Contents

[Consumer Side – Generate Contract 1](#_Toc30496479)

[Use Case 1: Provider API is ready 1](#_Toc30496480)

[Generate Contract for PDSM API from consumer side 3](#_Toc30496481)

[Sample Contract using groovy 3](#_Toc30496482)

[Provide name for the contract 4](#_Toc30496483)

[Passing Values from File: 4](#_Toc30496484)

[Use Case 2: Changes to the existing API 4](#_Toc30496485)

[Use Case 3: Provider and Consumer API is not ready 4](#_Toc30496486)

[Provider side 4](#_Toc30496487)

[Step1: Download Contract 4](#_Toc30496488)

[Step 2: Add dependency plugin 4](#_Toc30496489)

[Step 3: Create Base Class 5](#_Toc30496490)

[Step 4: Hook the API 5](#_Toc30496491)

[Step 5: Maven build 5](#_Toc30496492)

[Consumer / Client test against the Stub 6](#_Toc30496493)

[Step 1 Add Dependency jars 6](#_Toc30496494)

[Step 2: Auto Configure Stub Runner 7](#_Toc30496495)

[Step 3: Make HTTP Call 7](#_Toc30496496)

[Step 4: Validate the response 7](#_Toc30496497)

[Stub not available / Provider API not ready 8](#_Toc30496498)

[Service virtualization using wiremock 8](#_Toc30496499)

