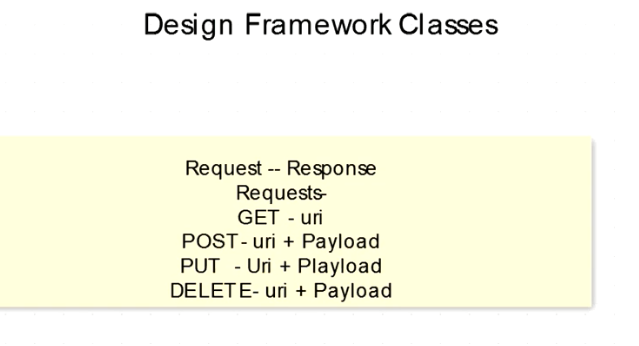
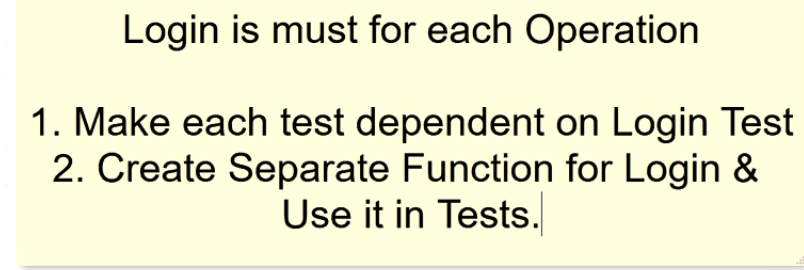
Rest Assured Current Verison -3.2.0



**REST Assured**

REST Assured can be used to test XML as well as JSON based web services. REST Assured can be integrated with JUnit and TestNG frameworks for writing test cases for our application.

REST Assured supports POST, GET, PUT, DELETE, OPTIONS, PATCH, and HEAD requests and can be used to validate and verify the response of these requests.

REST Assured is implemented in Groovy and uses the builder pattern to create requests, set headers, parse the response and then match them with expected data. It uses Hamcrest Matchers for comparing actual response with the expected response.

Note: equalTo and hasItems are Hamcrest matchers which you should statically import from org.hamcrest.Matchers.

### **Returning floats and doubles as BigDecimal**

You can configure Rest Assured and JsonPath to return BigDecimal's instead of float and double for Json Numbers. For example consider the following JSON document:

{

"price":12.12

}

By default you validate that price is equal to 12.12 as a float like this:

get("/price").then().body("price", is(12.12f));

**Hamcrest is library to get Pattern and put few more assertions**

Note that the fluent API used by REST Assured supports the familiar Given/When/Then syntax from [behavior-driven development](https://en.wikipedia.org/wiki/Behavior-driven_development) (BDD), resulting in a test that is easy to read and takes care of everything (setup, execution, and verification) with just a single line of code.

The *hasSize()* Hamcrest matcher counts the number of circuits—that's why you needed to add Hamcrest as a static import. The Hamcrest library contains a collection of matchers that allow you to create verifications of all kinds while keeping them readable.

The verification part of the test does the following:

1. Captures the (JSON) response of the API call
2. Queries for all elements called circuitId using the Groovy GPath expression *"MRData.CircuitTable.Circuits.circuitId"*
3. Verifies (using the aforementioned Hamcrest matcher) that the resulting collection of circuitId elements has size 20

There are Hamcrest matchers for a large number of different checks, including *equalTo()*for equality, *lessThan()* and *greaterThan()* for comparison, *hasItem()* to check whether a collection contains a given element, and many more. Reference the [Hamcrest library](http://hamcrest.org/JavaHamcrest/)

### Validating technical response data

With REST Assured, you can not only verify response body contents, but also check the correctness of technical response data, such as the HTTP response status code, the response content type, and other response headers. The example below checks that:

* The response status code is equal to 200.
* The response content type (telling the receiver of the response how to interpret the response body) equals "application/json."
* The value of the response header "Content-Length" equals "4567."

### JSON Schema validation

From version 2.1.0 REST Assured has support for [Json Schema](http://json-schema.org/) validation.

# REST Assured Schema Validation Setup

Before we can implement any REST Assured Schema Validation, we need to make sure we have the proper imports in our project. If you are just doing **XML validation**, then everything is already included in REST Assured. If you are doing **JSON validation**, you need to add the REST Assured Schema Validation to the dependencies in your build file:

1. compile 'io.rest-assured:json-schema-validator:3.0.3'

# For XML Schema

Firstly you need to generate an XML Schema definition for your endpoint. To do that, take the xml response from the endpoint and paste it into the [XSD generator](https://www.freeformatter.com/xsd-generator.html#ad-output)

In java Doc

given() returns **RequestSpecification**

when() returns **RequestSender**

then() returns **Returns a validatable response that's lets you validate the response.**

## OAuth

In order to use OAuth 1 and OAuth 2 (for query parameter signing) you need to add [Scribe](https://github.com/fernandezpablo85/scribe-java) to your classpath (if you're using version 2.1.0 or older of REST Assured then please refer to the [legacy](https://github.com/rest-assured/rest-assured/wiki/Usage_Legacy#OAuth)documentation). In Maven you can simply add the following dependency:

<dependency>

<groupId>com.github.scribejava</groupId>

<artifactId>scribejava-apis</artifactId>

<version>2.5.3</version>

<scope>test</scope>

</dependency>

### OAuth 2

Since version 2.5.0 you can use OAuth 2 authentication without depending on [Scribe](https://github.com/rest-assured/rest-assured/wiki/Usage#oauth):

given().auth().oauth2(accessToken). ..

