JsHamcrest Documentation

Release 0.6.8

Destaquenet Technology Solutions

January 26, 2013

CONTENTS

JsHamcrest is a JavaScript library heavily inspired by Hamcrest. It provides a *large* library of *matcher objects* (also known as constraints or predicates) allowing "match" rules to be defined declaratively. Typical scenarios include testing frameworks, mocking libraries, UI validation rules and object querying.

This is an open source project licenced under the terms of The BSD License and sponsored by Destaquenet Technology Solutions, a brazilian software development and consultancy startup.

See Also:

PDF version of this documentation.

CHAPTER

ONE

DOWNLOAD THE LATEST VERSION

Choose your compression level:

- Production – Minified code, hard to debug
- Development – Uncompressed code, easy to debug

JsHamcrest source code is proudly hosted by GitHub.

CHAPTER

TWO

DOCUMENTATION CONTENTS

2.1 Getting Started

The first thing to do is download the JsHamcrest source file. You can find the download links at the *index* page.

2.1.1 Setting Up The First Example

To be able to use JsHamcrest inside the browser, all you have to do is link the source file from within your HTML file. For example, create a new file with the following content and open it on your web browser of choice:

```
<html>
<header>
    <title>Page title</title>
    <script type="text/javascript" src="path/to/jshamcrest.js"></script>

    <script type="text/javascript">
        var odd = JsHamcrest.Matchers.odd();
        alert(odd.matches(11)); // Expected: true
    </script>
</header>
<body>
</body>
</html>
```

If an alert message "true" pops up when you open the page, then congratulations!

Note: Don't forget to change the 4th line to make it point to your actual JsHamcrest source file.

2.1.2 JsHamcrest Outside The Web Browser

Since JsHamcrest doesn't depend on any browser-specific JavaScript features (like `document`, `window`, and so on), you should be able to use it with any modern stand-alone JavaScript interpreter.

The snippet below shows how to reproduce the previous example using Rhino, a Java-based JavaScript interpreter/compiler developed by Mozilla:

```
js> load('path/to/jshamcrest.js')
js> var odd = JsHamcrest.Matchers.odd()
js> odd.matches(11)
true
```

```
js> odd.matches(10)
false
```

Understanding The Code

On the previous example, you saw the `JsHamcrest.Matchers.odd()` function, which returns a *matcher object* that checks whether the given number is odd. In other words, a *matcher* is an object that determines whether two things are equivalent.

Now try to figure out what the following matchers do:

```
// Make JsHamcrest matchers globally accessible
JsHamcrest.Integration.copyMembers(this);

equalTo('10').matches(10);        // Expected: true
between(5).and(10).matches(7);    // Expected: true
greaterThan(Math.PI).matches(4); // Expected: true
```

To make things easier, try to read each statement backwards. For instance:

- ...is 10 **equal to** '10'?
- ...is 7 **between** 5 **and** 10?
- ...is 4 **greater than** Math.PI?

It's not that hard after all, huh?

See Also:

`JsHamcrest.Matchers` namespace for the complete list of matchers.

2.2 Uses Of JsHamcrest

Now that you're getting familiar with JsHamcrest, you might be wondering if it can be used to do any *real* work. Here goes some situations where you might find JsHamcrest useful.

2.2.1 Unit Testing

JsHamcrest *is not* a testing framework, but the matcher library provided by JsHamcrest is very useful for testing since you can build really expressive test assertions.

For example, take a look at this QUnit-based test suite:

```
// Integrates JsHamcrest with QUnit
JsHamcrest.Integration.QUnit();

$(document).ready(function(){
    test("Array might be empty", function() {
        assertThat([], empty());
    });

    test("Array might not be empty", function() {
        assertThat([1,2,3], not(empty()));
    });
});
```

Besides QUnit, JsHamcrest integrates with several other JavaScript testing frameworks.

See Also:

`JsHamcrest.Integration` namespace.

Mocking Frameworks

To further improve your JavaScript testing experience, Chris Leishman created a stub/mock framework called JsMockito, which uses the matchers provided by JsHamcrest under the hood.

2.2.2 Object Filtering

Another great use of matchers is to easily filter lists of objects according to specific rules.

For example, let's suppose you have a list of objects that represents a people database:

```
var people = [
    {name: 'Daniel',  gender: 'M', age: 20},
    {name: 'Anthony', gender: 'M', age: 52},
    {name: 'Alicia',  gender: 'F', age: 37},
    {name: 'Maria',   gender: 'F', age: 46},
    {name: 'Carl',    gender: 'M', age: 30},
    {name: 'Nicky',   gender: 'F', age: 27}
];
```

Let's create some matchers to be able to filter people by gender or age:

```
var CustomMatchers = {
    male: function() {
        return new JsHamcrest.SimpleMatcher({
            matches: function(person) {
                return person.gender == 'M';
            }
        });
    },

    female: function() {
        return new JsHamcrest.SimpleMatcher({
            matches: function(person) {
                return person.gender == 'F';
            }
        });
    },

    middleAged: function() {
        return new JsHamcrest.SimpleMatcher({
            matches: function(person) {
                return person.age >= 40 && person.age <= 60;
            }
        });
    }
};

JsHamcrest.Integration.installMatchers(CustomMatchers);
```

See Also:

Writing Custom Matchers.

