
Php Data Tester Documentation

Release 0.1.0

Martin Poirier Théorêt

Aug 13, 2018

Contents

1	Getting Started	3
1.1	Requirements	3
1.2	Create a Data Tester	3
1.3	Path	4
1.4	Chaining Path	4
1.5	Deep Path	4
1.6	Each	5
2	Advance	7
2.1	Optional Path	7
2.2	Transform	8
2.3	Reusable Test Callable	9
3	Asserts	11
3.1	assertArraySubset	11
3.2	assertContains	11
3.3	assertNotContains	12
3.4	assertContainsOnly	13
3.5	assertContainsOnlyInstancesOf	13
3.6	assertNotContainsOnly	13
3.7	assertCount	14
3.8	assertNotCount	14
3.9	assertEquals	14
3.10	assertNotEquals	15
3.11	assertEmpty	15
3.12	assertNotEmpty	16
3.13	assertGreaterThan	16
3.14	assertGreaterThanOrEqual	16
3.15	assertLessThan	17
3.16	assertLessThanOrEqual	17
3.17	assertTrue	17
3.18	assertNotTrue	18
3.19	assertFalse	18
3.20	assertNotFalse	18
3.21	assertNull	19
3.22	assertNotNull	19
3.23	assertFinite	19

3.24	assertInfinite	20
3.25	assertNaN	20
3.26	assertSame	20
3.27	assertNotSame	21
3.28	assertInstanceOf	21
3.29	assertNotInstanceOf	22
3.30	assertInternalType	22
3.31	assertNotInternalType	22
3.32	assertRegExp	23
3.33	assertNotRegExp	23
3.34	assertSameSize	23
3.35	assertNotSameSize	24
3.36	assertStringMatchesFormat	24
3.37	assertStringNotMatchesFormat	24
3.38	assertStringStartsWith	25
3.39	assertStringStartsNotWith	25
3.40	assertStringEndsWith	25
3.41	assertStringEndsNotWith	26
3.42	assertJson	26
3.43	assertJsonStringEqualsJsonString	26
3.44	assertJsonStringNotEqualsJsonString	27

This library is a wrapper around **PHPUnit Assert** class to be able to use a fluent interface on the data you want to test. The library can be install via [Composer/Packagist](#).

Here is a quick example of how to use it in a **PHPUnit TestCase**:

```
<?php
namespace Your\Project\Name;

use PHPUnit\Framework\TestCase;
use Draw\DataTester\Tester;

class SimpleTest extends TestCase
{
    public function test ()
    {
        $data = [
            'key1' => 'value1',
            'key2' => (object) ['toto' => 'value']
        ];

        $tester = new Tester($data);
        $tester->assertInternalType('array')
            ->assertCount(2)
            ->path('[key1]')->assertSame('value1');
        $tester->path('[key2].toto')->assertSame('value');
    }
}
```

There is a lot more features available, just [Read the Docs!](#)

1.1 Requirements

This library is compatible with currently supported version of **PHPUnit** versions (^6.0|^5.7) and **PHP** (^5.6|^7.0). You can check the CI test on travis <https://travis-ci.org/mpoiriert/php-data-tester>.

1.2 Create a Data Tester

From a PHPUnit test case you simply create a new **Draw\DataTester\Tester** instance:

Listing 1: Example: Simple Test

```
<?php
namespace Your\Project\Name;

use PHPUnit\Framework\TestCase;
use Draw\DataTester\Tester;

class ExampleTest extends TestCase
{
    public function test()
    {
        $dataToTest = 'A string value';

        $tester = new Tester($dataToTest);
        $tester
            ->assertInternalType('string')
            ->assertSame('A string value');
    }
}
```

The **Tester** use a fluent interface by returning himself on all of the **assert*** methods and most of his methods. This allow to easily make multiple test on the same *data*.

If you don't need a reference to the tester you can be even more concise:

Listing 2: Example: New Concise

```
(new Tester('A string value'))
->assertInternalType('string')
->assertSame('A string value');
```

1.3 Path

For more complex data (array, object) you can use the **path** method to test something deeper in the data itself:

Listing 3: Example: Path

```
(new Tester((object) ["key" => "value"]))
->path('key')
->assertSame('value');
```

By Using the **path** method you are making a assertion that the *path* is accessible. It also return a new **Tester** instance with the *data* of the *path* to be tested.

