# Contributing

Contributions to Respect\Validation are always welcome. You make our lives

easier by sending us your contributions through [pull requests][].

Pull requests for bug fixes must be based on the oldest stable version's branch

whereas pull requests for new features must be based on the `master` branch.

Due to time constraints, we are not always able to respond as quickly as we

would like. Please do not take delays personal and feel free to remind us here,

on IRC, or on Gitter if you feel that we forgot to respond.

Please see the [project documentation][] before proceeding. You should also know

about [PHP-FIG][]'s standards and basic unit testing, but we're sure you can

learn that just by looking at other rules. Pick the simple ones like `ArrayType`

to begin.

Before writing anything, feature or bug fix:

- Check if there is already an issue related to it (opened or closed) and if

someone is already working on that;

- If there is not, [open an issue][] and notify everybody that you're going

to work on that;

- If there is, create a comment to notify everybody that you're going to

work on that.

- Make sure that what you need is not done yet

## Adding a new validator

A common validator (rule) on Respect\Validation is composed of three classes:

\* `library/Rules/YourRuleName.php`: the rule itself

\* `library/Exceptions/YourRuleNameException.php`: the exception thrown by the rule

\* `tests/unit/Rules/YourRuleNameTest.php`: tests for the rule

The classes are pretty straightforward. In the sample below, we're going to

create a validator that validates if a string is equal to "Hello World".

- Classes should be `final` unless they are used in a different scope;

- Properties should be `private` unless they are used in a different scope;

- Classes should use strict typing;

- Some docblocks are required.

### Creating the rule

The rule itself needs to implement the `Validatable` interface but, it is

convenient to just extend the `AbstractRule` class.

Doing that, you'll only need to declare one method: `validate($input)`.

This method must return `true` or `false`.

If your validator class is `HelloWorld`, it will be available as `v::helloWorld()`

and will natively have support for chaining and everything else.

```php

<?php

/\*

\* This file is part of Respect/Validation.

\*

\* (c) Alexandre Gomes Gaigalas <alexandre@gaigalas.net>

\*

\* For the full copyright and license information, please view the LICENSE file

\* that was distributed with this source code.

\*/

declare(strict\_types=1);

namespace Respect\Validation\Rules;

/\*\*

\* Explain in one sentence what this rule does.

\*

\* @author Your Name <youremail@yourdomain.tld>

\*/

final class HelloWorld extends AbstractRule

{

/\*\*

\* {@inheritDoc}

\*/

public function validate($input): bool

{

return $input === 'Hello World';

}

}

```

### Creating the rule exception

Just that and we're done with the rule code. The Exception requires you to

declare messages used by `assert()` and `check()`. Messages are declared in

affirmative and negative moods, so if anyone calls `v::not(v::helloWorld())` the

library will show the appropriate message.

```php

<?php

/\*

\* This file is part of Respect/Validation.

\*

\* (c) Alexandre Gomes Gaigalas <alexandre@gaigalas.net>

\*

\* For the full copyright and license information, please view the LICENSE file

\* that was distributed with this source code.

\*/

declare(strict\_types=1);

namespace Respect\Validation\Exceptions;

/\*\*

\* @author Your Name <youremail@yourdomain.tld>

\*/

final class HelloWorldException extends ValidationException

{

/\*\*

\* {@inheritDoc}

\*/

protected $defaultTemplates = [

self::MODE\_DEFAULT => [

self::STANDARD => '{{name}} must be a Hello World',

],

self::MODE\_NEGATIVE => [

self::STANDARD => '{{name}} must not be a Hello World',

]

];

}

```

### Creating unit tests

Finally, we need to test if everything is running smooth. We have `RuleTestCase`

that allows us to make easier to test rules, but you fell free to use the

`PHPUnit\Framework\TestCase` if you want or you need it's necessary.

The `RuleTestCase` extends PHPUnit's `PHPUnit\Framework\TestCase` class, so you

are able to use any methods of it. By extending `RuleTestCase` you should

implement two methods that should return a [data provider][] with the rule as

first item of the arrays:

- `providerForValidInput`: Will test when `validate()` should return `true`

- `providerForInvalidInput`: Will test when `validate()` should return `false`

```php

<?php

/\*

\* This file is part of Respect/Validation.

\*

\* (c) Alexandre Gomes Gaigalas <alexandre@gaigalas.net>

\*

\* For the full copyright and license information, please view the LICENSE file

\* that was distributed with this source code.

\*/

declare(strict\_types=1);

namespace Respect\Validation\Rules;

use Respect\Validation\Test\RuleTestCase;

/\*\*

\* @group rule

\*

\* @covers \Respect\Validation\Rules\HelloWorld

\*

\* @author Your Name <youremail@yourdomain.tld>

\*/

final class HelloWorldTest extends RuleTestCase

{

/\*\*

\* {@inheritDoc}

\*/

public function providerForValidInput(): array

{

$rule = new HelloWorld();

return [

[$rule, 'Hello World'],

];

}

/\*\*

\* {@inheritDoc}

\*/

public function providerForInvalidInput(): array

{

$rule = new HelloWorld();

return [

[$rule, 'Not a hello'],

[$rule, 'Hello darkness, my old friend'],

[$rule, 'Hello is it me you\'re looking for?'],

];

}

}

```

If the constructor of your rule accepts arguments you may create specific tests

for it other than what is covered by `RuleTestCase`.

### Helping us a little bit more

You rule will be accepted only with these 3 files (rule, exception and unit test),

but if you really want to help us, you can follow the example of [ArrayType][] by:

- Adding your new rule on the `Validator`'s class docblock;

- Writing a documentation for your new rule;

- Creating integration tests with PHPT.

As we already said, none of them are required but you will help us a lot.

## Documentation

Our docs at https://respect-validation.readthedocs.io are generated from our

Markdown files. Add your brand new rule and it should be soon available.

## Running Tests

After run `composer install` on the library's root directory you must run PHPUnit.

### Linux

You can test the project using the commands:

```sh

$ vendor/bin/phpunit

```

or

```sh

$ composer phpunit

```

### Windows

You can test the project using the commands:

```sh

> vendor\bin\phpunit

```

or

```sh

> composer phpunit

```

No test should fail.

You can tweak the PHPUnit's settings by copying `phpunit.xml.dist` to `phpunit.xml`

and changing it according to your needs.

[ArrayType]: https://github.com/Respect/Validation/commit/f08a1fa

[data provider]: https://phpunit.de/manual/current/en/writing-tests-for-phpunit.html#writing-tests-for-phpunit.data-providers "PHPUnit Data Providers"

[open an issue]: http://github.com/Respect/Validation/issues

[PHP-FIG]: http://www.php-fig.org "PHP Framework Interop Group"

[project documentation]: https://respect-validation.readthedocs.io/ "Respect\Validation documentation"

[pull requests]: http://help.github.com/pull-requests "GitHub pull requests"