no longer need it. So, what you need is to close the session for the current request after you no longer need it.

Long story short - call session\_write\_close() once you no longer need anything to do with session variables.

[**What is Session Data?**](https://codeigniter.com/user_guide/libraries/sessions.html#id7)

Session data is simply an array associated with a particular session ID (cookie).

If you’ve used sessions in PHP before, you should be familiar with PHP’s [$\_SESSION superglobal](https://php.net/manual/en/reserved.variables.session.php) (if not, please read the content on that link).

CodeIgniter gives access to its session data through the same means, as it uses the session handlers’ mechanism provided by PHP. Using session data is as simple as manipulating (read, set and unset values) the $\_SESSION array.

In addition, CodeIgniter also provides 2 special types of session data that are further explained below: flashdata and tempdata.

**Note**

In previous versions, regular session data in CodeIgniter was referred to as ‘userdata’. Have this in mind if that term is used elsewhere in the manual. Most of it is written to explain how the custom ‘userdata’ methods work.

[**Retrieving Session Data**](https://codeigniter.com/user_guide/libraries/sessions.html#id8)

Any piece of information from the session array is available through the $\_SESSION superglobal:

$\_SESSION['item']

Or through the magic getter:

$this**->**session**->**item

And for backwards compatibility, through the userdata() method:

$this**->**session**->**userdata('item');

Where item is the array key corresponding to the item you wish to fetch. For example, to assign a previously stored ‘name’ item to the $name variable, you will do this:

$name **=** $\_SESSION['name'];

*// or:*

$name **=** $this**->**session**->**name

*// or:*

$name **=** $this**->**session**->**userdata('name');

**Note**

The userdata() method returns NULL if the item you are trying to access does not exist.

If you want to retrieve all of the existing userdata, you can simply omit the item key (magic getter only works for properties):

$\_SESSION

*// or:*

$this**->**session**->**userdata();

[**Adding Session Data**](https://codeigniter.com/user_guide/libraries/sessions.html#id9)

Let’s say a particular user logs into your site. Once authenticated, you could add their username and e-mail address to the session, making that data globally available to you without having to run a database query when you need it.

You can simply assign data to the $\_SESSION array, as with any other variable. Or as a property of $this->session.

Alternatively, the old method of assigning it as “userdata” is also available. That however passing an array containing your new data to the set\_userdata() method:

$this**->**session**->**set\_userdata($array);

Where $array is an associative array containing your new data. Here’s an example:

$newdata **=** **array**(

'username' **=>** 'johndoe',

'email' **=>** 'johndoe@some-site.com',

'logged\_in' **=>** **TRUE**

);

$this**->**session**->**set\_userdata($newdata);

If you want to add userdata one value at a time, set\_userdata() also supports this syntax:

$this**->**session**->**set\_userdata('some\_name', 'some\_value');

If you want to verify that a session value exists, simply check with isset():

*// returns FALSE if the 'some\_name' item doesn't exist or is NULL,*

*// TRUE otherwise:*

isset($\_SESSION['some\_name'])

Or you can call has\_userdata():

$this**->**session**->**has\_userdata('some\_name');

[**Removing Session Data**](https://codeigniter.com/user_guide/libraries/sessions.html#id10)

Just as with any other variable, unsetting a value in $\_SESSION can be done through unset():

unset($\_SESSION['some\_name']);

*// or multiple values:*

unset(

$\_SESSION['some\_name'],

$\_SESSION['another\_name']

);

Also, just as set\_userdata() can be used to add information to a session, unset\_userdata() can be used to remove it, by passing the session key. For example, if you wanted to remove ‘some\_name’ from your session data array:

$this**->**session**->**unset\_userdata('some\_name');

This method also accepts an array of item keys to unset:

$array\_items **=** **array**('username', 'email');

$this**->**session**->**unset\_userdata($array\_items);

**Note**

In previous versions, the unset\_userdata() method used to accept an associative array of key => 'dummy value' pairs. This is no longer supported.

[**Flashdata**](https://codeigniter.com/user_guide/libraries/sessions.html#id11)

CodeIgniter supports “flashdata”, or session data that will only be available for the next request, and is then automatically cleared.

This can be very useful, especially for one-time informational, error or status messages (for example: “Record 2 deleted”).

It should be noted that flashdata variables are regular session vars, only marked in a specific way under the ‘\_\_ci\_vars’ key (please don’t touch that one, you’ve been warned).