That's all it takes! Now let's filter some data:

```
// First, let's make all JsHamcrest stuff globally accessible
JsHamcrest.Integration.copyMembers(this);

filter(people, male());            // Daniel, Anthony, Carl
filter(people, female());          // Alicia, Maria, Nicky
filter(people, middleAged());      // Anthony, Maria
filter(people, not(middleAged())); // Daniel, Alicia, Carl, Nicky

filter(people, either(middleAged()).or(female())); // Anthony, Alicia, Maria, Nicky
filter(people, either(male()).or(middleAged()));   // Daniel, Anthony, Maria, Carl

filter(people, both(middleAged()).and(female()));  // Maria
filter(people, both(male()).and(middleAged()));    // Anthony
```

2.2.3 Et Cetera

We are sure JsHamcrest can do a lot more than what's shown here. If you are doing something else with JsHamcrest, please *let us know*!

2.3 Writing Custom Matchers

JsHamcrest already provides a large set of matchers, but that doesn't mean you should stick with those. In fact, we encourage you to create your own matchers every time you feel the need for something more suitable to the problem you have in hand.

2.3.1 A Basic Example

To introduce you to this topic, we are going to implement a new matcher whose task is just say whether the actual number is the answer to life, the universe, and everything. This is not a very useful matcher, but it shows everything you need to create a basic matcher:

```
var theAnswerToLifeTheUniverseAndEverything = function() {
    return new JsHamcrest.SimpleMatcher({
        matches: function(actual) {
            return actual == 42;
        },

        describeTo: function(description) {
            description.append('the answer to life, the universe, and everything');
        }
    });
};
```

As you can see, a matcher is nothing more than a function that returns a `JsHamcrest.SimpleMatcher` object . All this particular matcher does is test whether the actual number is equal to 42.

Okay, let's put that matcher to use:

```
// Output: Expected the answer to life, the universe, and everything: Success
assertThat(42, theAnswerToLifeTheUniverseAndEverything());
```

```
// Output: Expected the answer to life, the universe, and everything but was 10
assertThat(10, theAnswerToLifeTheUniverseAndEverything());
```

Do I Need Custom Matchers?

But wouldn't it be simpler if we just use the `JsHamcrest.Matchers.equalTo()` matcher instead? Let's see an example:

```
// Output: Expected equal to 42: Success
assertThat(42, equalTo(42));

// Output: Expected equal to 42 but was 10
assertThat(10, equalTo(42));
```

At the end, the result is the same. The only downside though is that we lose the ability to describe the assertion with a meaningful language, more appropriate to the problem we need to solve.

Composing Matchers

You can use your custom matchers together with the built-in ones to compose even more interesting match rules. For example, if you want to match when a number *is not* the answer to life, the universe, and everything, just wrap your custom matcher with a `JsHamcrest.Matchers.not()` matcher and you're done:

```
// Output: Expected not the answer to life, the universe, and everything: Success
assertThat(10, not(theAnswerToLifeTheUniverseAndEverything()));

// Output: Expected not the answer to life, the universe, and everything but was 42
assertThat(42, not(theAnswerToLifeTheUniverseAndEverything()));
```

That way you get the best of both worlds.

2.3.2 Learning By Example

Another great way to learn how to implement new matchers is to look at the *existing ones*. There's plenty of examples, of various levels of complexity.

For instance, take a look at the source code of the `JsHamcrest.Matchers.hasSize()` matcher:

```
JsHamcrest.Matchers.hasSize = function(matcherOrValue) {
    // Uses 'equalTo' matcher if the given object is not a matcher
    if (!JsHamcrest.isMatcher(matcherOrValue)) {
        matcherOrValue = JsHamcrest.Matchers.equalTo(matcherOrValue);
    }

    return new JsHamcrest.SimpleMatcher({
        matches: function(actual) {
            return matcherOrValue.matches(actual.length);
        },

        describeTo: function(description) {
            description.append('has size ').appendDescriptionOf(matcherOrValue);
        },

        describeValueTo: function(actual, description) {
            description.append(actual.length);
```

```
        }
    });
};
```

This matcher is prepared to use either matchers or numbers as the expected array size:

```
assertThat([1,2,3], hasSize(3));
assertThat([1,2,3], hasSize(lessThan(5)));
```

2.3.3 Distributing Your Custom Set Of Matchers

Let's suppose you have a couple of custom matchers you want to distribute to some people:

```
// filename: power_matchers.js

PowerMatchers = {
    rocks: function() {
        // ...
    },

    sucks: function() {
        // ...
    }
};
```

All you need to do is call `JsHamcrest.Integration.installMatchers()` at the end of your script, passing the namespace of your matchers as argument:

```
// filename: power_matchers.js

PowerMatchers = {
    // ...
};

JsHamcrest.Integration.installMatchers(PowerMatchers);
```

That's it. To plug your new matcher library, just link your script after JsHamcrest itself:

```
<html>
<header>
    <title>Page title</title>
    <script type="text/javascript" src="path/to/jshamcrest.js"></script>
    <script type="text/javascript" src="path/to/power_matchers.js"></script>

    <script type="text/javascript">
        // Displays the assertion descriptions using web browser's alert() function
        JsHamcrest.Integration.WebBrowser();

        var obj1 = {...}, obj2 = {...};

        assertThat(obj1, rocks());
        assertThat(obj2, sucks());
    </script>
</header>
<body>
</body>
</html>
```

2.4 JsHamcrest API Reference

JsHamcrest namespace overview.