Listing 4: Example: Path Callable

```
(new Tester((object) ["key" => "value"]))
->path('key')
->assertSame('value');
```

The library use behind it is the symfony/property-access, make sure you read the doc https://symfony.com/doc/current/components/property_access.html

1.4 Chaining Path

Since the **path** method return a new **Tester** you must keep a reference on the original **Tester** if you want to test other **path**.

Listing 5: Example: Chain Path

```
$tester = new Tester((object) ["key1" => "value1", "key2" => "value2"]);
$tester->path('key1')->assertSame('value1');
$tester->path('key2')->assertSame('value2');
```

1.5 Deep Path

If you have a deeper object you can call **path** in chain:

Listing 6: Example: Deeper Path

```
(new Tester((object) ["level1" => (object) ["level2" => "value"]]))
->path('level1')
->path('level2')->assertSame('value');
```

1.6 Each

If your data is **iterable** trough **foreach** you can test all the entry via a callable:

Listing 7: Example: Each

```
(new Tester(['value1', 'value2']))
->each(
    function (Tester $tester) {
        $tester->assertInternalType('string');
    }
);
```


Here is a list of advance examples that you can follow to make your test more efficient and maintainable.

2.1 Optional Path

Considering you have a complex structure with optional **path** into it. You can use the method **ifPathIsReadable** to make some test **optional**:

Listing 1: Example: If Path Is Readable

```
(new Tester(null))
  ->ifPathIsReadable(
    'notExistingPath',
    function (Tester $tester) {
      //Will not be call with current data to test
    }
  );
```

This obviously make more sense with a combination of **each**. In this more complex example lets say you receive a list of users object that don't have the same properties available:

Listing 2: Example If Path Is Readable And Each

```
$users = [
  (object)[
    'firstName' => 'Martin',
    'active' => true,
    'referral' => 'Google'
  ],
  (object)[
    'firstName' => 'Julie',
    'active' => false
  ]
]
```

(continues on next page)

(continued from previous page)

```
];
(new Tester($users))
    ->each(
        function (Tester $tester) {
            $tester->path('firstName')->assertInternalType('string');
            $tester->path('active')->assertInternalType('boolean');
            $tester->ifPathIsReadable(
                'referral',
                function (Tester $tester) {
                    $tester->assertInternalType('string');
                }
            );
        }
    );
```

2.2 Transform

If you need to **transform** the *data* during the test you can call the **transform** method with a **callable** as the first argument for the transformation. It will return a new **Tester** with the transformed data to test.

Let's you have a **json** string as *data*, that you want to test the content, it will look like this:

Listing 3: Example: Transform

```
(new Tester('{"key":"value"}'))
    ->transform('json_decode')
    ->path('key')->assertSame('value');
```

Ideally you should **test** your *data* before transforming it:

Listing 4: Example: Assert Before Transform

```
(new Tester('{"key":"value"}'))
    ->assertJson()
    ->transform('json_decode')
    ->path('key')->assertSame('value');
```

If you would like to transform the data but not with the default values of callable you can simply create a custom callable with the appropriate option. Let say you want **json_decode** with a associative array:

Listing 5: Example: Assert Before Transform-custom

```
(new Tester('{"key":"value"}'))
->assertJson()
->transform(
    function ($data) {
        return json_decode($data, true);
    }
)->path('[key]')->assertSame('value');
```

Take a note that since it's a associative array the path must be change from **key** to **[key]**.

2.3 Reusable Test Callable

Some time you want to execute the same set of test on different source of data. You have a method that return you a user and one that return you a list of users. You can simply create a class that as all the test inside of it.