To mark an existing item as “flashdata”:

$this**->**session**->**mark\_as\_flash('item');

If you want to mark multiple items as flashdata, simply pass the keys as an array:

$this**->**session**->**mark\_as\_flash(**array**('item', 'item2'));

To add flashdata:

$\_SESSION['item'] **=** 'value';

$this**->**session**->**mark\_as\_flash('item');

Or alternatively, using the set\_flashdata() method:

$this**->**session**->**set\_flashdata('item', 'value');

You can also pass an array to set\_flashdata(), in the same manner as set\_userdata().

Reading flashdata variables is the same as reading regular session data through $\_SESSION:

$\_SESSION['item']

**Important**

The userdata() method will NOT return flashdata items.

However, if you want to be sure that you’re reading “flashdata” (and not any other kind), you can also use the flashdata() method:

$this**->**session**->**flashdata('item');

Or to get an array with all flashdata, simply omit the key parameter:

$this**->**session**->**flashdata();

**Note**

The flashdata() method returns NULL if the item cannot be found.

If you find that you need to preserve a flashdata variable through an additional request, you can do so using the keep\_flashdata() method. You can either pass a single item or an array of flashdata items to keep.

$this**->**session**->**keep\_flashdata('item');

$this**->**session**->**keep\_flashdata(**array**('item1', 'item2', 'item3'));

[**Tempdata**](https://codeigniter.com/user_guide/libraries/sessions.html#id12)

CodeIgniter also supports “tempdata”, or session data with a specific expiration time. After the value expires, or the session expires or is deleted, the value is automatically removed.

Similarly to flashdata, tempdata variables are regular session vars that are marked in a specific way under the ‘\_\_ci\_vars’ key (again, don’t touch that one).

To mark an existing item as “tempdata”, simply pass its key and expiry time (in seconds!) to the mark\_as\_temp() method:

*// 'item' will be erased after 300 seconds*

$this**->**session**->**mark\_as\_temp('item', 300);

You can mark multiple items as tempdata in two ways, depending on whether you want them all to have the same expiry time or not:

*// Both 'item' and 'item2' will expire after 300 seconds*

$this**->**session**->**mark\_as\_temp(**array**('item', 'item2'), 300);

*// 'item' will be erased after 300 seconds, while 'item2'*

*// will do so after only 240 seconds*

$this**->**session**->**mark\_as\_temp(**array**(

'item' **=>** 300,

'item2' **=>** 240

));

To add tempdata:

$\_SESSION['item'] **=** 'value';

$this**->**session**->**mark\_as\_temp('item', 300); *// Expire in 5 minutes*

Or alternatively, using the set\_tempdata() method:

$this**->**session**->**set\_tempdata('item', 'value', 300);

You can also pass an array to set\_tempdata():

$tempdata **=** **array**('newuser' **=>** **TRUE**, 'message' **=>** 'Thanks for joining!');

$this**->**session**->**set\_tempdata($tempdata, **NULL**, $expire);

**Note**

If the expiration is omitted or set to 0, the default time-to-live value of 300 seconds (or 5 minutes) will be used.

To read a tempdata variable, again you can just access it through the $\_SESSION superglobal array:

$\_SESSION['item']

**Important**

The userdata() method will NOT return tempdata items.

Or if you want to be sure that you’re reading “tempdata” (and not any other kind), you can also use the tempdata() method:

$this**->**session**->**tempdata('item');

And of course, if you want to retrieve all existing tempdata:

$this**->**session**->**tempdata();

**Note**

The tempdata() method returns NULL if the item cannot be found.

If you need to remove a tempdata value before it expires, you can directly unset it from the $\_SESSION array:

unset($\_SESSION['item']);

However, this won’t remove the marker that makes this specific item to be tempdata (it will be invalidated on the next HTTP request), so if you intend to reuse that same key in the same request, you’d want to use unset\_tempdata():

$this**->**session**->**unset\_tempdata('item');

[**Destroying a Session**](https://codeigniter.com/user_guide/libraries/sessions.html#id13)

To clear the current session (for example, during a logout), you may simply use either PHP’s [session\_destroy()](https://php.net/session_destroy) function, or the sess\_destroy() method. Both will work in exactly the same way:

session\_destroy();

*// or*

$this**->**session**->**sess\_destroy();

**Note**

This must be the last session-related operation that you do during the same request. All session data (including flashdata and tempdata) will be destroyed permanently and functions will be unusable during the same request after you destroy the session.