2.4.1 `JsHamcrest` — Main Namespace

Provides the main namespace, along with core abstractions.

`JsHamcrest.areArraysEqual`(*array*, *anotherArray*)
Returns whether the arrays *array* and *anotherArray* are equivalent.

Parameters

- **array** – An array.
- **anotherArray** – Another array.

Returns True if both arrays are equivalent; false otherwise.

`JsHamcrest.isMatcher`(*obj*)
Returns whether *obj* is a matcher.

Parameters **obj** – Object.

Returns True if the given object is a matcher; false otherwise.

`JsHamcrest.version`
Library version.

`JsHamcrest.CombinableMatcher` Class

class `JsHamcrest.CombinableMatcher`(*{matches*, *describeTo*, *describeValueTo}*)
Extends `JsHamcrest.SimpleMatcher`. Defines a composite matcher, that is, a matcher that wraps several matchers into one.

Parameters

- **matches** – Function that provides the matching logic.
- **describeTo** – Function that describes this matcher to a `JsHamcrest.Description`.
- **describeValueTo** – *(Optional)* Function that describes the actual value under test. If not provided, the actual value will be described as a JavaScript literal.

`CombinableMatcher.and`(*matcherOrValue*)
Wraps this matcher and the given matcher using `JsHamcrest.Matchers.allOf()`.

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.CombinableMatcher`.

`CombinableMatcher.or`(*matcherOrValue*)
Wraps this matcher and the given matcher using `JsHamcrest.Matchers.anyOf()`.

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.CombinableMatcher`.

JsHamcrest.Description Class

class JsHamcrest.**Description**
Extends Object. Defines a textual description builder.

Description.**append**(*text*)
Appends *text* to this description.

Parameters **text** – Text to append to this description.

Returns this.

Description.**appendDescriptionOf**(*selfDescribingObject*)
Appends the description of a *self describing object* to this description.

Parameters **selfDescribingObject** – Any object that have a describeTo() function that accepts a JsHamcrest.Description object as argument.

Returns this.

Description.**appendList**(*start*, *separator*, *end*, *list*)
Appends the description of several *self describing objects* to this description.

Parameters

- **start** – Start string.
- **separator** – Separator string.
- **end** – End string.
- **list** – Array of *self describing objects*. These objects must have a describeTo() function that accepts a JsHamcrest.Description object as argument.

Returns this.

Description.**appendLiteral**(*literal*)
Appends a JavaScript language's *literal* to this description.

Parameters **literal** – Literal to append to this description.

Returns this.

Description.**appendValueList**(*start*, *separator*, *end*, *list*)
Appends an array of values to this description.

Parameters

- **start** – Start string.
- **separator** – Separator string.
- **end** – End string.
- **list** – Array of values to be described to this description.

Returns this.

Description.**get**()
Gets the current content of this description.

Returns Current content of this description.

JsHamcrest.SimpleMatcher Class

class JsHamcrest.**SimpleMatcher**(*{matches*, *describeTo*, *describeValueTo}*)
Extends Object. Defines a matcher that relies on the external functions provided by the caller in order to shape the current matching logic.

Below, an example of matcher that matches middle-aged people:

```
var middleAged = new JsHamcrest.SimpleMatcher({
    matches: function(person) {
        return person.age >= 40 && person.age <= 60;
    },
    describeTo: function(description) {
        description.append('middle-aged');
    }
});

// Matcher usage
middleAged.matches({name:'Gregory', age:50});  // Expected: true
middleAged.matches({name:'Jeniffer', age:27}); // Expected: false
```

Parameters

- **matches** – Function that provides the matching logic.
- **describeTo** – Function that describes this matcher to a JsHamcrest.Description.
- **describeValueTo** – *(Optional)* Function that describes the actual value under test. If not provided, the actual value will be described as a JavaScript literal.

SimpleMatcher.**describeTo**(*description*)
Describes this matcher's tasks to *description*.

Parameters **description** – Instance of JsHamcrest.Description.

Returns Nothing.

SimpleMatcher.**describeValueTo**(*actual*, *description*)
Describes *actual* to *description*.

Parameters

- **actual** – Actual value to be described.
- **description** – Instance of JsHamcrest.Description.

Returns Nothing.

SimpleMatcher.**matches**(*actual*)
Checks if this matcher matches *actual*.

Parameters **actual** – Actual value.

Returns True if the matcher matches the actual value; false otherwise.

See Also:

JsHamcrest API Reference

2.4.2 JsHamcrest.Matchers — Built-in Matchers

Built-in matcher library.

Collection Matchers

JsHamcrest.Matchers.**empty**()
The length of the actual value must be zero:

```
assertThat([], empty());
assertThat('', empty());
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**everyItem**(*matcherOrValue*)
The actual value should be an array and *matcherOrValue* must match all items:

```
assertThat([1,2,3], everyItem(greaterThan(0)));
assertThat([1,'1'], everyItem(1));
```

Parameters **matcherOrValue** – Instance of JsHamcrest.SimpleMatcher or a value.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**hasItem**(*matcherOrValue*)
The actual value should be an array and it must contain at least one value that matches *matcherOrValue*:

```
assertThat([1,2,3], hasItem(equalTo(3)));
assertThat([1,2,3], hasItem(3));
```

Parameters **matcherOrValue** – Instance of JsHamcrest.SimpleMatcher or a value.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**hasItems**(*MatchersOrValues...*)
The actual value should be an array and *matchersOrValues* must match at least one item:

```
assertThat([1,2,3], hasItems(2,3));
assertThat([1,2,3], hasItems(greaterThan(2)));
assertThat([1,2,3], hasItems(1, greaterThan(2));
```

Parameters **MatchersOrValues** – Matchers and/or values.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**hasSize**(*matcherOrValue*)
The length of the actual value value must match *matcherOrValue*:

```
assertThat([1,2,3], hasSize(3));
assertThat([1,2,3], hasSize(lessThan(5)));
assertThat('string', hasSize(6));
assertThat('string', hasSize(greaterThan(3)));
assertThat({a:1, b:2}, hasSize(equalTo(2)));
```

Parameters **matcherOrValue** – Instance of JsHamcrest.SimpleMatcher or a value.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**isIn**(*item...*)
The given array or arguments must contain the actual value:

```
assertThat(1, isIn([1,2,3]));
assertThat(1, isIn(1,2,3));
```

Parameters **item...** – Array or list of values.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`oneOf`**`()`
Alias to `isIn()` function.

Core Matchers

`JsHamcrest.Matchers.`**`allOf`**`(`*matchersOrValues...*`)`
All *matchesOrValues* must match the actual value. This matcher behaves pretty much like the JavaScript `&&` (and) operator:

```
assertThat(5, allOf([greaterThan(0), lessThan(10)]));
assertThat(5, allOf([5, lessThan(10)]));
assertThat(5, allOf(greaterThan(0), lessThan(10)));
assertThat(5, allOf(5, lessThan(10)));
```

Parameters **matchersOrValues** – Instances of `JsHamcrest.SimpleMatcher` and/or values.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`anyOf`**`(`*matchersOrValues*`)`
At least one of the *matchersOrValues* should match the actual value. This matcher behaves pretty much like the `||` (or) operator:

```
assertThat(5, anyOf([even(), greaterThan(2)]));
assertThat(5, anyOf(even(), greaterThan(2)));
```

Parameters **matchersOrValues** – Instances of `JsHamcrest.SimpleMatcher` and/or values.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`anything`**`()`
Useless always-match matcher:

```
assertThat('string', anything());
assertThat(null, anything());
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`both`**`(`*matcherOrValue*`)`
Combinable matcher where the actual value must match all the given matchers or values:

```
assertThat(10, both(greaterThan(5)).and(even()));
```

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.CombinableMatcher`.

`JsHamcrest.Matchers.`**`either`**`(`*matcherOrValue*`)`
Combinable matcher where the actual value must match at least one of the given matchers or values:

```
assertThat(10, either(greaterThan(50)).or(even()));
```

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.CombinableMatcher`.

`JsHamcrest.Matchers.`**`equalTo`**(*expected*)

The actual value must be equal to *expected*:

```
assertThat('10', equalTo(10));
```

Parameters **expected** – Expected value.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`is`**(*matcherOrValue*)

Delegate-only matcher frequently used to improve readability:

```
assertThat('10', is(10));
assertThat('10', is(equalTo(10)));
```

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`nil`**()

The actual value must be `null` or `undefined`:

```
var undef;
assertThat(undef, nil());
assertThat(null, nil());
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`not`**(*matcherOrValue*)

The actual value must not match *matcherOrValue*:

```
assertThat(10, not(20));
assertThat(10, not(equalTo(20)));
```

Parameters **matcherOrValue** – Instance of `JsHamcrest.SimpleMatcher` or a value.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`raises`**(*exceptionName*)

The actual value is a function and, when invoked, it should thrown an exception with the given name:

```
var MyException = function(message) {
    this.name = 'MyException';
    this.message = message;
};

var myFunction = function() {
    // Do something dangerous...
    throw new MyException('Unexpected error');
};
```

```
assertThat(myFunction, raises('MyException'));
```

Parameters **exceptionName** – Name of the expected exception.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**raisesAnything**()
The actual value is a function and, when invoked, it should raise any exception:

```
var myFunction = function() {
    // Do something dangerous...
    throw 'Some unexpected error';
};

assertThat(myFunction, raisesAnything());
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**sameAs**(*expected*)
The actual value must be the same as *expected*:

```
var number = 10, anotherNumber = number;
assertThat(number, sameAs(anotherNumber));
```

Parameters **expected** – Expected value.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**truth**()
The actual value must be any value considered truth by the JavaScript engine:

```
var undef;
assertThat(10, truth());
assertThat({}, truth());
assertThat(0, not(truth()));
assertThat('', not(truth()));
assertThat(null, not(truth()));
assertThat(undef, not(truth()));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**equivalentMap**(*expected*)
The actual value must be equivalent to *expected*. This works for maps with nested arrays and maps:

```
var firstMap = {"key" : 1, "key2" : "String"};
var equivMap = {"key" : 1, "key2" : "String"};

assertThat(firstMap, equivalentMap(equivMap));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**equivalentArray**(*expected*)
The actual value must be equivalent to *expected*. This works for arrays with nested arrays and maps:

```
var firstArray = [ 1, "String"];
var equivArray = [ 1, "String"];

assertThat(firstArray, equivalentArray(equivArray));
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

Number Matchers

`JsHamcrest.Matchers.`**`between`**(*start*)

The actual number must be between the given range (inclusive):

