live search...

One Runner for Multiple Applications
Namespaces
Extension
Events
Enabling Extension
Configuring Extension
Custom Commands
Group Objects
Custom Reporters
Installation Templates
Conclusion

Customization

In this chapter we will explain how you can extend and customize the file structure and test execution routines.

One Runner for Multiple Applications

If your project consists of several applications (frontend, admin, api) or you are using the Symfony framework with its bundles, you may be interested in having all tests for all applications (bundles) executed in one runner. In this case you will get one report that covers the whole project.

Place the `codeception.yml` file into the root folder of your project and specify the paths to the other `codeception.yml` configurations that you want to include:

```
include:
  - frontend/src/*Bundle
  - admin
  - api/rest
paths:
  output: _output
settings:
  colors: false
```

You should also specify the path to the `log` directory, where the reports and logs will be saved.

> Wildcards (*) can be used to specify multiple directories at once.

Namespaces

To avoid naming conflicts between Actor classes and Helper classes, they should be separated into namespaces. To create test suites with namespaces you can add `--namespace` option to the bootstrap command:

```
php codecept bootstrap --namespace frontend
```

This will bootstrap a new project with the `namespace: frontend` parameter in the `codeception.yml` file. Helpers will be in the `frontend\Codeception\Module` namespace and Actor classes will be in the `frontend` namespace.

Once each of your applications (bundles) has its own namespace and different Helper or Actor classes, you can execute all the tests in a single runner. Run the Codeception tests as usual, using the meta-config we created earlier:

```
php codecept run
```

This will launch the test suites for all three applications and merge the reports from all of them. This is very useful when you run your tests on a Continuous Integration server and you want to get a single report in JUnit and HTML format. The code coverage report will be merged too.

If you want to run a specific suite from the application you can execute:

Follow @codeception

```
codecept run unit -c frontend
```

Where `unit` is the name of suite and the `-c` option specifies the path to the `codeception.yml` configuration file to use. In this example we will assume that there is `frontend/codeception.yml` configuration file and that we will execute the unit tests for only that app.

Extension

Codeception has limited capabilities to extend its core features. Extensions are not supposed to override current functionality, but can be useful if you are an experienced developer and you want to hook into the testing flow.

By default, one `RunFailed` Extension is already enabled in your global `codeception.yml`. It allows you to rerun failed tests by using the `-g failed` option:

```
codecept run -g failed
```

Codeception comes with bundled extensions located in `ext` directory. For instance, you can enable the Logger extension to log the test execution with Monolog:

```
extensions:
    enabled:
        - Codeception\Extension\RunFailed # default extension
        - Codeception\Extension\Logger: # enabled extension
            max_files: 5 # logger configuration
```

But what are extensions, anyway? Basically speaking, extensions are nothing more than event listeners based on the Symfony Event Dispatcher (http://symfony.com/doc/current/components/event_dispatcher/introduction.html) component.

Events

Here are the events and event classes. The events are listed in the order in which they happen during execution. All listed events are available as constants in `Codeception\Events` class.

Event	When?	Triggered by
`suite.before`	Before suite is executed	Suite, Settings (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/SuiteEvent.php)
`test.start`	Before test is executed	Test (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/TestEvent.php)
`test.before`	At the very beginning of test execution	Codeception Test (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/TestEvent.php)
`step.before`	Before step	Step (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/StepEvent.php)
`step.after`	After step	Step (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/StepEvent.php)
`step.fail`	After failed step	Step (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/StepEvent.php)
`test.fail`	After failed test	Test, Fail (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/FailEvent.php)
`test.error`	After test ended with error	Test, Fail (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/FailEvent.php)
`test.incomplete`	After executing incomplete test	Test, Fail (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/FailEvent.php)

Follow @codeception

Event	When?	Triggered by
`test.skipped`	After executing skipped test	Test, Fail (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/FailEvent.php)
`test.success`	After executing successful test	Test (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/TestEvent.php)
`test.after`	At the end of test execution	Codeception Test (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/TestEvent.php)
`test.end`	After test execution	Test (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/TestEvent.php)
`suite.after`	After suite was executed	Suite, Result, Settings (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/SuiteEvent.php)
`test.fail.print`	When test fails are printed	Test, Fail (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/FailEvent.php)
`result.print.after`	After result was printed	Result, Printer (https://github.com/Codeception/Codeception/blob/master/src/Codeception/Event/PrintResultEvent.php)

There may be some confusion between `test.start` / `test.before` and `test.after` / `test.end` . The start and end events are triggered by PHPUnit, but the before and after events are triggered by Codeception. Thus, when you are using classical PHPUnit tests (extended from `PHPUnit\Framework\TestCase`), the before/after events won't be triggered for them. During the `test.before` event you can mark a test as skipped or incomplete, which is not possible in `test.start` . You can learn more from Codeception internal event listeners (https://github.com/Codeception/Codeception/tree/master/src/Codeception/Subscriber).