[**Accessing session metadata**](https://codeigniter.com/user_guide/libraries/sessions.html#id14)

In previous CodeIgniter versions, the session data array included 4 items by default: ‘session\_id’, ‘ip\_address’, ‘user\_agent’, ‘last\_activity’.

This was due to the specifics of how sessions worked, but is now no longer necessary with our new implementation. However, it may happen that your application relied on these values, so here are alternative methods of accessing them:

* session\_id: session\_id()
* ip\_address: $\_SERVER['REMOTE\_ADDR']
* user\_agent: $this->input->user\_agent() (unused by sessions)
* last\_activity: Depends on the storage, no straightforward way. Sorry!

[**Session Preferences**](https://codeigniter.com/user_guide/libraries/sessions.html#id15)

CodeIgniter will usually make everything work out of the box. However, Sessions are a very sensitive component of any application, so some careful configuration must be done. Please take your time to consider all of the options and their effects.

You’ll find the following Session related preferences in your **application/config/config.php** file:

| **Preference** | **Default** | **Options** | **Description** |
| --- | --- | --- | --- |
| **sess\_driver** | files | files/database/redis/memcached/*custom* | The session storage driver to use. |
| **sess\_cookie\_name** | ci\_session | [A-Za-z\_-] characters only | The name used for the session cookie. |
| **sess\_expiration** | 7200 (2 hours) | Time in seconds (integer) | The number of seconds you would like the session to last. If you would like a non-expiring session (until browser is closed) set the value to zero: 0 |
| **sess\_save\_path** | NULL | None | Specifies the storage location, depends on the driver being used. |
| **sess\_match\_ip** | FALSE | TRUE/FALSE (boolean) | Whether to validate the user’s IP address when reading the session cookie. Note that some ISPs dynamically changes the IP, so if you want a non-expiring session you will likely set this to FALSE. |
| **sess\_time\_to\_update** | 300 | Time in seconds (integer) | This option controls how often the session class will regenerate itself and create a new session ID. Setting it to 0 will disable session ID regeneration. |
| **sess\_regenerate\_destroy** | FALSE | TRUE/FALSE (boolean) | Whether to destroy session data associated with the old session ID when auto-regenerating the session ID. When set to FALSE, the data will be later deleted by the garbage collector. |

**Note**

As a last resort, the Session library will try to fetch PHP’s session related INI settings, as well as legacy CI settings such as ‘sess\_expire\_on\_close’ when any of the above is not configured. However, you should never rely on this behavior as it can cause unexpected results or be changed in the future. Please configure everything properly.

In addition to the values above, the cookie and native drivers apply the following configuration values shared by the [Input](https://codeigniter.com/user_guide/libraries/input.html) and [Security](https://codeigniter.com/user_guide/libraries/security.html) classes:

| **Preference** | **Default** | **Description** |
| --- | --- | --- |
| **cookie\_domain** | ‘’ | The domain for which the session is applicable |
| **cookie\_path** | / | The path to which the session is applicable |
| **cookie\_secure** | FALSE | Whether to create the session cookie only on encrypted (HTTPS) connections |

**Note**

The ‘cookie\_httponly’ setting doesn’t have an effect on sessions. Instead the HttpOnly parameter is always enabled, for security reasons. Additionally, the ‘cookie\_prefix’ setting is completely ignored.

[**Session Drivers**](https://codeigniter.com/user_guide/libraries/sessions.html#id16)

As already mentioned, the Session library comes with 4 drivers, or storage engines, that you can use:

* files
* database
* redis
* memcached

By default, the [Files Driver](https://codeigniter.com/user_guide/libraries/sessions.html#files-driver) will be used when a session is initialized, because it is the most safe choice and is expected to work everywhere (virtually every environment has a file system).

However, any other driver may be selected via the $config['sess\_driver'] line in your **application/config/config.php** file, if you chose to do so. Have it in mind though, every driver has different caveats, so be sure to get yourself familiar with them (below) before you make that choice.

In addition, you may also create and use [Custom Drivers](https://codeigniter.com/user_guide/libraries/sessions.html#custom-drivers), if the ones provided by default don’t satisfy your use case.

**Note**

In previous CodeIgniter versions, a different, “cookie driver” was the only option and we have received negative feedback on not providing that option. While we do listen to feedback from the community, we want to warn you that it was dropped because it is **unsafe** and we advise you NOT to try to replicate it via a custom driver.