```
assertThat(5, between(4).and(7));
```

Parameters **start** – Range start.

Returns Builder object with an `end()` method, which returns a `JsHamcrest.SimpleMatcher` instance and thus should be called to finish the matcher creation.

`JsHamcrest.Matchers.`**`closeTo`**(*expected*[, *delta*])

The actual number must be close enough to *expected*, that is, the actual number is equal to a value within some range of acceptable error:

```
assertThat(0.5, closeTo(1.0, 0.5));
assertThat(1.0, closeTo(1.0, 0.5));
assertThat(1.5, closeTo(1.0, 0.5));
assertThat(2.0, not(closeTo(1.0, 0.5)));
```

Parameters

- **expected** – Expected number.
- **delta** – *(Optional, default=0)* Expected difference delta.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`divisibleBy`**(*divisor*)

The actual value must be divisible by *divisor*:

```
assertThat(21, divisibleBy(3));
```

Parameters **divisor** – Divisor.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`even`**()

The actual number must be even:

```
assertThat(4, even());
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`greaterThan`**(*expected*)

The actual number must be greater than *expected*:

```
assertThat(10, greaterThan(5));
```

Parameters **expected** – Expected number.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`greaterThanOrEqualTo`**`(`*expected*`)`
The actual number must be greater than or equal to *expected*:

```
assertThat(10, greaterThanOrEqualTo(5));
```

Parameters **expected** – Expected number.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`lessThan`**`(`*expected*`)`
The actual number must be less than *expected*:

```
assertThat(5, lessThan(10));
```

Parameters **expected** – Expected number.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`lessThanOrEqualTo`**`(`*expected*`)`
The actual number must be less than or equal to *expected*:

```
assertThat(5, lessThanOrEqualTo(10));
```

Parameters **expected** – Expected number.

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`notANumber`**`()`
The actual value must not be a number:

```
assertThat(Math.sqrt(-1), notANumber());
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`odd`**`()`
The actual number must be odd:

```
assertThat(5, odd());
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

`JsHamcrest.Matchers.`**`zero`**`()`
The actual number must be zero:

```
assertThat(0, zero());
assertThat('0', not(zero()));
```

Returns Instance of `JsHamcrest.SimpleMatcher`.

Object Matchers

JsHamcrest.Matchers.**bool**()
The actual value must be a boolean:

```
assertThat(true, bool());
assertThat(false, bool());
assertThat("text" not(bool()));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**func**()
The actual value must be a function:

```
assertThat(function() {}, func());
assertThat("text", not(func()));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**hasFunction**(*functionName*)
The actual value has a function with the given name:

```
var greeter = {
    sayHello: function(name) {
        alert('Hello, ' + name);
    }
};

assertThat(greeter, hasFunction('sayHello'));
```

Parameters **functionName** – Function name.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**hasMember**(*memberName*[, *matcherOrValue*])
The actual value has an attribute with the given name:

```
var greeter = {
    marco: 'polo',
    sayHello: function(name) {
        alert('Hello, ' + name);
    }
};

assertThat(greeter, hasMember('marco'));
assertThat(greeter, hasMember('sayHello'));
```

It's also possible to match the member's value if necessary:

```
assertThat(greeter, hasMember('marco', equalTo('polo')));
```

Parameters

- **memberName** – Member name.
- **matcherOrValue** – Matcher used to match the member's value.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**instanceOf**(*clazz*)
The actual value must be an instance of *clazz*:

```
assertThat([], instanceOf(Array));
```

Parameters **clazz** – Constructor function.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**number**()
The actual value must be a number:

```
assertThat(10, number());
assertThat('10', not(number()));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**object**()
The actual value must be an object:

```
assertThat({}, object());
assertThat(10, not(object()));
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**string**()
The actual value must be an string:

```
assertThat('10', string());
assertThat(10, not(string());
```

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**typeOf**(*typeName*)
The actual value must be of the given type:

```
assertThat(10, typeOf('number'));
assertThat({}, typeOf('object'));
assertThat('10', typeOf('string');
assertThat(function(){}, typeOf('function'));
```

Parameters **typeName** – Name of the type.

Returns Instance of JsHamcrest.SimpleMatcher.

Text Matchers

JsHamcrest.Matchers.**containsString**(*str*)
The actual string must have a substring equals to *str*:

```
assertThat('string', containsString('tri'));
```

Parameters **str** – Substring.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**emailAddress**()
The actual string must look like an e-mail address:

```
assertThat('user@domain.com', emailAddress());
```

Returns Instance of JsHamcrest.SimpleMatcher.

Warning: This matcher is not fully compliant with RFC2822 due to its complexity.

JsHamcrest.Matchers.**endsWith**(*str*)
The actual string must end with *str*:

```
assertThat('string', endsWith('ring'));
```

Parameters **str** – String.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**equalIgnoringCase**(*str*)
The actual string must be equal to *str*, ignoring case:

```
assertThat('str', equalIgnoringCase('Str'));
```

Parameters **str** – String.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**matches**(*regex*)
The actual string must match *regex*:

```
assertThat('0xa4f2c', matches(/\b0[xX][0-9a-fA-F]+\b/));
```

Parameters **regex** – Regular expression.

Returns Instance of JsHamcrest.SimpleMatcher.

JsHamcrest.Matchers.**startsWith**(*str*)
The actual string must start with *str*:

```
assertThat('string', startsWith('str'));
```

Parameters **str** – String.

Returns Instance of JsHamcrest.SimpleMatcher.