The extension class itself is inherited from `Codeception\Extension` :

Follow @codeception

```
<?php
use \Codeception\Events;

class MyCustomExtension extends \Codeception\Extension
{
    // list events to listen to
    // Codeception\Events constants used to set the event

    public static $events = array(
        Events::SUITE_AFTER  => 'afterSuite',
        Events::SUITE_BEFORE => 'beforeTest',
        Events::STEP_BEFORE => 'beforeStep',
        Events::TEST_FAIL => 'testFailed',
        Events::RESULT_PRINT_AFTER => 'print',
    );

    // methods that handle events

    public function afterSuite(\Codeception\Event\SuiteEvent $e) {}

    public function beforeTest(\Codeception\Event\TestEvent $e) {}

    public function beforeStep(\Codeception\Event\StepEvent $e) {}

    public function testFailed(\Codeception\Event\FailEvent $e) {}

    public function print(\Codeception\Event\PrintResultEvent $e) {}
}
```

By implementing event handling methods you can listen for events and even update passed objects. Extensions have some basic methods you can use:

- `write` - prints to the screen
- `writeln` - prints to the screen with a new-line character at the end
- `getModule` - allows you to access a module
- `hasModule` - checks if a module is enabled
- `getModuleNames` - list all enabled modules
- `_reconfigure` - can be implemented instead of overriding the constructor

Enabling Extension

Once you've implemented a simple extension class, you can require it in `tests/_bootstrap.php`, load it with Composer's autoloader defined in `composer.json`, or store the class inside `tests/_support` dir.

You can then enable it in `codeception.yml`

```
extensions:
    enabled: [MyCustomExtension]
```

Extensions can also be enabled per suite inside suite configs (like `acceptance.suite.yml`) and for a specific environment.

To enable extension dynamically, execute the `run` command with `--ext` option. Provide a class name as a parameter:

```
codecept run --ext MyCustomExtension
codecept run --ext "\My\Extension"
```

If a class is in a `Codeception\Extension` namespace you can skip it and provide only a shortname. So Recorder extension can be started like this:

```
codecept run --ext Recorder
```

Configuring Extension

Follow @codeception

In the extension, you can access the currently passed options via the `options` property. You also can access the global configuration via the `\Codeception\Configuration::config()` method. If you want to have custom options for your extension, you can pass them in the `codeception.yml` file:

```
extensions:
    enabled: [MyCustomExtension]
    config:
        MyCustomExtension:
            param: value
```

The passed in configuration is accessible via the `config` property: `$this->config['param']`.

Check out a very basic extension Notifier (https://github.com/Codeception/Notifier).

Custom Commands

You can add your own commands to Codeception.

Your custom commands have to implement the interface Codeception\CustomCommandInterface, because there has to be a function to get the name of the command.

You have to register your command in the file `codeception.yml`:

```
extensions:
    commands: [Project\Command\MyCustomCommand]
```

If you want to activate the Command globally, because you are using more then one `codeception.yml` file, you have to register your command in the `codeception.dist.yml` in the root folder of your project.

Please see the complete example (https://github.com/Codeception/Codeception/blob/2.3/tests/data/register_command/examples/MyCustomCommand.php)

Group Objects

Group Objects are extensions listening for the events of tests belonging to a specific group. When a test is added to a group:

```
<?php
/**
 * @group admin
 */
public function testAdminCreatingNewBlogPost(\AcceptanceTester $I)
{
}
```

This test will trigger the following events:

- `test.before.admin`
- `step.before.admin`
- `step.after.admin`
- `test.success.admin`
- `test.fail.admin`
- `test.after.admin`

A group object is built to listen for these events. It is useful when an additional setup is required for some of your tests. Let’s say you want to load fixtures for tests that belong to the `admin` group:

Follow @codeception

```
<?php
namespace Group;

class Admin extends \Codeception\GroupObject
{
    public static $group = 'admin';

    public function _before(\Codeception\Event\TestEvent $e)
    {
        $this->writeln('inserting additional admin users...');

        $db = $this->getModule('Db');
        $db->haveInDatabase('users', array('name' => 'bill', 'role' => 'admin'));
        $db->haveInDatabase('users', array('name' => 'john', 'role' => 'admin'));
        $db->haveInDatabase('users', array('name' => 'mark', 'role' => 'banned'));
    }

    public function _after(\Codeception\Event\TestEvent $e)
    {
        $this->writeln('cleaning up admin users...');
        // ...
    }
}
```

GroupObjects can also be used to update the module configuration before running a test. For instance, for `nocleanup` group we prevent Doctrine2 module from wrapping test into transaction:

```
<?php
    public static $group = 'nocleanup';

    public function _before(\Codeception\Event\TestEvent $e)
    {
        $this->getModule('Doctrine2')->_reconfigure(['cleanup' => false]);
    }
```

A group class can be created with `php codecept generate:group groupname` command. Group classes will be stored in the `tests/_support/Group` directory.

