Behat Code Coverage Documentation

Release version 5

Doug Wright

Sep 26, 2021

CONTENTS

Behat Code Coverage is an extension for Behat that can generate code coverage reports when testing a PHP application. The extension wraps the same php-code-coverage library that is used by PHPUnit thereby producing reports that are familiar to developers and interoperable with other tooling.

Note: The authors of Behat pedantically, but correctly, point out that `.feature` files are not strictly speaking tests even though when constructed properly the scenarios described in them should cover both the happy and sad paths in an application.

Technically, Behat is a *scenario* runner, not a *test* runner. The scenarios might be run by hand. Or the application under scrutiny might not be a local PHP application, it might be running on a remote server and/or the software might not even be written in PHP. Additionally by the very nature of needing to invoke the entire application to perform each scenario, it would be very hard to construct a set of scenarios that cover all possible codepaths in an application. Something like PHPUnit is much better to use here if your goal is comprehensive code coverage as you can unit test each component in isolation.

However, out in the real world we don't normally draw a distinction between the `.feature` files as a standalone concept and the `Context`s that implement them - we simply refer to Behat testing. We also tend to use Behat when the application being tested is written in PHP. And as with any test suite, it's nice to know how much of your application code is covered by a test suite. What you do with that information is up to you :)

CHAPTER

ONE

LICENSE

Behat Code Coverage is licensed under the BSD-2-Clause License. See license.txt for full details.

1.1 Installation

The recommended way to install PHPCoord is to use Composer. From the command line simply execute the following to add `dvdoug/behat-code-coverage` to the `require-dev` section of your project's `composer.json` file. Composer will automatically take care of downloading the source and configuring an autoloader:

```
composer require --dev dvdoug/behat-code-coverage
```

The code is also available to download from GitHub

After installation, in your project's `behat.yml` or `behat.yml.dist`, add `DVDoug\Behat\CodeCoverage\Extension` into the `extensions` key under `default` to enable it.

The below example represents a typical configuration you may wish to use as a starting point. Each option is more fully explained in the next section.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            filter:
                include:
                    directories:
                        'src': ~
            reports:
                html:
                    target: build/coverage-behat
                text:
                    showColors: true
```

1.1.1 Compatibility

Since both PHPUnit and Behat Code Coverage both depend upon php-code-coverage, it is not always possible to run an older version of PHPUnit alongside the newest version of this library or vice versa since they both need to be compatible with the same underlying version of the code coverage component. Where reasonably possible Behat Code Coverage maintains support for multiple versions of the coverage library to ease the upgrade path, however where a new version is significantly incompatible with older ones or where a compelling new feature exists the minimum supported version may be raised. Fixes for any subsequently discovered bugs will be backported as appropriate.

Behat Code Coverage	Behat	php-code-coverage	PHPUnit	Notes
5.3	3.8.1+	9.2+, 10	9.5.5+, 10	Not yet released
5.2	3.5+	9.2+	9.4+	Cobertura format support
5.1	3.5+	9.1+	9.3.4+	Caching between runs
5.0	3.5+	6, 7, 8, 9	7, 8, 9	

1.2 Configuration

To do anything useful, once installed, you should configure Behat Code Coverage according to your project needs.

Each of the settings described below has a direct equivalent in PHPUnit, if your codebase uses both testing frameworks you may wish to ensure the settings are aligned between the two tools.

Settings are configured in your project's `behat.yml` or `behat.yml.dist`.

1.2.1 Driver

In order to generate code coverage data, you must have a code coverage driver installed - using either Xdebug or the PCOV extensions is recommended, although you can also make use of PHP's built in PHPDBG. Drivers are detected at runtime, they do not need to be configured.

1.2.2 Filter

A filter is the mechanism to include (or exclude) directories or files from the generated coverage reports. Technically this is an optional configuration setting, but practically speaking most projects will want to configure this to report only on their own source files rather than e.g. test files or vendor files.

For directories, list them by **key**. Optionally, you can specify a prefix or suffix:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            filter:
                include:
                    directories:
                        'src':
                            suffix: 'Controller.php'
                        'src/Repository':
                            prefix: 'API'
                exclude:
                    directories:
                        'tests/data' ~
```

For files, list them by **name**:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            filter:
                include:
                    files:
                        - 'bootstrap.php'
                        - 'index.php'
                exclude:
                    files:
                        - 'src/ExcludeMe.php'
                        - 'src/ExcludeMeToo.php'
```

Filters have two additional settings, to control how files that have not been covered should be handled in the report. By default uncovered files are **included** but not **processed**.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            filter:
                includeUncoveredFiles: true # defaults to true
                processUncoveredFiles: false # defaults to false
```

- *Included* means that the file is incorporated into the reporting, showing up as 0% covered. This allows you to see files that you haven't written any tests for yet. If you choose not to include uncovered files, then only files that have been loaded by your test suite (whether directly or indirectly) will be shown on the report.
- Uncovered files, by definition, will not been loaded into the PHP runtime environment during the execution of the test suite. This means that they have not been analysed by the coverage driver for detection of things like executable vs non-executable lines, dead code detection or calculation of branches and paths. By opting uncovered files into *processing*, the files will be loaded via an `include()` call and passed through the driver in exactly the same way as covered files are. However, not all files can be safely loaded in this way, for instance `include()`ing a script may have runtime side-effects. Because Behat Code Coverage cannot know the structure of your codebase and which files are safe to `include()` and which are not, by default uncovered file processing is disabled for safety reasons.