This will put the OAuth2 accessToken in a header. To be more explicit you can also do:

given().auth().preemptive().oauth2(accessToken). ..

There reason why given().auth().oauth2(..) still exists is for backward compatibility (they do the same thing). If you need to provide the OAuth2 token in a query parameter you currently need [Scribe](https://github.com/rest-assured/rest-assured/wiki/Usage#oauth) in the classpath. Then you can do like this:

given().auth().oauth2(accessToken, OAuthSignature.QUERY\_STRING). ..

## Headers

given().header("MyHeader", "Something").and(). ..

given().headers("MyHeader", "Something", "MyOtherHeader", "SomethingElse").and(). ..

You can also specify a multi-value headers like this:

given().header("headerName", "value1", "value2"). ..

This will create two headers, headerName: value1 and headerName: value2.

#### Header Merging/Overwriting

By default headers are merged. So for example if you do like this:

given().header("x", "1").header("x", "2"). ..

## Cookies

In its simplest form you specify cookies like this:

given().cookie("username", "John").when().get("/cookie").then().body(equalTo("username"));

You can also specify a multi-value cookie like this:

given().cookie("cookieName", "value1", "value2"). ..

This will create two cookies, cookieName=value1 and cookieName=value2.

You can also specify a detailed cookie using:

Cookie someCookie = new Cookie.Builder("some\_cookie", "some\_value").setSecured(true).setComment("some comment").build();

given().cookie(someCookie).when().get("/cookie").then().assertThat().body(equalTo("x"));

### Multi-value parameter

Multi-value parameters are parameters with more then one value per parameter name (i.e. a list of values per name). You can specify these either by using var-args:

given().param("myList", "value1", "value2"). ..

or using a list:

List<String> values = new ArrayList<String>();

values.add("value1");

values.add("value2");

given().param("myList", values). ..

### No-value parameter

You can also specify a query, request or form parameter without a value at all:

given().param("paramName"). ..

### Path parameters

You can also specify so called path parameters in your request, e.g.

post("/reserve/{hotelId}/{roomNumber}", "My Hotel", 23);

These kinds of path parameters are referred to "unnamed path parameters" in REST Assured since they are index based (hotelId will be equal to "My Hotel" since it's the first placeholder).

You can also use named path parameters:

given().

pathParam("hotelId", "My Hotel").

pathParam("roomNumber", 23).

when().

post("/reserve/{hotelId}/{roomNumber}").

then().

..

Path parameters makes it easier to read the request path as well as enabling the request path to easily be re-usable in many tests with different parameter values.

As of version 2.8.0 you can mix unnamed and named path parameters:

given().

pathParam("hotelId", "My Hotel").

when().

post("/reserve/{hotelId}/{roomNumber}", 23).

then().

..

## Param Config

[ParamConfig](http://static.javadoc.io/io.rest-assured/rest-assured/3.2.0/io/restassured/config/ParamConfig.html) allows you to configure how different parameter types should be updated on "collision". By default all parameters are merged so if you do:

given().queryParam("param1", "value1").queryParam("param1", "value2").when().get("/x"). ...

**Delegated Authorizaion**: like Access from Gdrive from another like any tax website to sabve doc in Gdrive without any access in gmail <https://www.youtube.com/watch?v=6uQ44YxUqUs>

## Basic Authentication with the API

Let’s start with the standard way of configuring Basic Authentication on the HttpClient – via a CredentialsProvider:

**CredentialsProvider** provider = **new** **BasicCredentialsProvider**();

**UsernamePasswordCredentials** credentials = **new** **UsernamePasswordCredentials**("user1", "user1Pass");

provider.setCredentials(**AuthScope**.ANY, credentials);

**HttpClient** client = **HttpClientBuilder**.create().setDefaultCredentialsProvider(provider).build();

**HttpResponse** response = client.execute(**new** **HttpGet**(URL\_SECURED\_BY\_BASIC\_AUTHENTICATION));

**int** statusCode = response.getStatusLine().getStatusCode();

assertThat(statusCode, equalTo(**HttpStatus**.SC\_OK));



RequestSpecification requestSpecification = RestAssured.*given*();

requestSpecification.auth().preemptive().basic("username", "password");

requestSpecification.contentType(ContentType.***JSON***);

**Oauth 2 in Rest Assured**

**public** **void** **uploadData**() **throws** IOException {

String jsonString = IOUtils.toString(DataLoader.class.getClassLoader().getResourceAsStream(**"daily\_feed.json"**), Charset.forName(**"UTF-8"**));

Response response = given().auth().oauth2(accessToken)

.contentType(**"application/json"**)

.body(jsonString.toString())

.when()

.post(**"/feed/create-feed/daily"**);

String responseBody = response.getBody().asString();

**if** (response.getStatusCode() >= 200 && response.getStatusCode() <= 299) {

logger.info(**"Create Daily Feed Response = "** + responseBody);

} **else** {

logger.error(**"Error creating daily feed = {}"**, responseBody);

}

}

|  |  |
| --- | --- |
|  | Passing the OAuth2 AccessToken in request. |