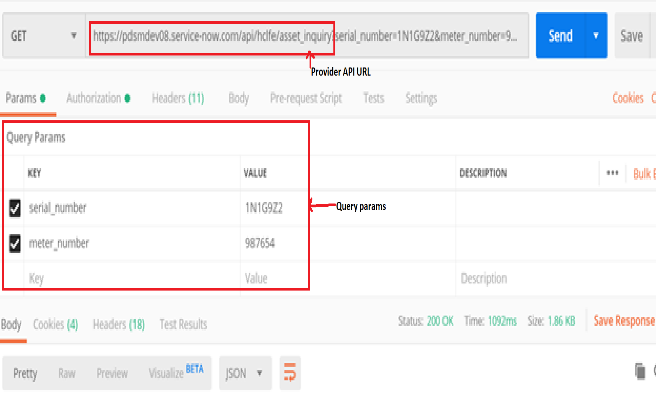
Spring Cloud Contract

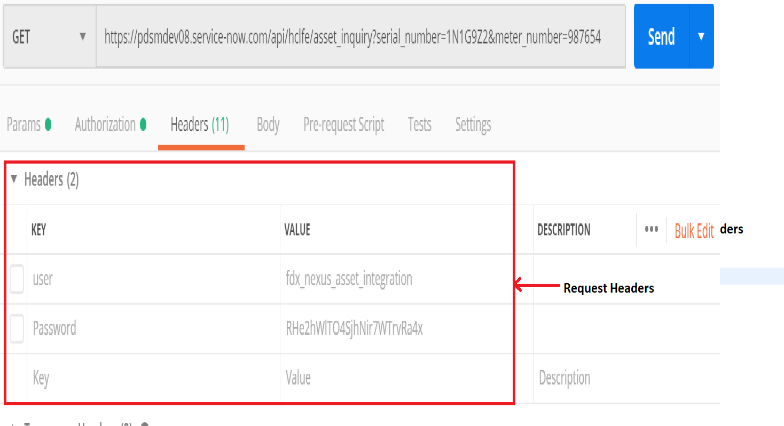
# Consumer Side – Generate Contract

## Use Case 1: Provider API is ready

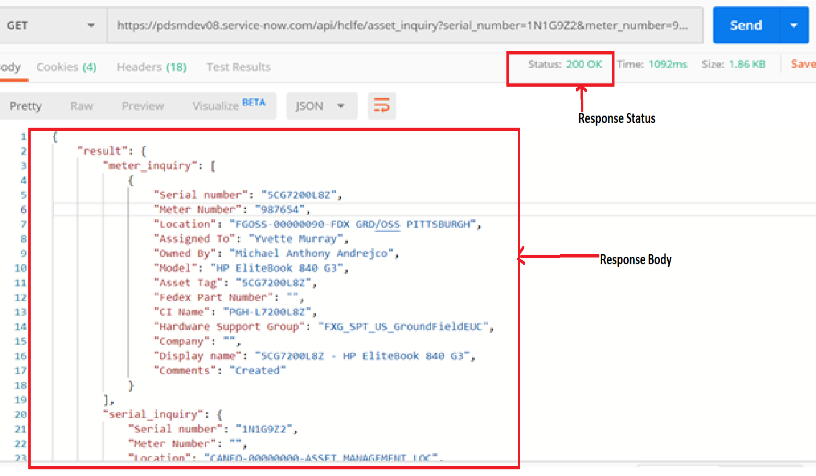
Test PDSM API - Get Request with ‘serial\_number’ and ‘meter\_number’ as query parameters and using Basic Auth for authorization for the below URL

https://pdsmdev08.service-now.com/api/hclfe/asset\_inquiry





The response of the PDSM API is 200 OK



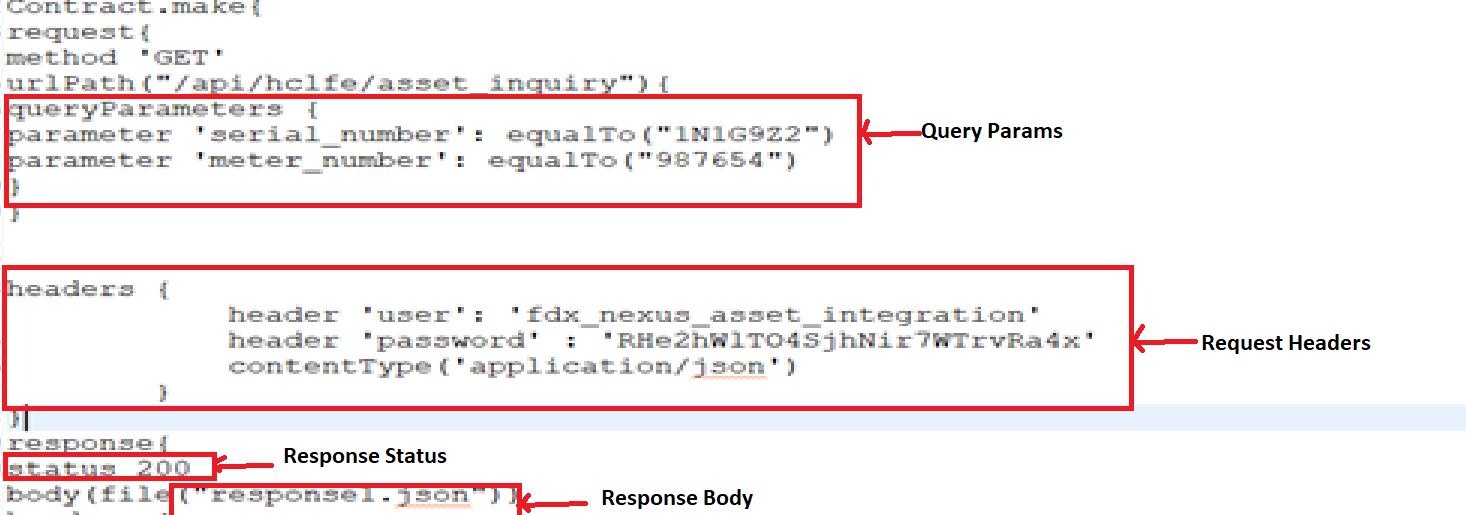
### Generate Contract for PDSM API from consumer side

The contracts are written with either **Groovy** or **YAML** and are placed in src/test/resources/contracts.

**Using Groovy**

The contract of the REST service can be defined as a .groovy script. This contract specifies that if there is a GET request to url /api/hclfe/asset\_inquiry with query params and auth token, it returns the response as application/json with Http 200 status and response body defined in response1.json

### Sample Contract using groovy



### Provide name for the contract

This name will be used by the provider to validate the contract against the API

### Passing Values from File:

Request and Response can be passed from external files in groovy

Once the Consumer has the contract defined. Its published to the repository. The provider verifies the published contract from consumer and creates a validated stub out of it

## Use Case 2: Changes to the existing API

Case 1: Create a new contract and publish to repository

Case 2: Change the current groovy contract with revised version

## Use Case 3: Provider and Consumer API is not ready

Consumer starts developing the contract first even before the API is developed which acts as an agreement between provider and consumer.