Note: The `processUncoveredFiles` setting has been removed in php-code-coverage v10, configuring it here is deprecated

---

1.2.3 Reports

Behat Code Coverage allows you to generate reports in any/all of the report formats provided by `php-code-coverage`. Reports are configured under the `reports` key.

Text

The text report is the easiest way to get started, it simply outputs the report results to the screen after each test run. It is configured by setting the `text` key. The default values are outlined below:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                text:
                    showColors: false
                    showOnlySummary: false
                    showUncoveredFiles: false
                    lowUpperBound: 50
                    highLowerBound: 90
```

- If `showColors` is `true`, the results will be output in a colour-coded format, red for low coverage, amber/yellow for a medium amount of coverage and green for high coverage. If desired, the thresholds for each colour can be configured via `lowUpperBound` and `highLowerBound`.
- If you have a large codebase, outputting the coverage data for each and every individual file to the CLI may be too be noisy to be helpful. If so, you can set `showOnlySummary` to `true` which will output only a project-level overview.
- By default, when showing data for individual files the text report does not show data for uncovered files (even if data collection enabled under `filter`). This can be changed if desired by setting `showUncoveredFiles` to true. Note that this is a report-specific display option only, choosing not to show uncovered files in the text report has no impact on whether they are included in other types of report.

HTML

The HTML report is the most common format of report. As well as a summary report for the project providing high-level data, it also includes a detailed overview of each file showing the coverage on a function by function, line by line basis. It is configured by setting the `html` key. The default values are outlined below:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                html:
                    target: <directory>  # no default value, you must specify
                    lowUpperBound: 50
                    highLowerBound: 90
```

- The mandatory `target` key specifies the target directory to place the report files.
- The HTML report is heavily colour-coded format using red for low coverage, amber/yellow for a medium amount of coverage and green for high coverage. If desired, the thresholds for each colour can be configured via `lowUpperBound` and `highLowerBound`.

Clover

Originating from the Java world, Clover-format reports are a standard way of exchanging coverage data between tools. It is configured by setting the `clover` key. The default values are outlined below:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                clover:
                    target: <file>  # no default value, you must specify
                    name: ''
```

- The mandatory `target` key specifies the destination filename to use for the report. The reports are in XML format, an `.xml` file extension is suggested.
- Optionally, you can configure the name of your project via the `name` key.

Cobertura

Also originating from the Java world, Cobertura-format reports are becoming a standard way of exchanging coverage data between tools. It is configured by setting the `cobertura` key. The default values are outlined below:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                cobertura:
                    target: <file>  # no default value, you must specify
                    name: ''
```

- The mandatory `target` key specifies the destination filename to use for the report. The reports are in XML format, an `.xml` file extension is suggested.
- Optionally, you can configure the name of your project via the `name` key.

Crap4j

An older, discontinued tool from the Java world. You can generate Crap4j-compatible reports by setting the `crap4j` key. The default values are outlined below:

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                crap4j:
                    target: <file>  # no default value, you must specify
                    name: ''
```

- The mandatory `target` key specifies the destination filename to use for the report. The reports are in XML format, an `.xml` file extension is suggested.
- Optionally, you can configure the name of your project via the `name` key.

PHP ".cov"

A PHP or ".cov" report is a raw serialisation of internal php-code-coverage state, allowing for full fidelity of data to be preserved. They can be manipulated by the phpcov tool, for instance to combine reports from multiple testing tools. You can generate PHP ".cov" reports by setting the `php` key.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                php:
                    target: <file>  # no default value, you must specify
```

- The mandatory `target` key specifies the destination filename to use for the report. The reports are actually PHP, but a `.cov` file extension is customary.

PHPUnit XML

You can generate PHPUnit XML reports by setting the `xml` key.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            reports:
                xml:
                    target: <directory>  # no default value, you must specify
```

- The mandatory `target` key specifies the target directory to use for the report.

1.2.4 Branch and path coverage

When using Xdebug as a coverage driver, it has the ability to generate branch and path coverage data as well as the traditional line-based data. More information on this topic is available at https://doug.codes/php-code-coverage.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            branchAndPathCoverage: true
```

By default `branchAndPathCoverage` is true when running under Xdebug, false otherwise.

1.2.5 Caching

Since analysing source code files to generate coverage reports is computationally expensive, Behat Code Coverage makes use of a cache to ameliorate this.

```
default:
    extensions:
        DVDoug\Behat\CodeCoverage\Extension:
            cache: <directory>
```

The default `cache` directory is `sys_get_temp_dir() . '/behat-code-coverage-cache'`. You may wish to relocate this this to be inside your project workspace.