Listing 6: Example: Class Callable

```
<?php\nnamespace Your\\Project\\Name;\\nuse Draw\\DataTester\\Tester;
class UserDataTester
{
    public function __invoke(Tester $tester)
    {
        $tester->path('firstName')->assertInternalType('string');
        $tester->path('active')->assertInternalType('boolean');
        $tester->ifPathIsReadable(
            'referral',
            function (Tester $tester) {
                $tester->assertInternalType('string');
            }
        );
    }
}
```

And now you can use it to test the *data* of one user:

Listing 7: Example: Test With Class Callable

```
$user = (object)[
    'firstName' => 'Martin',
    'active' => true,
    'referral' => 'Google'
];

(new Tester($user))
->test(new UserDataTester());
```

Or with **each** in case of a list of users:

Listing 8: Example: Each With Class Callable

```
$users = [
    (object)[
        'firstName' => 'Martin',
```

(continues on next page)

(continued from previous page)

```
        'active' => true,
        'referral' => 'Google'
    ],
    (object) [
        'firstName' => 'Julie',
        'active' => false
    ]
];

(new Tester($users))
->each(new UserDataTester());
```

The list of asserts available are a sub-set of the **PHPUnit Assert** available.

Some of the methods have been remove since they are replicable trough a combination of **path** and another assert. Other are not available either for compatibility issues. If you think that some must be added just open a issue in the git repository.

For a more exhaustive documentation please refer to [PHPUnit Documentation](#). Do not forgot that all the asserts are not available and that the **\$this->getData()** replace the data you want to test that is normally pass trough the **PHPUnit Assert** methods.

3.1 assertArraySubset

```
p
/**
 * Asserts that an array has a specified subset.
 *
 * @param array|ArrayAccess $subset
 * @param bool $strict Check for object identity
 * @param string $message
 * @return $this
 */
public function assertArraySubset($subset, $strict = false, $message = '')
{
    Assert::assertArraySubset($subset, $this->getData(), $strict, $message);

    return $this;
}
```

3.2 assertContains

```
p
/**
 * Asserts that a haystack contains a needle.
 *
 * @param mixed $needle
 * @param string $message
 * @param bool $ignoreCase
 * @param bool $checkForObjectIdentity
 * @param bool $checkForNonObjectIdentity
 * @return $this
 */
public function assertContains(
    $needle,
    $message = '',
    $ignoreCase = false,
    $checkForObjectIdentity = true,
    $checkForNonObjectIdentity = false
) {
    Assert::assertContains($needle, $this->getData(), $message, $ignoreCase,
↪$checkForObjectIdentity,
        $checkForNonObjectIdentity);

    return $this;
}
```

3.3 assertNotContains

```
p
/**
 * Asserts that a haystack does not contain a needle.
 *
 * @param mixed $needle
 * @param string $message
 * @param bool $ignoreCase
 * @param bool $checkForObjectIdentity
 * @param bool $checkForNonObjectIdentity
 * @return $this
 */
public function assertNotContains(
    $needle,
    $message = '',
    $ignoreCase = false,
    $checkForObjectIdentity = true,
    $checkForNonObjectIdentity = false
) {
    Assert::assertNotContains($needle, $this->getData(), $message, $ignoreCase,
↪$checkForObjectIdentity,
        $checkForNonObjectIdentity);

    return $this;
}
```


3.4 assertContainsOnly

```

p
/**
 * Asserts that a haystack contains only values of a given type.
 *
 * @param string $type
 * @param bool $isNativeType
 * @param string $message
 * @return $this
 */
public function assertContainsOnly($type, $isNativeType = null, $message = '')
{
    Assert::assertContainsOnly($type, $this->getData(), $isNativeType, $message);

    return $this;
}

```

3.5 assertContainsOnlyInstancesOf

```

p
/**
 * Asserts that a haystack contains only instances of a given classname
 *
 * @param string $classname
 * @param string $message
 * @return $this
 */
public function assertContainsOnlyInstancesOf($classname, $message = '')
{
    Assert::assertContainsOnlyInstancesOf($classname, $this->getData(), $message);

    return $this;
}

```

3.6 assertNotContainsOnly

```

p
/**
 * Asserts that a haystack does not contain only values of a given type.
 *
 * @param string $type
 * @param bool $isNativeType
 * @param string $message
 * @return $this
 */
public function assertNotContainsOnly($type, $isNativeType = null, $message = '')
{
    Assert::assertNotContainsOnly($type, $this->getData(), $isNativeType, $message);

    return $this;
}