# Provider side

### Step1: Download Contract

Download the contract from the repository and place it in src/test/resources/contracts.



### Step 2: Add dependency plugin

Add the below plugins and dependency for verifying the contract

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-contract-verifier</artifactId>

<version>2.2.1.RELEASE</version>

</dependency>

<

<plugin>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-contract-maven- plugin</artifactId>

<version>2.1.1.RELEASE</version>

<extensions>true</extensions>

<configuration>

<baseClassForTests>com.fedex.demo.BaseClass</baseClassForTests>

</configuration>

</plugin>

### Step 3: Create Base Class

Create a base class for test using RestAssuredMockMVC

Add the below dependency in pom.xml for RestassuredMockMVC

<dependency>

<groupId>io.rest-assured</groupId>

<artifactId>spring-mock-mvc</artifactId>

<version>4.1.2</version>

<scope>test</scope>

</dependency>

### Step 4: Hook the API

Pass the controller to the MockMVC as below in the baseclass

@RunWith(SpringRunner.**class**)

@SpringBootTest(classes=API3AppApplication.**class**)

**public** **class** BaseClass {

//Inject your controller

@Autowired

API3Controller controller;

@Before

**public** **void** setup(){

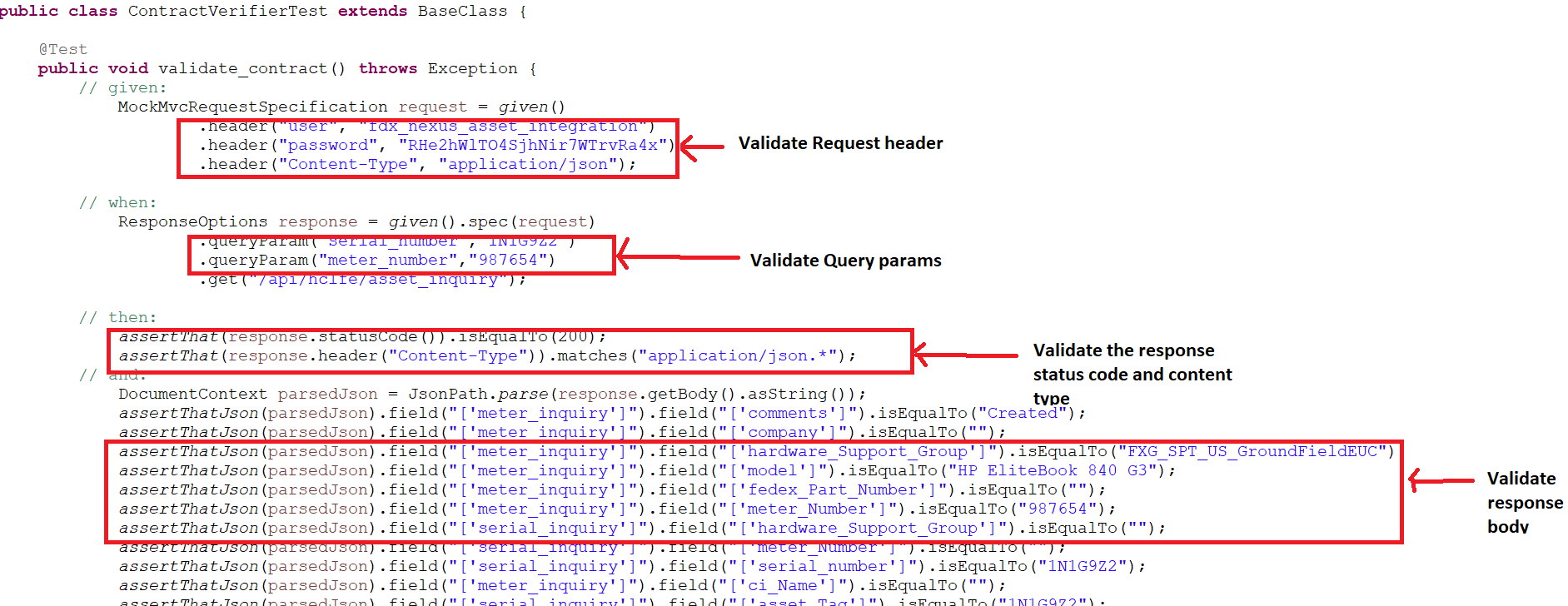
RestAssuredMockMvc.*standaloneSetup*(controller);