[**Files Driver**](https://codeigniter.com/user_guide/libraries/sessions.html#id17)

The ‘files’ driver uses your file system for storing session data.

It can safely be said that it works exactly like PHP’s own default session implementation, but in case this is an important detail for you, have it mind that it is in fact not the same code and it has some limitations (and advantages).

To be more specific, it doesn’t support PHP’s [directory level and mode formats used in session.save\_path](https://php.net/manual/en/session.configuration.php#ini.session.save-path), and it has most of the options hard-coded for safety. Instead, only absolute paths are supported for $config['sess\_save\_path'].

Another important thing that you should know, is to make sure that you don’t use a publicly-readable or shared directory for storing your session files. Make sure that *only you* have access to see the contents of your chosen *sess\_save\_path* directory. Otherwise, anybody who can do that, can also steal any of the current sessions (also known as “session fixation” attack).

On UNIX-like operating systems, this is usually achieved by setting the 0700 mode permissions on that directory via the *chmod* command, which allows only the directory’s owner to perform read and write operations on it. But be careful because the system user *running* the script is usually not your own, but something like ‘www-data’ instead, so only setting those permissions will probable break your application.

Instead, you should do something like this, depending on your environment

mkdir **/<**path to your application directory**>/**sessions**/**

chmod 0700 **/<**path to your application directory**>/**sessions**/**

chown www**-**data **/<**path to your application directory**>/**sessions**/**

[**Bonus Tip**](https://codeigniter.com/user_guide/libraries/sessions.html#id18)

Some of you will probably opt to choose another session driver because file storage is usually slower. This is only half true.

A very basic test will probably trick you into believing that an SQL database is faster, but in 99% of the cases, this is only true while you only have a few current sessions. As the sessions count and server loads increase - which is the time when it matters - the file system will consistently outperform almost all relational database setups.

In addition, if performance is your only concern, you may want to look into using [tmpfs](http://eddmann.com/posts/storing-php-sessions-file-caches-in-memory-using-tmpfs/), (warning: external resource), which can make your sessions blazing fast.

[**Database Driver**](https://codeigniter.com/user_guide/libraries/sessions.html#id19)

The ‘database’ driver uses a relational database such as MySQL or PostgreSQL to store sessions. This is a popular choice among many users, because it allows the developer easy access to the session data within an application - it is just another table in your database.

However, there are some conditions that must be met:

* Only your **default** database connection (or the one that you access as $this->db from your controllers) can be used.
* You must have the [Query Builder](https://codeigniter.com/user_guide/database/query_builder.html) enabled.
* You can NOT use a persistent connection.
* You can NOT use a connection with the *cache\_on* setting enabled.

In order to use the ‘database’ session driver, you must also create this table that we already mentioned and then set it as your $config['sess\_save\_path'] value. For example, if you would like to use ‘ci\_sessions’ as your table name, you would do this:

$config['sess\_driver'] **=** 'database';

$config['sess\_save\_path'] **=** 'ci\_sessions';

**Note**

If you’ve upgraded from a previous version of CodeIgniter and you don’t have ‘sess\_save\_path’ configured, then the Session library will look for the old ‘sess\_table\_name’ setting and use it instead. Please don’t rely on this behavior as it will get removed in the future.

And then of course, create the database table …

For MySQL:

CREATE TABLE **IF** **NOT** EXISTS `ci\_sessions` (

`id` varchar(128) **NOT** **NULL**,

`ip\_address` varchar(45) **NOT** **NULL**,

`timestamp` int(10) unsigned **DEFAULT** 0 **NOT** **NULL**,

`data` blob **NOT** **NULL**,

KEY `ci\_sessions\_timestamp` (`timestamp`)

);

For PostgreSQL:

CREATE TABLE "ci\_sessions" (

"id" varchar(128) **NOT** **NULL**,

"ip\_address" varchar(45) **NOT** **NULL**,

"timestamp" bigint **DEFAULT** 0 **NOT** **NULL**,

"data" text **DEFAULT** '' **NOT** **NULL**

);

CREATE INDEX "ci\_sessions\_timestamp" ON "ci\_sessions" ("timestamp");

You will also need to add a PRIMARY KEY **depending on your ‘sess\_match\_ip’ setting**. The examples below work both on MySQL and PostgreSQL:

*// When sess\_match\_ip = TRUE*

ALTER TABLE ci\_sessions ADD PRIMARY KEY (id, ip\_address);

*// When sess\_match\_ip = FALSE*

ALTER TABLE ci\_sessions ADD PRIMARY KEY (id);

*// To drop a previously created primary key (use when changing the setting)*

ALTER TABLE ci\_sessions DROP PRIMARY KEY;

**Important**

Only MySQL and PostgreSQL databases are officially supported, due to lack of advisory locking mechanisms on other platforms. Using sessions without locks can cause all sorts of problems, especially with heavy usage of AJAX, and we will not support such cases. Use session\_write\_close() after you’ve done processing session data if you’re having performance issues.

