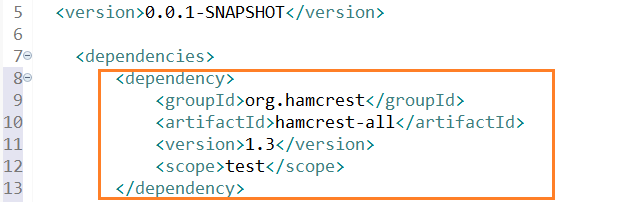
# **Hamcrest Assertion with Rest Assured**

Assertions are nothing but checkpoints are verifications where you will be checking the values currently present and the values without we are expecting our matching or not.  
If the current value and the expected value match then the assertion passes when the assertion passes nothing happens but when an assertion fails it will fail the test case.  
  
**In one test case, you can have multiple assertion statements. If you write a test case without an assertion then there is a chance that your test case will never fail.**  
  
With respect to Rest Assured, we are going to see Hamcrest assertions majorly. We can also use the Junit for the assertions.

##### **Install hamcrest Assertion :**

* Go to Hamcreset maven repo: <https://mvnrepository.com/artifact/org.hamcrest/hamcrest-all>
* Choose the latest version ( I am choosing 1.3)
* Copy the dependency for Java
* <dependency>
* <groupId>org.hamcrest</groupId>
* <artifactId>hamcrest-all</artifactId>
* <version>1.3</version>
* <scope>test</scope>

</dependency>​

* **Paste the same dependency in your pom.xml file.**  
  
* You should able to see the Hamcrest jar file under maven dependencies  
  

I am writing the same code using the hamcrest assertion below. You will not see much difference. The body() function accepts two parameters

1. The JsonPath locator/XML path locator
2. The hamcrest assertion with an expected value.

To use hamcrest assertion please import the Matchers class static member

import static org.hamcrest.Matchers.\*;

The test using the hamcrest matchers

import org.junit.jupiter.api.Test;

import io.restassured.RestAssured;

import static org.hamcrest.Matchers.\*;

public class AssertionExample {

@Test

void test() {

RestAssured.given().when()

.get("https://chercher.tech/sample/api/product/read?id=3793")

.then()

.body("id[0]", equalTo("3793"));

}

}

If you ask me, will there be a difference in execution, the answer is NO.

**You can use the hamcrest assertion when there is no need to extract the values for further evaluations. If you need the value to be stored for further evaluation then better go for Junit assertion.**

**Hamcrest assertions**

In hamcrest assertion, there are different types of verifications, we can verify whether a string is equal to another string, a list of items is equal to another list of items, and so on.

* Number related assertions
* String related assertions
* Collection related assertion
* not assertion

### Number related assertions:

In number-related assertions, we can assert whether the retrieved number is greater than the expected number, less than the expected number, equal, greater than equal to, or less than equal.

* **equalTo:**checks whether the retrieved number is equal to the expected number

.body("id[0]", equalTo(3793));

* **greaterThan :** checks extracted number is greater than the expected number

.body("id[0]", greaterThan(3000));​

* **greaterThanOrEqualTo:**checks whether the extracted number is greater than equal to the expected number.

.body("id[0]", greaterThanOrEqualTo(3000));​

* **lessThan:**checks whether the extracted number is lesser than the expected number

.body("id[0]", lessThan(3000));​

* **lessThanOrEqualTo:**checks whether the extracted number is lesser than or equal to the expected number

.body("id[0]", lessThanOrEqualTo(3000));​

### String related Assertions:

* **equalTo:**Checks whether the extracted string is equal to the expected string

.body("name[0]", equalTo("Samsung 43 inch TV"));​

* **equalToIgnoringCase:**Checks whether the extracted string is equal to the expected string without considering the case

.body("name[0]", equalToIgnoringCase("SAMSUNG 43 INCH tv"));​

* **equalToIgnoringWhiteSpace:**Checks whether the extracted string is equal to the expected string by ignoring the white spaces present in the string

.body("name[0]", equalToIgnoringWhiteSpace("Samsung 43 inch TV"));​

* **isEmptyOrNullString :** Checks whether the extracted string is an empty string or Null

.body("name[0]", isEmptyOrNullString());​

* **containsString:**Checks whether the extracted string contains the expected string as a substring

