How to Get Custom Config Values in CodeIgniter 3 and 4

TechRounder PDF Edition

Live article: https://www.techrounder.com/codeigniter/get-custom-added-config-value-in-codeigniter/

By Vipin PG | Published May 1, 2013 | Updated March 14, 2026 | Format: Guide | 4 min read

Quick answer

Working with custom configuration values in CodeIgniter is, at its core, pretty simple: define a value once, then pull it in wherever your app needs it. But here's where a lot of developers hit a wall - the right approach depends entirely on whether you're running CodeIgniter 3 or CodeIgniter 4.

Working with custom configuration values in CodeIgniter is, at its core, pretty simple: define a value once, then pull it in wherever your app needs it. But here's where a lot of developers hit a wall - the right approach depends entirely on whether you're running CodeIgniter 3 or CodeIgniter 4. The older array-based method is still perfectly valid in CI3, but CI4 moved to config classes with public properties - a shift covered in both the CI3 Config Class docs and the CI4 configuration guide.

Getting a Custom Config Value in CodeIgniter 3

If your project is still on CI3, you're in good shape - that original approach is valid. CI3 stores configuration values in the '$config' array, typically inside application/config/config.php. You can also create separate config files and load them on demand, just as the official config docs describe.

1. Add custom config values

```
$config['config_variable1'] = 'Config value 1';
$config['config_variable2'] = 'Config value 2';
$config['config_variable3'] = 'Config value 3';
```

You can drop these directly into application/config/config.php if they're global site-wide values. Personally, I prefer putting feature-specific settings in a dedicated file like application/config/custom.php. It keeps your main config file from becoming a dumping ground that nobody wants to scroll through.

2. Retrieve the config values

```
$config_variable1 = $this->config->item('config_variable1');
$config_variable2 = $this->config->item('config_variable2');
$config_variable3 = $this->config->item('config_variable3');
```

If you stored the values in a separate file like custom.php, load that file first:

```
$this->config->load('custom');
$config_variable1 = $this->config->item('config_variable1');
$config_variable2 = $this->config->item('config_variable2');
$config_variable3 = $this->config->item('config_variable3');
```

That's the clean CI3 way to handle it. Readable, predictable, and still perfectly appropriate for legacy projects that have no reason to change.

Why Your Original Example Needs an Update

Your snippet works fine for CodeIgniter 3, but it’s not the right pattern for CodeIgniter 4. And that distinction matters because CI4 is essentially a ground-up rewrite. Configuration is handled in a fundamentally different way - the framework now uses config classes inside app/Config with typed public properties rather than the old array setup. The official upgrade guide is explicit about this: calls like ‘$this->config->item(’item_name’)‘ should be replaced with ‘config(’MyConfig’)->item_name‘.

Storing Custom Config Values in CodeIgniter 4

In CI4, the cleaner approach is to create your own config class rather than piling everything into one sprawling global file.

1. Create a config file in app/Config

Create a file called app/Config/Custom.php:

```
<?php
namespace Config;
use CodeIgniter\Config\BaseConfig;
class Custom extends BaseConfig
{
public string $config_variable1 = ’Config value 1’;
public string $config_variable2 = ’Config value 2’;
public string $config_variable3 = ’Config value 3’;
}
```

This follows the current CI4 structure outlined in the configuration documentation.

2. Retrieve the values anywhere in your app

```
$config = config(’Custom’);
$config_variable1 = $config->config_variable1;
$config_variable2 = $config->config_variable2;
$config_variable3 = $config->config_variable3;
```

Think of this as the modern equivalent of CI3’s ‘item()‘ approach. It’s cleaner, supports type declarations if you want them, and scales much better as your application grows.

Using Environment Values in CodeIgniter 4

Here’s where things get genuinely practical. If a value changes between local development, staging, and production - things like API keys, feature flags, service URLs, or credentials - don’t hardcode it unless you absolutely have to. CI4 is built with environment-based configuration in mind, and the official starter project specifically recommends copying env to .env and tailoring it for your application, as explained in the app starter setup.

A simple .env entry might look like this:

```
custom.config_variable1 = "Production Value 1"
custom.config_variable2 = "Production Value 2"
```

Your config class then provides sensible defaults, and the environment file overrides them per deployment. That approach is a lot safer than manually editing live PHP config files every time something needs to change in production.

Separating App Settings From Runtime Settings

Not every value belongs in a config file - and this is something I see developers get wrong fairly often. Static values like app names, default paths, labels, or feature toggles are solid candidates for config. But if something needs to change from an admin panel, be adjusted per user, or get updated without a deployment, it probably belongs in the database instead. The official Settings library package for CI4 exists for exactly that use case.

I've watched projects mix these two concepts together, and it almost always causes headaches down the road. Config files are for developer-owned settings. They're not a substitute for editable application settings that need to change at runtime.

CI3 vs CI4: A Quick Side-by-Side

- CodeIgniter 3: Store values in '$config['key']' and read them with '$this->config->item('key')' .
- CodeIgniter 4: Store values as public properties in a config class and read them with 'config('ClassName')->property' .
- Environment-specific values: Prefer .env in CI4 for anything that differs between server environments or deployment stages.

Updated Examples for Both Versions

CodeIgniter 3

```
$config['site_name'] = 'My Website';
$config['support_email'] = 'support@example.com';
// Get value
$site_name = $this->config->item('site_name');
$support_email = $this->config->item('support_email');
```

CodeIgniter 4

```
<?php
namespace Config;
use CodeIgniter\Config\BaseConfig;
class Site extends BaseConfig
{
public string $siteName = 'My Website';
public string $supportEmail = 'support@example.com';
}
$config = config('Site');
$siteName = $config->siteName;
$supportEmail = $config->supportEmail;
```

The Bottom Line

If your project is running on CodeIgniter 3, your original method is still fine - no need to change a thing. But if you're writing a tutorial today, or maintaining documentation aimed at developers who might be on CI4, show both approaches. Otherwise, someone copies a CI3 snippet into a CI4 app and spends an hour wondering why nothing works.

That's really the whole point here. The technique hasn't gone anywhere. It just grew up a bit.

References

1. codeigniter.com - userguide3 / libraries - https://www.codeigniter.com/userguide3/libraries/config.html
2. codeigniter.com - user_guide / general - https://codeigniter.com/user_guide/general/configuration.html
3. codeigniter4.github.io - userguide / installation - https://codeigniter4.github.io/userguide/installation/upgrade_configuration.html
4. github.com - codeigniter4 / appstarter - https://github.com/codeigniter4/appstarter
5. github.com - codeigniter4 / settings - https://github.com/codeigniter4/settings