[**Redis Driver**](https://codeigniter.com/user_guide/libraries/sessions.html#id20)

**Note**

Since Redis doesn’t have a locking mechanism exposed, locks for this driver are emulated by a separate value that is kept for up to 300 seconds.

Redis is a storage engine typically used for caching and popular because of its high performance, which is also probably your reason to use the ‘redis’ session driver.

The downside is that it is not as ubiquitous as relational databases and requires the [phpredis](https://github.com/phpredis/phpredis) PHP extension to be installed on your system, and that one doesn’t come bundled with PHP. Chances are, you’re only be using the ‘redis’ driver only if you’re already both familiar with Redis and using it for other purposes.

Just as with the ‘files’ and ‘database’ drivers, you must also configure the storage location for your sessions via the $config['sess\_save\_path'] setting. The format here is a bit different and complicated at the same time. It is best explained by the *phpredis* extension’s README file, so we’ll simply link you to it:

<https://github.com/phpredis/phpredis#php-session-handler>

**Warning**

CodeIgniter’s Session library does NOT use the actual ‘redis’ session.save\_handler. Take note **only** of the path format in the link above.

For the most common case however, a simple host:port pair should be sufficient:

$config['sess\_driver'] **=** 'redis';

$config['sess\_save\_path'] **=** 'tcp://localhost:6379';

[**Memcached Driver**](https://codeigniter.com/user_guide/libraries/sessions.html#id21)

**Note**

Since Memcache doesn’t have a locking mechanism exposed, locks for this driver are emulated by a separate value that is kept for up to 300 seconds.

The ‘memcached’ driver is very similar to the ‘redis’ one in all of its properties, except perhaps for availability, because PHP’s [Memcached](https://php.net/memcached) extension is distributed via PECL and some Linux distrubutions make it available as an easy to install package.

Other than that, and without any intentional bias towards Redis, there’s not much different to be said about Memcached - it is also a popular product that is usually used for caching and famed for its speed.

However, it is worth noting that the only guarantee given by Memcached is that setting value X to expire after Y seconds will result in it being deleted after Y seconds have passed (but not necessarily that it won’t expire earlier than that time). This happens very rarely, but should be considered as it may result in loss of sessions.

The $config['sess\_save\_path'] format is fairly straightforward here, being just a host:port pair:

$config['sess\_driver'] **=** 'memcached';

$config['sess\_save\_path'] **=** 'localhost:11211';

[**Bonus Tip**](https://codeigniter.com/user_guide/libraries/sessions.html#id22)

Multi-server configuration with an optional *weight* parameter as the third colon-separated (:weight) value is also supported, but we have to note that we haven’t tested if that is reliable.

If you want to experiment with this feature (on your own risk), simply separate the multiple server paths with commas:

*// localhost will be given higher priority (5) here,*

*// compared to 192.0.2.1 with a weight of 1.*

$config['sess\_save\_path'] **=** 'localhost:11211:5,192.0.2.1:11211:1';

[**Custom Drivers**](https://codeigniter.com/user_guide/libraries/sessions.html#id23)

You may also create your own, custom session drivers. However, have it in mind that this is typically not an easy task, as it takes a lot of knowledge to do it properly.

You need to know not only how sessions work in general, but also how they work specifically in PHP, how the underlying storage mechanism works, how to handle concurrency, avoid deadlocks (but NOT through lack of locks) and last but not least - how to handle the potential security issues, which is far from trivial.

Long story short - if you don’t know how to do that already in raw PHP, you shouldn’t be trying to do it within CodeIgniter either. You’ve been warned.

If you only want to add some extra functionality to your sessions, just extend the base Session class, which is a lot more easier. Read the [Creating Libraries](https://codeigniter.com/user_guide/general/creating_libraries.html) article to learn how to do that.