See Also:

JsHamcrest API Reference

2.4.3 JsHamcrest.Operators — Matcher Operators

Provides utilitary functions on top of matchers.

List Processing

JsHamcrest.Operators.**filter**(*array*, *matcherOrValue*)

Returns those items of *array* for which *matcherOrValue* matches:

```
var filtered = filter([0,1,2,3,4,5,6], even());
assertThat(filtered, equalTo([0,2,4,6]));

var filtered = filter([0,1,2,'1',0], 1);
assertThat(filtered, equalTo([1,'1']));
```

Parameters

- **array** – Array of items to be filtered.
- **matcherOrValue** – Instance of JsHamcrest.SimpleMatcher or a value.

Returns Filtered array.

Unit Testing

JsHamcrest.Operators.**assert**(*actualValue*, *matcherOrValue*[, *{fail*, *pass*, *message}*])

Generic assert function to be used for easy integration with testing frameworks. Usage example:

```
// Add the following method to your testing framework...

function iAssertThat(actualValue, matcherOrValue, message) {
    return JsHamcrest.Operators.assert(actualValue, matcherOrValue, {
        message: message,
        fail: function(failMessage) {
            // Forward the call to the appropriate method provided by the testing framework
            myTestingFramework.fail(failMessage);
        },
        pass: function(passMessage) {
            // Forward the call to the appropriate method provided by the testing framework
            myTestingFramework.pass(passMessage);
        }
    });
}

// ...and then you'll be able to leverage the power of JsHamcrest in your test cases
result = iAssertThat(50, between(0).and(100));

result.passed // Output: true
result.get()  // Output: "50 between 0 and 100: Success"
```

Parameters

- **actualValue** – Actual value.
- **matcherOrValue** – Instance of JsHamcrest.SimpleMatcher or a value.
- **fail** – *(Optional)* Callback function to be called when *actualValue* doesn't match *matcherOrValue*.
- **pass** – *(Optional)* Callback function to be called when *actualValue* does match *matcherOrValue*.
- **message** – *(Optional)* Text that describes the assertion on an even higher level.

Returns Instance of `JsHamcrest.Description` with the assertion description. Also, the result of the assertion (success or failure/error) can be accessed through the `passed` attribute.

`JsHamcrest.Operators.`**`callTo`**(*func*[, *arg...*])

Returns a zero-args function that calls the function *func* with the given *args*:

```
var func = callTo(parseInt, "2");
assertThat(func(), sameAs(2));
```

This is specially useful when used along with `JsHamcrest.Matchers.raises()` or `JsHamcrest.Matchers.raisesAnything()`:

```
assertThat(callTo(myFunc, arg1, arg2), raisesAnything());
```

Parameters

- **func** – Function to delegate calls to.
- **arg** – Optional arguments to *func*.

Returns Function that delegates calls to *func*.

See Also:

JsHamcrest API Reference

2.4.4 `JsHamcrest.Integration` — Third-Party Integration Utilities

Provides functions to make it easy to integrate JsHamcrest with other popular JavaScript frameworks.

`JsHamcrest.Integration.`**`copyMembers`**([*source*], *target*)

Copies all members of an object to another.

Parameters

- **source** – *(Optional)* Source object. If not provided, this function will copy all members of `JsHamcrest.Matchers` and `JsHamcrest.Operators` to *target*.
- **target** – Target object.

Returns Nothing.

`JsHamcrest.Integration.`**`installMatchers`**(*matchersNamespace*)

Copies all members of *matchersNamespace* to `JsHamcrest.Matchers`.

Parameters **matchersName** – Namespace that contains the matchers to be installed.

Returns Nothing.

`JsHamcrest.Integration.`**`installOperators`**(*operatorsNamespace*)

Copies all members of *operatorsNamespace* to `JsHamcrest.Operators`.

Parameters **operatorsNamespace** – Namespace that contains the operators to be installed.

Returns Nothing.

Unit Testing Frameworks

JsHamcrest is perfect to enhance your JavaScript testing code.

Dummy Functions For Easy Prototyping

JsHamcrest.Integration.**WebBrowser**()

Uses the web browser's `alert()` function to display the assertion results. Great for quick prototyping:

```
<!-- Activate dummy integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.WebBrowser();

    var calc = new MyCalculator();
    assertThat(calc.add(2,3), equalTo(5));
</script>
```

Returns Nothing.

JsHamcrest.Integration.**Rhino**()

Uses the Rhino's `print()` function to display the assertion results. Great for quick prototyping:

```
js> load('jshamcrest.js')
js> JsHamcrest.Integration.Rhino();
js>
js> var calc = new MyCalculator();
js> assertThat(calc.add(2,3), equalTo(5));
[SUCCESS] Expected equal to 5: Success
```

Returns Nothing.

Jasmine – BDD for your JavaScript

JsHamcrest.Integration.**jasmine**(*{scope}*)

Integrates JsHamcrest with Jasmine.

The following code is an example on how to set up your project:

```
<!-- Jasmine dependencies -->
<link rel="stylesheet" type="text/css" href="jasmine.css">
<script type="text/javascript" src="jasmine.js"></script>
<script type="text/javascript" src="jasmine-html.js"></script>

<!-- Activate Jasmine integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.jasmine();

    // Same as above
    JsHamcrest.Integration.jasmine({
        scope: this
    });

    describe('Calculator', function() {
        var calc;

        beforeEach(function() {
            calc = new MyCalculator();
        });
```

```
        it('should add two numbers', function() {
            assertThat(calc.add(2,3), equalTo(5));
        });
    });
</script>

<script type="text/javascript">
   jasmine.getEnv().addReporter(new jasmine.TrivialReporter());
   jasmine.getEnv().execute();
</script>
```

Parameters **scope** – *(Optional, default=this)* Copies all matchers to the given scope.

