# This is the PHP port of Hamcrest Matchers

[Build Status](https://travis-ci.org/hamcrest/hamcrest-php)

Hamcrest is a matching library originally written for Java, but subsequently ported to many other languages. hamcrest-php is the official PHP port of Hamcrest and essentially follows a literal translation of the original Java API for Hamcrest, with a few Exceptions, mostly down to PHP language barriers:

1. instanceOf($theClass) is actually anInstanceOf($theClass)
2. both(containsString('a'))->and(containsString('b')) is actually both(containsString('a'))->andAlso(containsString('b'))
3. either(containsString('a'))->or(containsString('b')) is actually either(containsString('a'))->orElse(containsString('b'))
4. Unless it would be non-semantic for a matcher to do so, hamcrest-php allows dynamic typing for it's input, in "the PHP way". Exception are where semantics surrounding the type itself would suggest otherwise, such as stringContains() and greaterThan().
5. Several official matchers have not been ported because they don't make sense or don't apply in PHP:
   * typeCompatibleWith($theClass)
   * eventFrom($source)
   * hasProperty($name) \*\*
   * samePropertyValuesAs($obj) \*\*
6. When most of the collections matchers are finally ported, PHP-specific aliases will probably be created due to a difference in naming conventions between Java's Arrays, Collections, Sets and Maps compared with PHP's Arrays.

\*\* [Unless we consider POPO's (Plain Old PHP Objects) akin to JavaBeans] - The POPO thing is a joke. Java devs coin the term POJO's (Plain Old Java Objects).

## Usage

Hamcrest matchers are easy to use as:

Hamcrest\_MatcherAssert::assertThat('a', Hamcrest\_Matchers::equalToIgnoringCase('A'));

Alternatively, you can use the global proxy-functions:

$result = true;

// with an identifier

assertThat("result should be true", $result, equalTo(true));

// without an identifier

assertThat($result, equalTo(true));

// evaluate a boolean expression

assertThat($result === true);

// with syntactic sugar is()

assertThat(true, is(true));

:warning: **NOTE:** the global proxy-functions aren't autoloaded by default, so you will need to load them first:

\Hamcrest\Util::registerGlobalFunctions();

For brevity, all of the examples below use the proxy-functions.

## Documentation

A tutorial can be found on the [Hamcrest site](https://code.google.com/archive/p/hamcrest/wikis/TutorialPHP.wiki).

## Available Matchers

* [Array](http://../master/README.md#array)
* [Collection](http://../master/README.md#collection)
* [Object](http://../master/README.md#object)
* [Numbers](http://../master/README.md#numbers)
* [Type checking](http://../master/README.md#type-checking)
* [XML](http://../master/README.md#xml)

### Array

* anArray - evaluates an array

assertThat([], anArray());

* hasItemInArray - check if item exists in array

$list = range(2, 7, 2);

$item = 4;

assertThat($list, hasItemInArray($item));

* hasValue - alias of hasItemInArray
* arrayContainingInAnyOrder - check if array contains elements in any order

assertThat([2, 4, 6], arrayContainingInAnyOrder([6, 4, 2]));

assertThat([2, 4, 6], arrayContainingInAnyOrder([4, 2, 6]));

* containsInAnyOrder - alias of arrayContainingInAnyOrder
* arrayContaining - An array with elements that match the given matchers in the same order.

assertThat([2, 4, 6], arrayContaining([2, 4, 6]));

assertthat([2, 4, 6], not(arrayContaining([6, 4, 2])));

* contains - check array in same order

assertThat([2, 4, 6], contains([2, 4, 6]));

* hasKeyInArray - check if array has given key

assertThat(['name'=> 'foobar'], hasKeyInArray('name'));

* hasKey - alias of hasKeyInArray
* hasKeyValuePair - check if arary has given key, value pair

assertThat(['name'=> 'foobar'], hasKeyValuePair('name', 'foobar'));

* hasEntry - same as hasKeyValuePair
* arrayWithSize - check array has given size

assertthat([2, 4, 6], arrayWithSize(3));

* emptyArray - check if array is emtpy

assertThat([], emptyArray());

* nonEmptyArray

assertThat([1], nonEmptyArray());

### Collection

* emptyTraversable - check if traversable is empty

$empty\_it = new EmptyIterator;

assertThat($empty\_it, emptyTraversable());

* nonEmptyTraversable - check if traversable isn't empty

$non\_empty\_it = new ArrayIterator(range(1, 10));

assertThat($non\_empty\_it, nonEmptyTraversable());

a

* traversableWithSize

$non\_empty\_it = new ArrayIterator(range(1, 10));

assertThat($non\_empty\_it, traversableWithSize(count(range(1, 10))));

`

### Core

* allOf - Evaluates to true only if ALL of the passed in matchers evaluate to true.

assertThat([2,4,6], allOf(hasValue(2), arrayWithSize(3)));

* anyOf - Evaluates to true if ANY of the passed in matchers evaluate to true.

assertThat([2, 4, 6], anyOf(hasValue(8), hasValue(2)));

* noneOf - Evaluates to false if ANY of the passed in matchers evaluate to true.

assertThat([2, 4, 6], noneOf(hasValue(1), hasValue(3)));

* both + andAlso - This is useful for fluently combining matchers that must both pass.

assertThat([2, 4, 6], both(hasValue(2))->andAlso(hasValue(4)));

* either + orElse - This is useful for fluently combining matchers where either may pass,

assertThat([2, 4, 6], either(hasValue(2))->orElse(hasValue(4)));

* describedAs - Wraps an existing matcher and overrides the description when it fails.