```

3.7 assertCount

```
p
/**
 * Asserts the number of elements of an array, Countable or Traversable.
 *
 * @param int $expectedCount
 * @param string $message
 * @return $this
 */
public function assertCount($expectedCount, $message = '')
{
    Assert::assertCount($expectedCount, $this->getData(), $message);

    return $this;
}
```

3.8 assertNotCount

```
p
/**
 * Asserts the number of elements of an array, Countable or Traversable.
 *
 * @param int $expectedCount
 * @param string $message
 * @return $this
 */
public function assertNotCount($expectedCount, $message = '')
{
    Assert::assertNotCount($expectedCount, $this->getData(), $message);

    return $this;
}
```

3.9 assertEquals

```
p
/**
 * Asserts that two variables are equal.
 *
 * @param mixed $expected
 * @param string $message
 * @param float $delta
 * @param int $maxDepth
 * @param bool $canonicalize
 * @param bool $ignoreCase
 * @return $this
 */
public function assertEquals(
    $expected,
    $message = '',
```

(continues on next page)

(continued from previous page)

```

        $delta = 0,
        $maxDepth = 10,
        $canonicalize = false,
        $ignoreCase = false
    ) {
        Assert::assertEquals($expected, $this->getData(), $message, $delta, $maxDepth,
↪$canonicalize, $ignoreCase);

        return $this;
    }

```

3.10 assertNotEquals

```

p
/**
 * Asserts that two variables are not equal.
 *
 * @param mixed $expected
 * @param string $message
 * @param float $delta
 * @param int $maxDepth
 * @param bool $canonicalize
 * @param bool $ignoreCase
 * @return $this
 */
public function assertNotEquals(
    $expected,
    $message = '',
    $delta = 0,
    $maxDepth = 10,
    $canonicalize = false,
    $ignoreCase = false
) {
    Assert::assertNotEquals($expected, $this->getData(), $message, $delta, $maxDepth,
↪$canonicalize, $ignoreCase);

    return $this;
}

```

3.11 assertEmpty

```

p
/**
 * Asserts that a variable is empty.
 *
 * @param string $message
 *
 * @return $this
 */
public function assertEmpty($message = '')
{

```

(continues on next page)

(continued from previous page)

```
Assert::assertEmpty($this->getData(), $message);

return $this;
}
```

3.12 assertNotEmpty

```
p
/**
 * Asserts that a variable is not empty.
 *
 * @param string $message
 *
 * @return $this
 */
public function assertNotEmpty($message = '')
{
    Assert::assertNotEmpty($this->getData(), $message);

    return $this;
}
```

3.13 assertGreaterThan

```
p
/**
 * Asserts that a value is greater than another value.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
public function assertGreaterThan($expected, $message = '')
{
    Assert::assertGreaterThan($expected, $this->getData(), $message);

    return $this;
}
```

3.14 assertGreaterThanOrEqual

```
p
/**
 * Asserts that a value is greater than or equal to another value.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
```

(continues on next page)

(continued from previous page)

```

*/
public function assertGreaterThanOrEqual($expected, $message = '')
{
    Assert::assertGreaterThanOrEqual($expected, $this->getData(), $message);

    return $this;
}

```

3.15 assertLessThan

```

p
/**
 * Asserts that a value is smaller than another value.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
public function assertLessThan($expected, $message = '')
{
    Assert::assertLessThan($expected, $this->getData(), $message);

    return $this;
}

```

3.16 assertLessThanOrEqual

```

p
/**
 * Asserts that a value is smaller than or equal to another value.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
public function assertLessThanOrEqual($expected, $message = '')
{
    Assert::assertLessThanOrEqual($expected, $this->getData(), $message);

    return $this;
}

```

3.17 assertTrue

```

p
/**
 * Asserts that a condition is true.
 *

```

(continues on next page)

(continued from previous page)

```
* @param string $message
*
* @return $this
*/
public function assertTrue($message = '')
{
    Assert::assertTrue($this->getData(), $message);

    return $this;
}
```

3.18 assertTrue

```
p
/**
 * Asserts that a condition is not true.
 *
 * @param string $message
 *
 * @return $this
 */
public function assertTrue($message = '')
{
    Assert::assertTrue($this->getData(), $message);

    return $this;
}
```