Returns Nothing.

JsTestDriver – Remote JavaScript Console

JsHamcrest.Integration.**JsTestDriver**(*{scope}*)

Integrates JsHamcrest with JsTestDriver. Instructions on how to set up your project:

1. Let's assume your project root directory have a `lib` directory to keep your project's dependencies. In this case, copy the `jshamcrest.js` file to that directory;
2. Create a file `plugin/jshamcrest-plugin.js` in your project root directory and put one of the following lines inside it:

   ```
   JsHamcrest.Integration.JsTestDriver();

   // Same as above
   JsHamcrest.Integration.JsTestDriver({
       scope:this
   });
   ```

3. Finally, edit the `jsTestDriver.conf` file as follows:

   ```
   load:
     - lib/*.js
     - <source directory>
     - <test cases directory>
     - plugin/*.js
   ```

That's it. Your test cases should now have access to JsHamcrest functions:

```
CalculatorTest = TestCase('CalculatorTest');

CalculatorTest.prototype.testAdd = function() {
    var calc = new MyCalculator();
    assertThat(calc.add(2,3), equalTo(5));
};
```

Parameters **scope** – *(Optional, default=this)* Copies all matchers to the given scope.

Returns Nothing.

JsUnitTest – JavaScript Unit Testing Framework

JsHamcrest.Integration.**JsUnitTest**(*{scope}*)

Integrates JsHamcrest with JsUnitTest.

The following code is an example on how to set up your project:

```
<!-- JsUnitTest and dependencies -->
<script type="text/javascript" src="jsunittest.js"></script>

<!-- Activate JsUnitTest integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.JsUnitTest();

    // Same as above
    JsHamcrest.Integration.JsUnitTest({
        scope: JsUnitTest.Unit.Testcase.prototype
    });
</script>

<script type="text/javascript">
    new Test.Unit.Runner({
        setup: function() {
        },

        tearDown: function() {
        },

        testAdd: function() { with(this) {
            var calc = new MyCalculator();
            assertThat(calc.add(2,3), equalTo(5));
        }}
    }, {'testLog':'myLog'});
</script>
```

Parameters **scope** – *(Optional, default=JsUnitTest.Unit.Testcase.prototype)* Copies all matchers to the given scope.

Returns Nothing.

jsUnity – Lightweight JavaScript Testing Framework

JsHamcrest.Integration.**jsUnity**(*{scope, attachAssertions}*)

Integrates JsHamcrest with jsUnity.

The following code is an example on how to set up your project:

```
<!-- jsUnity and dependencies -->
<script type="text/javascript" src="jsunity.js"></script>

<!-- Activate jsUnity integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    function CalculatorTestSuite() {
        function testAdd() {
            var calc = new MyCalculator();
            assertThat(calc.add(2,3), equalTo(5));
```

```
        }
    }

    // Activate the jsUnity integration
    JsHamcrest.Integration.jsUnity();

    // Same as above
    JsHamcrest.Integration.jsUnity({
        scope: jsUnity.env.defaultScope,
        attachAssertions: false
    });

    var results = jsUnity.run(CalculatorTestSuite);
</script>
```

Parameters

- **scope** – *(Optional, default=jsUnity.env.defaultScope)* Copies all matchers to the given scope.
- **attachAssertions** – *(Optional, default=false)* Whether JsHamcrest should also copy jsUnity's assertion functions to the given scope.

Returns Nothing.

QUnit – JavaScript Test Suite

`JsHamcrest.Integration.`**`QUnit`**(*{scope}*)

Integrates JsHamcrest with QUnit.

The following code is an example on how to set up your project:

```
<!-- QUnit and dependencies -->
<script type="text/javascript" src="jquery.js"></script>

<!-- Activate QUnit integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.QUnit();

    // Same as above
    JsHamcrest.Integration.QUnit({
        scope: this
    });

    $(document).ready(function(){
        test('Calculator should add two numbers', function() {
            var calc = new MyCalculator();
            assertThat(calc.add(2,3), equalTo(5));
        });
    });
</script>

<!-- QUnit and dependencies -->
<script type="text/javascript" src="testrunner.js"></script>
```

Parameters **scope** – *(Optional, default=this)* Copies all matchers to the given scope.

Returns Nothing.

screw-unit – JavaScript BDD Framework

JsHamcrest.Integration.**screwunit**(*{scope}*)

Integrates JsHamcrest with screw-unit.

The following code is an example on how to set up your project:

```
<!-- screw-unit and dependencies -->
<script type="text/javascript" src="jquery-1.2.6.js"></script>
<script type="text/javascript" src="jquery.fn.js"></script>
<script type="text/javascript" src="jquery.print.js"></script>
<script type="text/javascript" src="screw.builder.js"></script>
<script type="text/javascript" src="screw.matchers.js"></script>
<script type="text/javascript" src="screw.events.js"></script>
<script type="text/javascript" src="screw.behaviors.js"></script>
<link rel="stylesheet" type="text/css" href="screw.css" />

<!-- Activate screw-unit integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.screwunit();

    // Same as above
    JsHamcrest.Integration.screwunit({
        scope: Screw.Matchers
    });

    Screw.Unit(function() {
        describe('Using JsHamcrest assertions in Screw.Unit', function() {
            it('should succeed', function() {
                assertThat(5, between(0).and(10), 'This assertion must succeed');
            });

            it('should fail', function() {
                assertThat([], not(empty()), 'This assertion must fail');
            });
        });
    });
</script>
```

Parameters **scope** – *(Optional, default=Screw.Matchers)* Copies all matchers to the given scope.