Now, to the point - there are three general rules that you must follow when creating a session driver for CodeIgniter:

* Put your driver’s file under **application/libraries/Session/drivers/** and follow the naming conventions used by the Session class.

For example, if you were to create a ‘dummy’ driver, you would have a Session\_dummy\_driver class name, that is declared in *application/libraries/Session/drivers/Session\_dummy\_driver.php*.

* Extend the CI\_Session\_driver class.

This is just a basic class with a few internal helper methods. It is also extendable like any other library, if you really need to do that, but we are not going to explain how … if you’re familiar with how class extensions/overrides work in CI, then you already know how to do it. If not, well, you shouldn’t be doing it in the first place.

* Implement the [SessionHandlerInterface](https://php.net/sessionhandlerinterface) interface.

**Note**

You may notice that SessionHandlerInterface is provided by PHP since version 5.4.0. CodeIgniter will automatically declare the same interface if you’re running an older PHP version.

The link will explain why and how.

So, based on our ‘dummy’ driver example above, you’d end up with something like this:

*// application/libraries/Session/drivers/Session\_dummy\_driver.php:*

**class** **CI\_Session\_dummy\_driver** **extends** CI\_Session\_driver **implements** SessionHandlerInterface

{

**public** **function** **\_\_construct**(**&**$params)

{

*// DO NOT forget this*

**parent::**\_\_construct($params);

*// Configuration & other initializations*

}

**public** **function** **open**($save\_path, $name)

{

*// Initialize storage mechanism (connection)*

}

**public** **function** **read**($session\_id)

{

*// Read session data (if exists), acquire locks*

}

**public** **function** **write**($session\_id, $session\_data)

{

*// Create / update session data (it might not exist!)*

}

**public** **function** **close**()

{

*// Free locks, close connections / streams / etc.*

}

**public** **function** **destroy**($session\_id)

{

*// Call close() method & destroy data for current session (order may differ)*

}

**public** **function** **gc**($maxlifetime)

{

*// Erase data for expired sessions*

}

}

If you’ve done everything properly, you can now set your *sess\_driver* configuration value to ‘dummy’ and use your own driver. Congratulations!

[**Class Reference**](https://codeigniter.com/user_guide/libraries/sessions.html#id24)

***class*CI\_Session**

**userdata([*$key = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Session item key or NULL |
| **Returns:** | Value of the specified item key, or an array of all userdata |
| **Return type:** | mixed |

Gets the value for a specific $\_SESSION item, or an array of all “userdata” items if not key was specified.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. You should directly access $\_SESSION instead.

**all\_userdata()**

|  |  |
| --- | --- |
| **Returns:** | An array of all userdata |
| **Return type:** | array |

Returns an array containing all “userdata” items.

**Note**

This method is DEPRECATED. Use userdata() with no parameters instead.

**&get\_userdata()**

|  |  |
| --- | --- |
| **Returns:** | A reference to $\_SESSION |
| **Return type:** | array |

Returns a reference to the $\_SESSION array.

**Note**

This is a legacy method kept only for backwards compatibility with older applications.

**has\_userdata(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*string*) – Session item key |
| **Returns:** | TRUE if the specified key exists, FALSE if not |
| **Return type:** | bool |

Checks if an item exists in $\_SESSION.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. It is just an alias for isset($\_SESSION[$key]) - please use that instead.

**set\_userdata(*$data*[, *$value = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*mixed*) – An array of key/value pairs to set as session data, or the key for a single item * **$value** (*mixed*) – The value to set for a specific session item, if $data is a key |
| **Return type:** | void |

Assigns data to the $\_SESSION superglobal.

**Note**

This is a legacy method kept only for backwards compatibility with older applications.

**unset\_userdata(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Key for the session data item to unset, or an array of multiple keys |
| **Return type:** | void |

Unsets the specified key(s) from the $\_SESSION superglobal.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. It is just an alias for unset($\_SESSION[$key]) - please use that instead.

**mark\_as\_flash(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Key to mark as flashdata, or an array of multiple keys |
| **Returns:** | TRUE on success, FALSE on failure |
| **Return type:** | bool |

Marks a $\_SESSION item key (or multiple ones) as “flashdata”.

**get\_flash\_keys()**

|  |  |
| --- | --- |
| **Returns:** | Array containing the keys of all “flashdata” items. |
| **Return type:** | array |

Gets a list of all $\_SESSION that have been marked as “flashdata”.