3.19 assertFalse

```
p
/**
 * Asserts that a condition is false.
 *
 * @param string $message
 *
 * @return $this
 */
public function assertFalse($message = '')
{
    Assert::assertFalse($this->getData(), $message);

    return $this;
}
```

3.20 assertTrue

```

p
/**
 * Asserts that a condition is not false.
 *
 * @param string $message
 *
 * @return $this
 */
public function assertNotFalse($message = '')
{
    Assert::assertNotFalse($this->getData(), $message);

    return $this;
}

```

3.21 assertNull

```

p
/**
 * Asserts that a variable is null.
 *
 * @param string $message
 * @return $this
 */
public function assertNull($message = '')
{
    Assert::assertNull($this->getData(), $message);

    return $this;
}

```

3.22 assertNotNull

```

p
/**
 * Asserts that a variable is not null.
 *
 * @param string $message
 * @return $this
 */
public function assertNotNull($message = '')
{
    Assert::assertNotNull($this->getData(), $message);

    return $this;
}

```

3.23 assertFinite

```
p
/**
 * Asserts that a variable is finite.
 *
 * @param string $message
 * @return $this
 */
public function assertFinite($message = '')
{
    Assert::assertFinite($this->getData(), $message);

    return $this;
}
```

3.24 assertInfinite

```
p
/**
 * Asserts that a variable is infinite.
 *
 * @param string $message
 * @return $this
 */
public function assertInfinite($message = '')
{
    Assert::assertInfinite($this->getData(), $message);

    return $this;
}
```

3.25 assertNan

```
p
/**
 * Asserts that a variable is nan.
 *
 * @param string $message
 * @return $this
 */
public function assertNan($message = '')
{
    Assert::assertNan($this->getData(), $message);

    return $this;
}
```

3.26 assertSame


```

p
/**
 * Asserts that two variables have the same type and value.
 * Used on objects, it asserts that two variables reference
 * the same object.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
public function assertSame($expected, $message = '')
{
    Assert::assertSame($expected, $this->getData(), $message);

    return $this;
}

```

3.27 assertNotSame

```

p
/**
 * Asserts that two variables do not have the same type and value.
 * Used on objects, it asserts that two variables do not reference
 * the same object.
 *
 * @param mixed $expected
 * @param string $message
 * @return $this
 */
public function assertNotSame($expected, $message = '')
{
    Assert::assertNotSame($expected, $this->getData(), $message);

    return $this;
}

```

3.28assertInstanceOf

```

p
/**
 * Asserts that a variable is of a given type.
 *
 * @param string $expected
 * @param string $message
 * @return $this
 */
public function assertInstanceOf($expected, $message = '')
{
    Assert::assertInstanceOf($expected, $this->getData(), $message);

    return $this;
}

```

3.29 assertNotInstanceOf

```
p
/**
 * Asserts that a variable is not of a given type.
 *
 * @param string $expected
 * @param string $message
 * @return $this
 */
public function assertNotInstanceOf($expected, $message = '')
{
    Assert::assertNotInstanceOf($expected, $this->getData(), $message);

    return $this;
}
```

3.30 assertInternalType

```
p
/**
 * Asserts that a variable is of a given type.
 *
 * @param string $expected
 * @param string $message
 * @return $this
 */
public function assertInternalType($expected, $message = '')
{
    Assert::assertInternalType($expected, $this->getData(), $message);

    return $this;
}
```

3.31 assertNotInternalType

```
p
/**
 * Asserts that a variable is not of a given type.
 *
 * @param string $expected
 * @param string $message
 * @return $this
 */
public function assertNotInternalType($expected, $message = '')
{
    Assert::assertNotInternalType($expected, $this->getData(), $message);

    return $this;
}
```

3.32 assertRegExp

```

p
/**
 * Asserts that a string matches a given regular expression.
 *
 * @param string $pattern
 * @param string $message
 * @return $this
 */
public function assertRegExp($pattern, $message = '')
{
    Assert::assertRegExp($pattern, $this->getData(), $message);

    return $this;
}