Returns Nothing.

YUITest – JavaScript Unit Testing Framework

JsHamcrest.Integration.**YUITest**(*{scope}*)

Integrates JsHamcrest with YUITest.

The following code is an example on how to set up your project:

```
<!-- YUITest and dependencies -->
<script type="text/javascript" src="yahoo-dom-event/yahoo-dom-event.js"></script>
<script type="text/javascript" src="yuilogger/logger.js"></script>
<script type="text/javascript" src="yuitest/yuitest.js"></script>
```

```
<!-- Activate YUITest integration -->
<script type="text/javascript" src="jshamcrest.js"></script>
<script type="text/javascript">
    JsHamcrest.Integration.YUITest();

    // Same as above
    JsHamcrest.Integration.YUITest({
        scope: this
    });
</script>

<script type="text/javascript">
    CalculatorTestCase = new YAHOO.tool.TestCase({
        name: 'Calculator test case',

        setUp: function() {
        },

        teardown: function() {
        },

        testAdd: function() {
            var calc = new MyCalculator();
            Assert.that(calc.add(2,3), equalTo(5));
        }
    });
</script>
```

Parameters **scope** – *(Optional, default=this)* Copies all matchers to the given scope.

Returns Nothing.

See Also:

JsHamcrest API Reference

2.5 Getting Involved

As with any open source project, there are several ways you can help:

- Report bugs, feature requests and other issues in the issue tracking system;
- Submit patches to reported issues (both those you find, or that others have filed);
- Help with the documentation by pointing out areas that are lacking or unclear, and if you are so inclined, submitting patches to correct it;
- Improve the overall project quality by suggesting refactorings and improving the test cases. A great way to learn – and in turn give value back to the community – is to review someone else's code. So, we invite you to review ours;
- Write about JsHamcrest in your blog or personal web site. Let your friends know about this project.

Your participation is much appreciated. Keep up with JsHamcrest development on GitHub.

2.5.1 How Do I Join The Team?

JsHamcrest is a mature project and it's probably not going to get lots of new features. But those that the developers notice participating to a high extent will be invited to join the team as a committer.

This is as much based on personality and ability to work with other developers and the community as it is with proven technical ability. Being unhelpful to other users, or obviously looking to become a committer for bragging rights and nothing else is frowned upon, as is asking to be made a committer without having contributed sufficiently to be invited.

2.5.2 Contact Information

Author Daniel Fernandes Martins <daniel@destaquenet.com>

Company Destaquenet Technology Solutions

2.6 Changelog

Like any other piece of software, JsHamcrest is evolving at each release. Here you can track our progress:

Version 0.6.7 *(Sep 11, 2011)*

- Updated `JsHamcrest.Matchers.hasSize()` function to make it work with objects;

Version 0.6.6 *(Jun 23, 2011)*

- Fixed broken logic of `JsHamcrest.areArraysEqual()` function;

Version 0.6.5 *(Jun 20, 2011)*

- Description of the Function literal now includes function name when available;

Version 0.6.4 *(May 25, 2011)*

- Updated `hasMember` matcher to make it possible to also match considering the member's value;

Version 0.6.3 *(Apr 23, 2011)*

- Integration with Jasmine;

Version 0.6.2 *(Sep 30, 2010)*

- Added `JsHamcrest.Operators.callTo()` function, which makes `JsHamcrest.Matchers.raises()` and `JsHamcrest.Matchers.raisesAnything()` easier to use.

Version 0.6.1 *(Dec 22, 2009)*

- Fixed the code that removes useless stacktrace entries when using JsHamcrest with js-test-driver;

Version 0.6 *(Nov 22, 2009)*

- Build script rewritten from scratch with Python;
- Documentation rewritten from scratch with Sphinx;
- Added a couple of new matchers;
- New `JsHamcrest.Operators` namespace;
- Several improvements and fixes;

Version 0.5.2 *(Jul 19, 2009)*

- JSLint fixes;
- Integration with screw-unit (thanks, Chris Leishman!)

Version 0.5.1 *(Jul 11, 2009)*

- Added a new function to `JsHamcrest.SimpleMatcher` that provides a better way to describe the actual value;
- Added a couple of new matchers;
- Improvements on test code;

Version 0.4 *(May 26, 2009)*

- Integration with JsTestDriver and JsUnity;
- Some code cleanup;

Version 0.3 *(May 23, 2009)*

- Integration with QUnit;
- Improvements on integration code;
- Added some integration tests;
- Documentation improvements;

Version 0.2 *(May 22, 2009)*

- Fixed small API documentation issues;
- Download page link in README;
- `JsHamcrest.Operators.assert()` function now accepts a string that describes the assertion;
- Refactoring to allow easy integration with JavaScript testing frameworks;

Version 0.1 *(April 21, 2009)*

- First public release;

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX

j

INDEX