

TABLE OF CONTENTS

CodeIgniter User Guide Version 2.1.3

[Table of Contents Page](#)[CodeIgniter Home](#) > [User Guide Home](#) > Input Class

Search User Guide

Go

Input Class

The Input Class serves two purposes:

1. It pre-processes global input data for security.
2. It provides some helper functions for fetching input data and pre-processing it.

Note: This class is initialized automatically by the system so there is no need to do it manually.

Security Filtering

The security filtering function is called automatically when a new [controller](#) is invoked. It does the following:

- ➡ If `$config['allow_get_array']` is FALSE(default is TRUE), destroys the global GET array.
- ➡ Destroys all global variables in the event `register_globals` is turned on.
- ➡ Filters the GET/POST/COOKIE array keys, permitting only alpha-numeric (and a few other) characters.
- ➡ Provides XSS (Cross-site Scripting Hacks) filtering. This can be enabled globally, or upon request.
- ➡ Standardizes newline characters to `\n`(In Windows `\r\n`)

XSS Filtering

The Input class has the ability to filter input automatically to prevent cross-site scripting attacks. If you want the filter to run automatically every time it encounters POST or COOKIE data you can enable it by opening your **`application/config/config.php`** file and setting this:

```
$config['global_xss_filtering'] = TRUE;
```

Please refer to the [Security class](#) documentation for information on using XSS Filtering in

your application.

Using POST, COOKIE, or SERVER Data




CodeIgniter comes with three helper functions that let you fetch POST, COOKIE or SERVER items. The main advantage of using the provided functions rather than fetching an item directly (`$_POST['something']`) is that the functions will check to see if the item is set and return false (boolean) if not. This lets you conveniently use data without having to test whether an item exists first. In other words, normally you might do something like this:

```
if ( ! isset($_POST['something']))
{
    $something = FALSE;
}
else
{
    $something = $_POST['something'];
}
```

With CodeIgniter's built in functions you can simply do this:

```
$something = $this->input->post('something');
```

The three functions are:

-  `$this->input->post()`
-  `$this->input->cookie()`
-  `$this->input->server()`

`$this->input->post()`

The first parameter will contain the name of the POST item you are looking for:

```
$this->input->post('some_data');
```

The function returns FALSE (boolean) if the item you are attempting to retrieve does not exist.

The second optional parameter lets you run the data through the XSS filter. It's enabled by setting the second parameter to boolean TRUE;

```
$this->input->post('some_data', TRUE);
```

To return an array of all POST items call without any parameters.

To return all POST items and pass them through the XSS filter set the first parameter NULL while setting the second parameter to boolean;

The function returns FALSE (boolean) if there are no items in the POST.

```
$this->input->post(NULL, TRUE); // returns all POST items with XSS filter  
$this->input->post(); // returns all POST items without XSS filter
```

`$this->input->get()`

This function is identical to the post function, only it fetches get data:

```
$this->input->get('some_data', TRUE);
```

To return an array of all GET items call without any parameters.

To return all GET items and pass them through the XSS filter set the first parameter NULL while setting the second parameter to boolean;

The function returns FALSE (boolean) if there are no items in the GET.

```
$this->input->get(NULL, TRUE); // returns all GET items with XSS filter  
$this->input->get(); // returns all GET items without XSS filtering
```

`$this->input->get_post()`

This function will search through both the post and get streams for data, looking first in post, and then in get:

```
$this->input->get_post('some_data', TRUE);
```

`$this->input->cookie()`

This function is identical to the post function, only it fetches cookie data:

```
$this->input->cookie('some_data', TRUE);
```

`$this->input->server()`

This function is identical to the above functions, only it fetches server data:

```
$this->input->server('some_data');
```

`$this->input->set_cookie()`

Sets a cookie containing the values you specify. There are two ways to pass information to this function so that a cookie can be set: Array Method, and Discrete Parameters:

Array Method

Using this method, an associative array is passed to the first parameter:

```
$cookie = array(
    'name'   => 'The Cookie Name',
    'value'  => 'The Value',
    'expire' => '86500',
    'domain' => '.some-domain.com',
    'path'   => '/',
    'prefix' => 'myprefix_',
    'secure' => TRUE
);

$this->input->set_cookie($cookie);
```

Notes:

Only the name and value are required. To delete a cookie set it with the expiration blank.

The expiration is set in **seconds**, which will be added to the current time. Do not include the time, but rather only the number of seconds from *now* that you wish the cookie to be valid. If the expiration is set to zero the cookie will only last as long as the browser is open.

For site-wide cookies regardless of how your site is requested, add your URL to the **domain** starting with a period, like this: `.your-domain.com`

The path is usually not needed since the function sets a root path.

The prefix is only needed if you need to avoid name collisions with other identically named cookies for your server.

The secure boolean is only needed if you want to make it a secure cookie by setting it to TRUE.

Discrete Parameters

If you prefer, you can set the cookie by passing data using individual parameters:

```
$this->input->set_cookie($name, $value, $expire, $domain, $path, $prefix, $secure);
```

`$this->input->cookie()`

Lets you fetch a cookie. The first parameter will contain the name of the cookie you are looking for (including any prefixes):

```
cookie('some_cookie');
```

The function returns FALSE (boolean) if the item you are attempting to retrieve does not exist.

The second optional parameter lets you run the data through the XSS filter. It's enabled by setting the second parameter to boolean TRUE;

```
cookie('some_cookie', TRUE);
```

`$this->input->ip_address()`

Returns the IP address for the current user. If the IP address is not valid, the function will return an IP of: 0.0.0.0

```
echo $this->input->ip_address();
```

`$this->input->valid_ip($ip)`

Takes an IP address as input and returns TRUE or FALSE (boolean) if it is valid or not. Note: The `$this->input->ip_address()` function above validates the IP automatically.

```
if ( ! $this->input->valid_ip($ip))
{
    echo 'Not Valid';
}
else
{
    echo 'Valid';
}
```

Accepts an optional second string parameter of "IPv4" or "IPv6" to specify an IP format. The default checks for both formats.

`$this->input->user_agent()`

Returns the user agent (web browser) being used by the current user. Returns FALSE if it's not available.

```
echo $this->input->user_agent();
```

See the [User Agent Class](#) for methods which extract information from the user agent string.

`$this->input->request_headers()`

Useful if running in a non-Apache environment where [apache_request_headers\(\)](#) will not be supported. Returns an array of headers.

```
$headers = $this->input->request_headers();
```

`$this->input->get_request_header();`

Returns a single member of the request headers array.

```
$this->input->get_request_header('some-header', TRUE);
```

`$this->input->is_ajax_request()`

Checks to see if the **HTTP_X_REQUESTED_WITH** server header has been set, and returns a boolean response.

`$this->input->is_cli_request()`

Checks to see if the **STDIN** constant is set, which is a failsafe way to see if PHP is being run on the command line.

```
$this->input->is_cli_request()
```

Previous Topic: [Image Manipulation Class](#) · [Top of Page](#) · [User Guide Home](#) · Next Topic: [Loader Class](#)

[CodeIgniter](#) · Copyright © 2006 – 2012 · [EllisLab, Inc.](#)