}

}

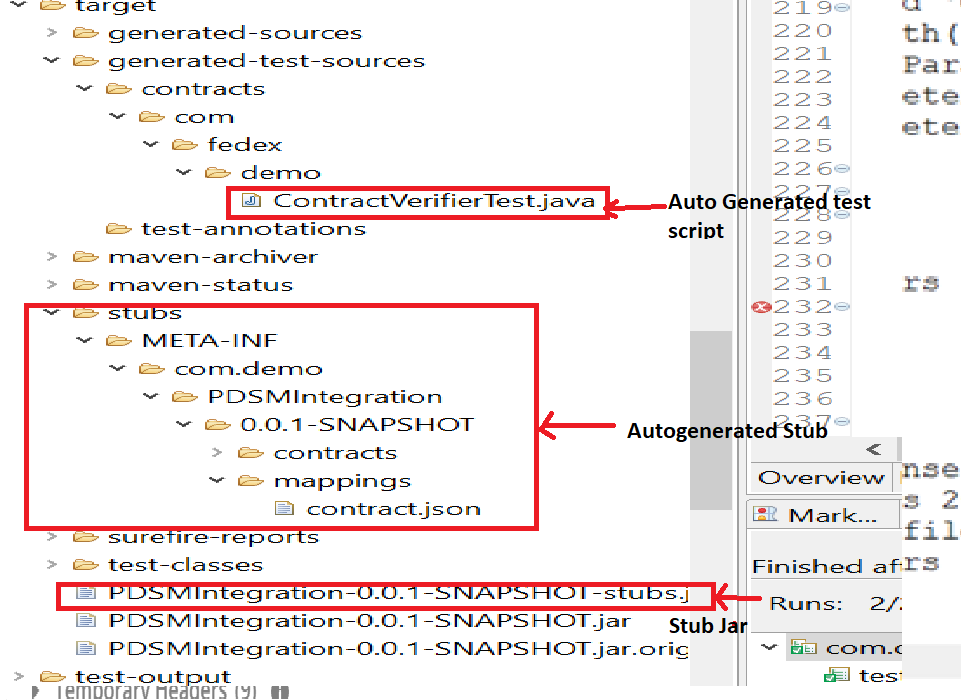
### Step 5: Maven build

Once the base class is ready, run maven clean install and the following are auto generated through the spring plugin



ContractVerifierTest – Automated test scripts are generated for the contract given by the consumer. It validates all the request and response parameters specified in the contract file. This contract file extends the base class as specified in the maven plugin. Screen shot below

Auto generated stub jar -The stubs are generated under target /stubs. These stub jars are checked or published into the repository where consumers can download and run tests against the stubs



# Consumer / Client test against the Stub

### Step 1 Add Dependency jars

Add the below dependency for stub runner

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-contract-wiremock</artifactId>

<version>2.2.1.RELEASE</version>

<scope>test</scope>

</dependency>

<dependency>

<groupId>org.springframework.cloud</groupId>

<artifactId>spring-cloud-starter-contract-stub-runner</artifactId>

<version>2.2.1.RELEASE</version>

<scope>test</scope>

</dependency>

### Step 2: Auto Configure Stub Runner

Using Spring annotation, stubs can be downloaded against the group id and artifact id and configured for a local stub server. For the below example, stubs are downloaded against the groupid : com.demo , artifactid: PDSMIntegration and port 8090

@AutoConfigureStubRunner(ids="com.demo:PDSMIntegration:+:stubs:8090",

stubsMode = StubRunnerProperties.StubsMode.LOCAL)

### Step 3: Make HTTP Call

Use RestTemplate client to make the HTTP request to get the request and response from the generated stubs.

### Step 4: Validate the response

Actual data from the consumer side is validated against the stub data



# Stub not available / Provider API not ready

# Service virtualization using wiremock

AutoConfigure Wiremock:

The following code will configure a response with a status of 201 to be returned when the relative URL exactly matches /api/hclfe/asset\_inquiry (including query parameters). The body of the response is an external file named response.json.