.body("name[0]", containsString("Samsung"));​

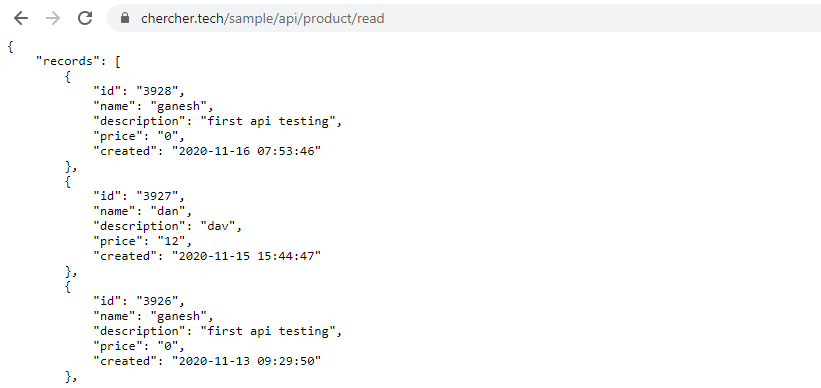
* **startsWith:**Checks whether the extracted string is starting with a given string or character

.body("name[0]", startsWith("S"));​

* **endsWith:**Checks whether the extracted string is ending with a given string or character

.body("name[0]", endsWith("TV"));​

### Collection Related Assertion in hamcrest:



@Test

void test() {

RestAssured.given().when()

.get("https://chercher.tech/sample/api/product/read")

.then()

.body("records.id", hasItem("3928"));

}

* ***equalTo:****Checks whether the extracted collection is a vehicle into the expected correction*
* ***hasItem :****Checks whether the extracted collection has a given value*

*.body("records.id", hasItem("3928"));​*

* ***hasItems:****Checks whether the extracted collection has expected collection or subset*

*.body("records.id", hasItems("3928", "3927", "3926"));​*

* ***hasKey :****Checks whether the extracted map has an expected key*

*.body("records[0]", hasKey("id"));​*

* ***hasValue :****Checks whether the extracted map has expected value*

*.body("records[0]", hasValue("3928"));​*

* ***hasEntry:****hasKey and hasValue will not check with which key or value they are associated, but if you want to check whether the right field has the right value then we have to use hasEntry; basically it checks map key value in the extracted map.*

*.body("records[0]", hasEntry("id", "3928"));*

* ***empty :****Checks whether the extracted collection has no items(below non-existing locator given)*

*.body("records.id123", empty());​*

### not Assertion in hamcrest:

Similar to the above assertions there is one more assertion called not assertion. Not assertion will not help you to assert anything but you have to use other assertions along with the not assertion.

The not assertion inverts the meaning of the other assertions.

For example, if you are using equalTo() assertion, it will basically check whether a number is equal to the expected number, but when you use not assertion along with equalTo() assertion then you are checking the number should not be equal to the expected number.

@Test

void test() {

RestAssured.given().when()

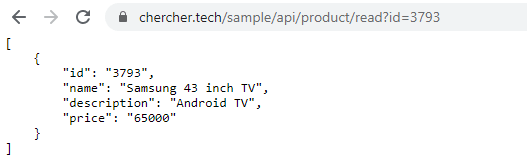
.get("https://chercher.tech/sample/api/product/read")

.then()

.body("records.id[0]", not(equalTo("3928")));

}

**Multiple assert statements in a test case**

In the above programs adding a single assertion to the test case. Sometimes you might need to add more than 1 assertion in a test case.  


@Test

void test() {

RestAssured.given().when()

.get("https://chercher.tech/sample/api/product/read?id=3793")

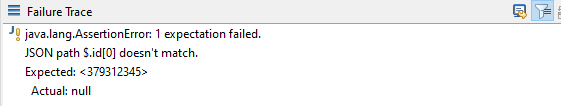
.then()

.body("$.id[0]", equalTo(379312345)) // will fail

.body("$.name[0]", equalTo("Samsung 43 inch TV eee")) // will fail

.body("$.description[0]", greaterThan("Android TV eee")); // will fail

}

The test failed at **.body("$.id[0]", equalTo(379312345))** because the actual value is 3793; but if you observe the below output, the last two lines did not execute; because they also should have failed if they got executed.  
  
  
To execute all the assertions present in the test case we have to combine all the assertion into a single body, just like below

@Test

void test() {

RestAssured.given().when()

.get("https://chercher.tech/sample/api/product/read?id=3793")

.then()

.body(

"$.id[0]", equalTo(379312345),

"$.name[0]", equalTo("Samsung 43 inch TV eee"),

"$.description[0]", equalTo("Android TV eee")

);

}