$expected = "Dog";

$found = null;

// this assertion would result error message as Expected: is not null but: was null

//assertThat("Expected {$expected}, got {$found}", $found, is(notNullValue()));

// and this assertion would result error message as Expected: Dog but: was null

//assertThat($found, describedAs($expected, notNullValue()));

* everyItem - A matcher to apply to every element in an array.

assertThat([2, 4, 6], everyItem(notNullValue()));

* hasItem - check array has given item, it can take a matcher argument

assertThat([2, 4, 6], hasItem(equalTo(2)));

* hasItems - check array has givem items, it can take multiple matcher as arguments

assertThat([1, 3, 5], hasItems(equalTo(1), equalTo(3)));

### Object

* hasToString - check \_\_toString or toString method

class Foo {

public $name = null;

public function \_\_toString() {

return "[Foo]Instance";

}

}

$foo = new Foo;

assertThat($foo, hasToString(equalTo("[Foo]Instance")));

* equalTo - compares two instances using comparison operator '=='

$foo = new Foo;

$foo2 = new Foo;

assertThat($foo, equalTo($foo2));

* identicalTo - compares two instances using identity operator '==='

assertThat($foo, is(not(identicalTo($foo2))));

* anInstanceOf - check instance is an instance|sub-class of given class

assertThat($foo, anInstanceOf(Foo::class));

* any - alias of anInstanceOf
* nullValue check null

assertThat(null, is(nullValue()));

* notNullValue check not null

assertThat("", notNullValue());

* sameInstance - check for same instance

assertThat($foo, is(not(sameInstance($foo2))));

assertThat($foo, is(sameInstance($foo)));

* typeOf- check type

assertThat(1, typeOf("integer"));

* notSet - check if instance property is not set

assertThat($foo, notSet("name"));

* set - check if instance property is set

$foo->name = "bar";

assertThat($foo, set("name"));

### Numbers

* closeTo - check value close to a range

assertThat(3, closeTo(3, 0.5));

* comparesEqualTo - check with '=='

assertThat(2, comparesEqualTo(2));

* greaterThan - check '>'

assertThat(2, greaterThan(1));

* greaterThanOrEqualTo

assertThat(2, greaterThanOrEqualTo(2));

* atLeast - The value is >= given value

assertThat(3, atLeast(2));

* lessThan

assertThat(2, lessThan(3));

* lessThanOrEqualTo

assertThat(2, lessThanOrEqualTo(3));

* atMost - The value is <= given value

assertThat(2, atMost(3));

### String

* emptyString - check for empty string

assertThat("", emptyString());

* isEmptyOrNullString

assertThat(null, isEmptyOrNullString());

* nullOrEmptyString

assertThat("", nullOrEmptyString());

* isNonEmptyString

assertThat("foo", isNonEmptyString());

* nonEmptyString

assertThat("foo", nonEmptyString());

* equalToIgnoringCase

assertThat("Foo", equalToIgnoringCase("foo"));

* equalToIgnoringWhiteSpace

assertThat(" Foo ", equalToIgnoringWhiteSpace("Foo"));

* matchesPattern - matches with regex pattern

assertThat("foobarbaz", matchesPattern('/(foo)(bar)(baz)/'));

* containsString - check for substring

assertThat("foobar", containsString("foo"));

* containsStringIgnoringCase

assertThat("fooBar", containsStringIgnoringCase("bar"));

* stringContainsInOrder

assertThat("foo", stringContainsInOrder("foo"));

* endsWith - check string that ends with given value

assertThat("foo", endsWith("oo"));

* startsWith - check string that starts with given value

assertThat("bar", startsWith("ba"));

### Type-checking

* arrayValue - check array type

assertThat([], arrayValue());

* booleanValue

assertThat(true, booleanValue());

* boolValue - alias of booleanValue
* callableValue - check if value is callable

$func = function () {};

assertThat($func, callableValue());

* doubleValue

assertThat(3.14, doubleValue());

* floatValue

assertThat(3.14, floatValue());

* integerValue

assertThat(1, integerValue());

* intValue - alias of integerValue
* numericValue - check if value is numeric

assertThat("123", numericValue());

* objectValue - check for object

$obj = new stdClass;

assertThat($obj, objectValue());

* anObject

assertThat($obj, anObject());

* resourceValue - check resource type

$fp = fopen("/tmp/foo", "w+");

assertThat($fp, resourceValue());

* scalarValue - check for scaler value

assertThat(1, scalarValue());

* stringValue

assertThat("", stringValue());

### XML

* hasXPath - check xml with a xpath

$xml = <<<XML

<books>

<book>

<isbn>1</isbn>

</book>

<book>

<isbn>2</isbn>

</book>

</books>

XML;

$doc = new DOMDocument;

$doc->loadXML($xml);

assertThat($doc, hasXPath("book", 2));