```

3.33 assertNotRegExp

```

p
/**
 * Asserts that a string does not match a given regular expression.
 *
 * @param string $pattern
 * @param string $message
 * @return $this
 */
public function assertNotRegExp($pattern, $message = '')
{
    Assert::assertNotRegExp($pattern, $this->getData(), $message);

    return $this;
}

```

3.34 assertSameSize

```

p
/**
 * Assert that the size of two arrays (or `Countable` or `Traversable` objects)
 * is the same.
 *
 * @param array|Countable|Traversable $expected
 * @param string $message
 * @return $this
 */
public function assertSameSize($expected, $message = '')
{
    Assert::assertSameSize($expected, $this->getData(), $message);

    return $this;
}

```

3.35 assertNotSameSize

```
p
/**
 * Assert that the size of two arrays (or `Countable` or `Traversable` objects)
 * is not the same.
 *
 * @param array|Countable|Traversable $expected
 * @param string $message
 * @return $this
 */
public function assertNotSameSize($expected, $message = '')
{
    Assert::assertNotSameSize($expected, $this->getData(), $message);

    return $this;
}
```

3.36 assertStringMatchesFormat

```
p
/**
 * Asserts that a string matches a given format string.
 *
 * @param string $format
 * @param string $message
 * @return $this
 */
public function assertStringMatchesFormat($format, $message = '')
{
    Assert::assertStringMatchesFormat($format, $this->getData(), $message);

    return $this;
}
```

3.37 assertStringNotMatchesFormat

```
p
/**
 * Asserts that a string does not match a given format string.
 *
 * @param string $format
 * @param string $message
 * @return $this
 */
public function assertStringNotMatchesFormat($format, $message = '')
{
    Assert::assertStringNotMatchesFormat($format, $this->getData(), $message);

    return $this;
}
```

3.38 assertStringStartsWith

```
p
/**
 * Asserts that a string starts with a given prefix.
 *
 * @param string $prefix
 * @param string $message
 * @return $this
 */
public function assertStringStartsWith($prefix, $message = '')
{
    Assert::assertStringStartsWith($prefix, $this->getData(), $message);

    return $this;
}
```

3.39 assertStringStartsWithNot

```
p
/**
 * Asserts that a string starts not with a given prefix.
 *
 * @param string $prefix
 * @param string $message
 * @return $this
 */
public function assertStringStartsWithNot($prefix, $message = '')
{
    Assert::assertStringStartsWithNot($prefix, $this->getData(), $message);

    return $this;
}
```

3.40 assertStringEndsWith

```
p
/**
 * Asserts that a string ends with a given suffix.
 *
 * @param string $suffix
 * @param string $message
 * @return $this
 */
public function assertStringEndsWith($suffix, $message = '')
{
    Assert::assertStringEndsWith($suffix, $this->getData(), $message);

    return $this;
}
```

3.41 assertStringEndsWith

```
p
/**
 * Asserts that a string ends not with a given suffix.
 *
 * @param string $suffix
 * @param string $message
 * @return $this
 */
public function assertStringEndsWith($suffix, $message = '')
{
    Assert::assertStringEndsWith($suffix, $this->getData(), $message);

    return $this;
}
```

3.42 assertJson

```
p
/**
 * Asserts that a string is a valid JSON string.
 *
 * @param string $message
 * @return $this
 */
public function assertJson($message = '')
{
    Assert::assertJson($this->getData(), $message);

    return $this;
}
```

3.43 assertJsonStringEqualsJsonString

```
p
/**
 * Asserts that two given JSON encoded objects or arrays are equal.
 *
 * @param string $expectedJson
 * @param string $message
 * @return $this
 */
public function assertJsonStringEqualsJsonString($expectedJson, $message = '')
{
    Assert::assertJsonStringEqualsJsonString($expectedJson, $this->getData(),
↪$message);

    return $this;
}
```

3.44 assertJsonStringNotEqualsJsonString

```
p
/**
 * Asserts that two given JSON encoded objects or arrays are not equal.
 *
 * @param string $expectedJson
 * @param string $message
 * @return $this
 */
public function assertJsonStringNotEqualsJsonString($expectedJson, $message = '')
{
    Assert::assertJsonStringNotEqualsJsonString($expectedJson, $this->getData(),
↪ $message);

    return $this;
}
```