A group class can be enabled just like you enable an extension class. In the file `codeception.yml` :

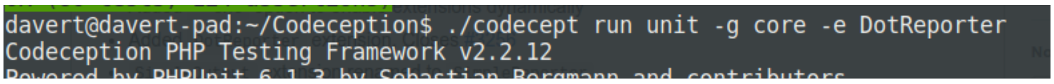
```
extensions:
    enabled: [Group\Admin]
```

Now the Admin group class will listen for all events of tests that belong to the `admin` group.

Custom Reporters

Alternative reporters can be implemented as extension. There are DotReporter (http://codeception.com/extensions#DotReporter) and SimpleReporter (http://codeception.com/extensions#SimpleReporter) extensions included. Use them to change output or use them as an example to build your own reporter. They can be easily enabled with `--ext` option

```
codecept run --ext DotReporter
```



```
davert@davert-pad:~/Codeception$ ./codecept run unit -g core -e DotReporter
Codeception PHP Testing Framework v2.2.12
Powered by PHPUnit 6.1.3 by Sebastian Bergmann and contributors
```

If you want to use it as default reporter enable it in `codeception.yml` .

Follow @codeception

But what if you need to change the output format of the XML or JSON results triggered with the `--xml` or `--json` options? Codeception uses PHPUnit printers and overrides them. If you need to customize one of the standard reporters you can override them too. If you are thinking on implementing your own reporter you should add a `reporters` section to `codeception.yml` and override one of the standard printer classes with one of your own:

```
reporters:
    xml: Codeception\PHPUnit\Log\JUnit
    html: Codeception\PHPUnit\ResultPrinter\HTML
    report: Codeception\PHPUnit\ResultPrinter\Report
```

All PHPUnit printers implement the PHPUnit_Framework_TestListener (https://phpunit.de/manual/current/en/extending-phpunit.html#extending-phpunit.PHPUnit_Framework_TestListener) interface. It is recommended to read the code of the original reporter before overriding it.

Installation Templates

Codeception setup can be customized for the needs of your application. If you build a distributable application and you have a personalized configuration you can build an Installation template which will help your users to start testing on their projects.

Codeception has built-in installation templates for

- Acceptance tests (https://github.com/Codeception/Codeception/blob/2.3/src/Codeception/Template/Acceptance.php)
- Unit tests (https://github.com/Codeception/Codeception/blob/2.3/src/Codeception/Template/Unit.php)
- REST API tests (https://github.com/Codeception/Codeception/blob/2.3/src/Codeception/Template/Api.php)

They can be executed with `init` command:

```
codecept init Acceptance
```

To init tests in specific folder use `--path` option:

```
codecept init Acceptance --path acceptance_tests
```

You will be asked several questions and then config files will be generated and all necessary directories will be created. Learn from the examples above to build a custom Installation Template. Here are the basic rules you should follow:

- Templates should be inherited from `Codeception\InitTemplate` (http://codeception.com/docs/reference/InitTemplate) class and implement `setup` method.
- Template class should be placed in `Codeception\Template` namespace so Codeception could locate them by class name
- Use methods like `say`, `saySuccess`, `sayWarning`, `sayError`, `ask`, to interact with a user.
- Use `createDirectoryFor`, `createEmptyDirectory` methods to create directories
- Use `createHelper`, `createActor` methods to create helpers and actors.
- Use Codeception generators (https://github.com/Codeception/Codeception/tree/2.3/src/Codeception/Lib/Generator) to create other support classes.

Conclusion

Each feature mentioned above may help dramatically when using Codeception to automate the testing of large projects, although some features may require advanced knowledge of PHP. There is no “best practice” or “use cases” when we talk about groups, extensions, or other powerful features of Codeception. If you see you have a problem that can be solved using these extensions, then give them a try.

- **Next Chapter: Data > (/docs/09-Data)**
- **Previous Chapter: < BDD (/docs/07-BDD)**

(https://www.jetbrains.com/phpstorm/) (https://www.rebilly.com/)

(https://github.com/codeception/codeception) (https://twitter.com/codeception) (http://www.facebook.com/pages/Codeception/288959711204412)

- Installation (/install)
- Credits (/credits)
- Releases (/changelog)
- Commercial Services (http://sdclabs.com/codeception?utm_source=codeception.com&utm_medium=bottom_menu&utm_term=link&utm_campaign=reference)
- License (https://github.com/Codeception/Codeception/blob/master/LICENSE)

Codeception Family

(http://robo.li)

Robo

(http://robo.li)

Modern PHP **Task Runner**. Allows to declare tasks with zero configuration in pure PHP.

(http://codecept.io)

CodeceptJS

(http://codecept.io)

Codeception for **NodeJS**. Write acceptance tests in ES6 and execute in webdriverio, Selenium WebDriver, and Protractor.

Follow @codeception