Hi, thank you for your interest in contributing to PHPCompatibility! We look forward to working with you.

Reporting bugs

--------------

Before reporting a bug, you should check what sniff an error is coming from.

Running `phpcs` with the `-s` flag will show the name of the sniff with each error.

Bug reports containing a minimal code sample which can be used to reproduce the issue are highly appreciated as those are most easily actionable.

Requesting features

-------------------

The PHPCompatibility standard only concerns itself with cross-version PHP compatibility of code.

When requesting a new feature, please add a link to a relevant page in the [PHP Manual](http://php.net/manual/en/) / PHP Changelog / [PHP RFC website](https://wiki.php.net/rfc) which illustrates the feature you are requesting.

Pull requests

-------------

Contributions in the form of pull requests are very welcome.

To start contributing, fork the repository, create a new branch in your fork, make your intended changes and pull the branch against the `develop` branch of this repository.

Please make sure that your pull request contains unit tests covering what's being addressed by it.

\* All code should be compatible with PHPCS >= 2.6.0 and PHPCS >= 3.1.0.

\* All code should be compatible with PHP 5.4 to PHP nightly.

\* All code should comply with the PHPCompatibility coding standards.

The [ruleset used by PHPCompatibility](https://github.com/PHPCompatibility/PHPCompatibility/blob/master/phpcs.xml.dist) is largely based on PSR-2 with minor variations and some additional checks for documentation and such.

### Typical sources of information about changes in PHP

\* The [PHP RFC wiki](https://wiki.php.net/rfc)

\* The [UPGRADING](https://github.com/php/php-src/blob/master/UPGRADING) document of each release

\* The [NEWS](https://github.com/php/php-src/blob/master/NEWS) document of each release

\* The [Migrating from PHP x.x.x to PHP x.x.x section](https://www.php.net/manual/en/appendices.php) in the manual for each release (once published)

\* The [Changelog](https://www.php.net/manual/en/doc.changelog.php) in the manual

\* The [PHP manual](https://www.php.net/manual/en/index.php) in general

\* The [PHP source code](https://github.com/php/php-src) in general

### Framework/CMS specific rulesets

Since mid 2018, framework/CMS/polyfill specific rulesets will be accepted to be hosted in separate repositories in the PHPCompatibility organisation. If you are interested in adding a ruleset for a framework/CMS/PHP polyfill library, you can request a repository for it by [opening an issue](https://github.com/PHPCompatibility/PHPCompatibility/issues/new) in this repo.

#### Guidelines for framework/CMS specific rulesets

A framework/CMS/polyfill specific ruleset will generally contain `<exclude ...>` directives for backfills/polyfills provided by the framework/CMS/polyfill to prevent false positives.

> A backfill is a function/constant/class (etc) which has been added to PHP in a later version than the minimum supported version of the framework/CMS and for which a function/constant/class of the same name is included in the framework/CMS when a PHP version is detected in which the function/constant/class did not yet exist.

These rulesets will not be actively maintained by the maintainers of PHPCompatibility.

The communities behind these PHP frameworks/CMSes/polyfill libraries are strongly encouraged to maintain these rulesets and pull requests with updates will be accepted gladly.

\*\*Note:\*\*

\* It is recommended to include a link to the framework/CMS/polyfill source file where the backfill is declared when sending in a pull request adding a new backfill for one of these rulesets.

\* If the backfills provided by different major versions of frameworks/CMSes/polyfill libraries are signficantly different, separate rulesets for the relevant major versions of frameworks/CMSes/polyfill libraries will be accepted.

\* Framework/CMS specific ruleset should \*\*\_not\_\*\* contain a `<config name="testVersion" value="..."/>` directive.

While a framework/CMS/polyfill may have a certain minimum PHP version, projects based on or using the framework/CMS/polyfill might have a different (higher) minimum PHP version.

As support for overruling a `<config>` directive [is patchy](https://github.com/squizlabs/PHP\_CodeSniffer/issues/1821), it should be recommended to set the desired `testVersion` either from the command line or in a project-specific custom ruleset.

Naming conventions and repository structure

-----------------------

### Regarding sniff names:

\* Per PHPCS convention, sniff files and class names have the `Sniff` suffix.

\* The name of sniffs relating to new PHP features should start with `New`.

\* The name of sniffs relating to deprecated or removed PHP features should start with `Removed` - as everything which has been deprecated is slated for removal in a later PHP version -.

