JSONPath Library

In this lesson, we will learn to validate responses using JSONPath library.

We'll cover the following

- What is JSONPath?
- JSONPath dependency
- JSONPath operators
- Parse JSON using JSONPath operators
- Parsing a JSON file in the Java

What is `JSONPath`?

`JSONPath` is a query language that helps us in parsing the `JSON` data, which can be used for validation or assertions in a test.

`JSONPath` dependency

To use `JSONPath`, we have to include its dependency on our project.

`Gradle`

For a `Gradle` project, add the following dependency in *build.gradle*:

```
compile group: 'com.jayway.jsonpath', name: 'json-path', version: '2.4.0'
```

`Maven`

For a `Maven` project, add the following dependency in the *pom.xml* file:

```
<dependency>
    <groupId>com.jayway.jsonpath</groupId>
    <artifactId>json-path</artifactId>
    <version>2.4.0</version>
</dependency>
```

`JSONPath` operators

Operator	Description
`$`	This is the root element and starting point for all path expressions.
`@`	This is a filter predicate for the current node.
`*`	This is a wildcard operator. It will return all objects regardless of their names or indexes.
`..`	This searches for the specified name recursively.
`.<name>`	This is used for denoting the child element of the current element by name or index using dots `.`.
`['<name>' (, '<name>')]`	This is used for denoting the child element of the current element by name or index using brackets `[` `]`.
`[<number> (, <number>)]`	This is used for denoting the array index of the element.
`[start:end]`	This is used for slicing the array based on start and end indexes.
`[?(<expression>)]`	This is a filter expression and must evaluate to a boolean value.

Parse `JSON` using `JSONPath` operators

Let’s parse the `JSON` below using `JSONPath` operators.

```
{
    "firstName": "John",
    "lastName": "doe",
    "age": 26,
    "id": 1,
    "address": [
    {
        "addressType": "Home",
        "country": "USA",
        "city": "Chicago",
        "zipCode": "60007"
    },
    {
        "addressType": "Office",
        "country": "USA",
        "city": "New York",
        "zipCode": "10018"
    }],
    "phoneNumbers": [
    {
        "type": "office",
        "number": "123-456-78910"
    },
    {
        "type": "home",
        "number": "0123-456-91011"
    }]
}
```

Here are few examples on parsing a `JSON` file using operators:

Syntax	Evaluation Result
`$..city`	List of all cities
`$.address.*`	All addresses
`$.address..zipCode`	All zip codes
`$.address[1]`	The second address

`$..address[:2]`	All addresses from index 0(inclusive) and 2 (exclusive)
`$..address[1:]`	Last address from tail
`$..address[?(@.city)]`	All addresses with their cities
`$..*`	Give me everything

Parsing a `JSON` file in the Java

Once you have added the dependency of `JSONPath` in your `JAVA` project, you can use it to parse a JSON file as shown below:

```
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import static org.hamcrest.Matchers.equalTo;
import org.testng.annotations.Test;
import static org.testng.Assert.assertEquals;
import static org.testng.Assert.assertTrue;

import io.restassured.response.Response;
import io.restassured.RestAssured;
import java.util.List;
import io.restassured.path.json.JsonPath;

public class GETRequestTest {

        private static Logger LOG = LoggerFactory.getLogger(GETRequestTest.class);

        @Test
        public void testGetAllStudentRecords() {

        String url = "http://ezifyautomationlabs.com:6565/educative-rest/students";

            LOG.info("Step - 1 : Send GET Request");
                Response response = RestAssured.given().get(url).andReturn();

                LOG.info("Step - 2 : Print the JSON response body");
                response.getBody().prettyPrint();

                LOG.info("Step - 3 : Assert StatusCode = 200");
                assertTrue(response.getStatusCode()==200);

                LOG.info("Step - 4 :Create a JSONPath object");
                JsonPath jpath = response.jsonPath();

                LOG.info("Step - 5 :Use JsonPath to get the list of all Student's first name");
                LOG.info("Step - 5 :Use JsonPath to get the list of all Student's first_name");
                // In java code you DO NOT have to write expression starting with `$`
                List<String> firstNames = jpath.get("first_name");

                LOG.info("List of all Student's first name: " +firstNames.toString());

            LOG.info("Step - 6 :Use JsonPath to get the first_name of the first Student record");
                String firstName = jpath.get("first_name[0]");
                LOG.info("Print the first name of the first Student record: " +firstName.toString(

        }
}
```

In the next lesson, we'll learn about the Hamcrest library, which is also used in validating responses.