**unmark\_flash(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Key to be un-marked as flashdata, or an array of multiple keys |
| **Return type:** | void |

Unmarks a $\_SESSION item key (or multiple ones) as “flashdata”.

**flashdata([*$key = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Flashdata item key or NULL |
| **Returns:** | Value of the specified item key, or an array of all flashdata |
| **Return type:** | mixed |

Gets the value for a specific $\_SESSION item that has been marked as “flashdata”, or an array of all “flashdata” items if no key was specified.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. You should directly access $\_SESSION instead.

**keep\_flashdata(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Flashdata key to keep, or an array of multiple keys |
| **Returns:** | TRUE on success, FALSE on failure |
| **Return type:** | bool |

Retains the specified session data key(s) as “flashdata” through the next request.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. It is just an alias for the mark\_as\_flash() method.

**set\_flashdata(*$data*[, *$value = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*mixed*) – An array of key/value pairs to set as flashdata, or the key for a single item * **$value** (*mixed*) – The value to set for a specific session item, if $data is a key |
| **Return type:** | void |

Assigns data to the $\_SESSION superglobal and marks it as “flashdata”.

**Note**

This is a legacy method kept only for backwards compatibility with older applications.

**mark\_as\_temp(*$key*[, *$ttl = 300*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Key to mark as tempdata, or an array of multiple keys * **$ttl** (*int*) – Time-to-live value for the tempdata, in seconds |
| **Returns:** | TRUE on success, FALSE on failure |
| **Return type:** | bool |

Marks a $\_SESSION item key (or multiple ones) as “tempdata”.

**get\_temp\_keys()**

|  |  |
| --- | --- |
| **Returns:** | Array containing the keys of all “tempdata” items. |
| **Return type:** | array |

Gets a list of all $\_SESSION that have been marked as “tempdata”.

**unmark\_temp(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Key to be un-marked as tempdata, or an array of multiple keys |
| **Return type:** | void |

Unmarks a $\_SESSION item key (or multiple ones) as “tempdata”.

**tempdata([*$key = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*mixed*) – Tempdata item key or NULL |
| **Returns:** | Value of the specified item key, or an array of all tempdata |
| **Return type:** | mixed |

Gets the value for a specific $\_SESSION item that has been marked as “tempdata”, or an array of all “tempdata” items if no key was specified.

**Note**

This is a legacy method kept only for backwards compatibility with older applications. You should directly access $\_SESSION instead.

**set\_tempdata(*$data*[, *$value = NULL*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$data** (*mixed*) – An array of key/value pairs to set as tempdata, or the key for a single item * **$value** (*mixed*) – The value to set for a specific session item, if $data is a key * **$ttl** (*int*) – Time-to-live value for the tempdata item(s), in seconds |
| **Return type:** | void |

Assigns data to the $\_SESSION superglobal and marks it as “tempdata”.

**Note**

This is a legacy method kept only for backwards compatibility with older applications.

**sess\_regenerate([*$destroy = FALSE*])**

|  |  |
| --- | --- |
| **Parameters:** | * **$destroy** (*bool*) – Whether to destroy session data |
| **Return type:** | void |

Regenerate session ID, optionally destroying the current session’s data.

**Note**

This method is just an alias for PHP’s native [session\_regenerate\_id()](https://php.net/session_regenerate_id) function.

**sess\_destroy()**

|  |  |
| --- | --- |
| **Return type:** | void |

Destroys the current session.

**Note**

This must be the *last* session-related function that you call. All session data will be lost after you do that.

**Note**

This method is just an alias for PHP’s native [session\_destroy()](https://php.net/session_destroy) function.

**\_\_get(*$key*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*string*) – Session item key |
| **Returns:** | The requested session data item, or NULL if it doesn’t exist |
| **Return type:** | mixed |

A magic method that allows you to use $this->session->item instead of $\_SESSION['item'], if that’s what you prefer.

It will also return the session ID by calling session\_id() if you try to access $this->session->session\_id.

**\_\_set(*$key*, *$value*)**

|  |  |
| --- | --- |
| **Parameters:** | * **$key** (*string*) – Session item key * **$value** (*mixed*) – Value to assign to the session item key |
| **Returns:** | void |

A magic method that allows you to assign items to $\_SESSION by accessing them as $this->session properties:

$this**->**session**->**foo **=** 'bar';

*// Results in:*

*// $\_SESSION['foo'] = 'bar';*