\* Sniffs in the `ParameterValue` category which relate to a specific function, should have the function name and parameter name in the sniff name, like so: `NewFunctionnameParameternameSniff`.

\* All sniffs should be placed in a category which relates to the type of PHP construct the sniff is checking for.

Most existing categories will be clear cut, however, to prevent confusion, here is some additional information about some closely related categories:

- `FunctionDeclarations` should be used for sniffs relating to the actual function declaration statement, i.e. `function functionName($param) {`.

- `FunctionNameRestrictions` can be regarded as a sub-category of `FunctionDeclaration` in so far as that sniffs which check for invalid function names in certain contexts, should be placed here.

- `FunctionUse` should be used for sniffs inspecting calls to certain PHP functions.

- `ParameterValues` can be regarded as a sub-category of `FunctionUse` in so far as that sniffs which check for calls to specific PHP functions and subsequently inspect the \_value\_ of parameters passed to that function call, should be placed here.

Additionally:

- The `Miscellaneous` category should be avoided if at all possible and should only be used as a last resort.

### About the unit tests:

\* Unit test files should be named the same as the sniff, replacing the `Sniff` suffix with `UnitTest`.

\* The test case file for the unit tests should be named the same as the unit test file, but should use the `.inc` file extension.

If several test case files are needed to test a sniff, the convention is to number the files starting with `1`, i.e. `SniffNameUnitTest.1.inc`, `SniffNameUnitTest.2.inc` etc.

\* Test case files should be placed in the same directory as the unit test file.

Running the Sniff Tests

-----------------------

All the sniffs are fully tested with PHPUnit tests and have `@group` annotations matching their categorization to allow for running subsets of the unit tests more easily.

In order to run the tests on the sniffs, the following installation steps are required.

1. Install PHP CodeSniffer and PHP Compatibility by following the instructions in the Readme for either [installing with Composer](https://github.com/PHPCompatibility/PHPCompatibility/blob/master/README.md#installation-in-a-composer-project-method-1) or via a [Git Checkout to an arbitrary directory](https://github.com/PHPCompatibility/PHPCompatibility/blob/master/README.md#installation-via-a-git-check-out-to-an-arbitrary-directory-method-2).

If you install using Composer, make sure you run `composer install --prefer-source` to get access to the unit tests and other development related files.

\*\*Pro-tip\*\*: If you develop regularly for the PHPCompatibility standard, it may be preferable to use a git clone based install of PHP CodeSniffer to allow you to easily test sniffs with different PHP CodeSniffer versions by switching between tags.

2. If you used Composer, PHPUnit should be installed automatically and you are done.

Run the tests by running `phpunit` in the root directory of PHPCompatibility.

It will read the `phpunit.xml.dist` file and execute the tests.

3. If you used any of the other installation methods and don't have PHPUnit installed on your system yet, download and [install PHPUnit](https://phpunit.de/getting-started.html).

4. To get the unit tests running with a non-Composer-based install, you need to set an environment variable so the PHPCompatibility unit test suite will know where to find PHPCS.

The most flexible way to do this, is by setting this variable in a custom `phpunit.xml` file.

1. Copy the existing `phpunit.xml.dist` file in the root directory of the PHPCompatibility repository and name it `phpunit.xml`.

2. Add the following snippet to the new file, replacing the value `/path/to/PHPCS` with the path to the directory in which you installed PHP CodeSniffer on your system:

```xml

<php>

<env name="PHPCS\_DIR" value="/path/to/PHPCS"/>

</php>

```

3. Run the tests by running `phpunit` from the root directory of your PHPCompatibility install.

It will automatically read the `phpunit.xml` file and execute the tests.

#### Issues when running the PHPCS Unit tests for another standard

This sniff library uses its own PHPUnit setup rather than the PHP CodeSniffer native unit testing framework to allow for testing the sniffs with various settings for the `testVersion` config variable.

If you are running the PHPCS native unit tests or the unit tests for another sniff library which uses the PHPCS native unit testing framework, PHPUnit might throw errors related to this sniff library depending on your setup.

This will generally only happen if you have both PHPCompatibility as well as another custom sniff library in your PHPCS `installed\_paths` setting.

To fix these errors, make sure you are running PHPCS 2.7.1 or higher and add the following to the `phpunit.xml` file for the sniff library you are testing:

```xml

<php>

<env name="PHPCS\_IGNORE\_TESTS" value="PHPCompatibility"/>

</php>

```

This will prevent PHPCS trying to include the PHPCompatibility unit tests when creating the test suite.