

Extensibility API Toolkit Guide
Oracle Banking Digital Experience
Patchset Release 21.1.1.0.0

Part No. F40800-01

June 2021

ORACLE®

Extensibility API Toolkit Guide

June 2021

Oracle Financial Services Software Limited

Oracle Park

Off Western Express Highway

Goregaon (East)

Mumbai, Maharashtra 400 063

India

Worldwide Inquiries:

Phone: +91 22 6718 3000

Fax:+91 22 6718 3001

www.oracle.com/financialservices/

Copyright © 2006, 2021, Oracle and/or its affiliates. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are “commercial computer software” pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate failsafe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

This software or hardware and documentation may provide access to or information on content, products and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

Table of Contents

| | |
|--|-------------|
| 1. Preface | 1-1 |
| 1.1 Intended Audience | 1-1 |
| 1.2 Documentation Accessibility | 1-1 |
| 1.3 Access to Oracle Support | 1-1 |
| 1.4 Structure | 1-1 |
| 1.5 Related Information Sources | 1-1 |
| 2. Introduction | 2-1 |
| 3. Technology Stack | 3-1 |
| 4. Pre-requisites | 4-1 |
| 5. Write your First service | 5-1 |
| 5.1 Obtain Toolkit | 5-1 |
| 5.2 Set Environment Variables | 5-1 |
| 5.3 API Generation using UI Toolkit | 5-4 |
| 5.4 Write JSON | 5-9 |
| 5.5 Generate JSON through Swagger file | 5-13 |
| 5.6 Execute Gradle tasks | 5-14 |
| 6. JSON Explained | 6-19 |
| 7. FAQs | 7-1 |

1. Preface

1.1 Intended Audience

This document is intended for the following audience:

- Customers
- Partners

1.2 Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

1.3 Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit

<http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

1.4 Structure

This manual is organized into the following categories:

Preface gives information on the intended audience. It also describes the overall structure of the User Manual.

The subsequent chapters describes following details:

- Introduction
- Preferences & Database
- Configuration / Installation.

1.5 Related Information Sources

For more information on Oracle Banking Digital Experience Patchset Release 21.1.1.0.0, refer to the following documents:

- Oracle Banking Digital Experience Installation Manuals

2. Introduction

API Toolkit is a tool to generate new services as per the product framework. Users having host APIs for core banking can now generate their respective channel APIs.

It takes a JSON file as an input that includes various fields, methods and other details pertaining to the service to be created. The JSON carries set of keys whose values has to be provided by the user. On the basis of the JSON input provided, the tool :

- generates source code.
- compiles and packages it into relevant jar and war files.
- generates various database scripts required for the functioning of the services.

[Home](#)

3. Technology Stack

| Software | Version |
|----------|---------------------------|
| Java | Java JDK or JRE version 8 |
| Gradle | gradle-4.7 |
| OBDX | 20.1 |

[Home](#)

4. Pre-requisites

- Java JDK or JRE version 7 or higher must be installed. For installation of the Java please refer [installation guide](#).
- User must have gradle-4.7 installed
- Provide the following dependency “**javax.ws.rs-api-2.0.jar**” in build.gradle of endpoint and wherever required

[Home](#)

5. Write your First service

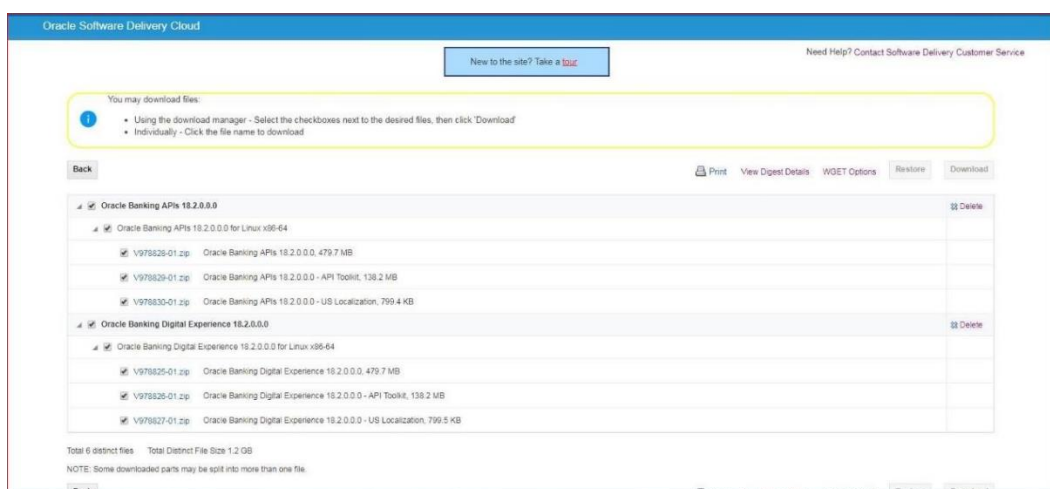
Let us start with a Hello World Service. This section will help you in writing an end to end Hello World service. The following needs to be done:

- You will need the API Toolkit.
- Set the home of the toolkit by adding an environment variable for it.
- Once the above two steps are done and all the prerequisites are fulfilled, you will be able to proceed further with the execution of simple gradle tasks.
- Execution of the gradle tasks as mentioned below will get you the deployables (WARs) and the database scripts (The order in which the tasks are executed must be maintained as provided in the explanation)
- Deploy the WAR's and run the database scripts.
- You can test your generated services.

The detailed steps for the first Hello World service is mentioned below:

5.1 Obtain Toolkit

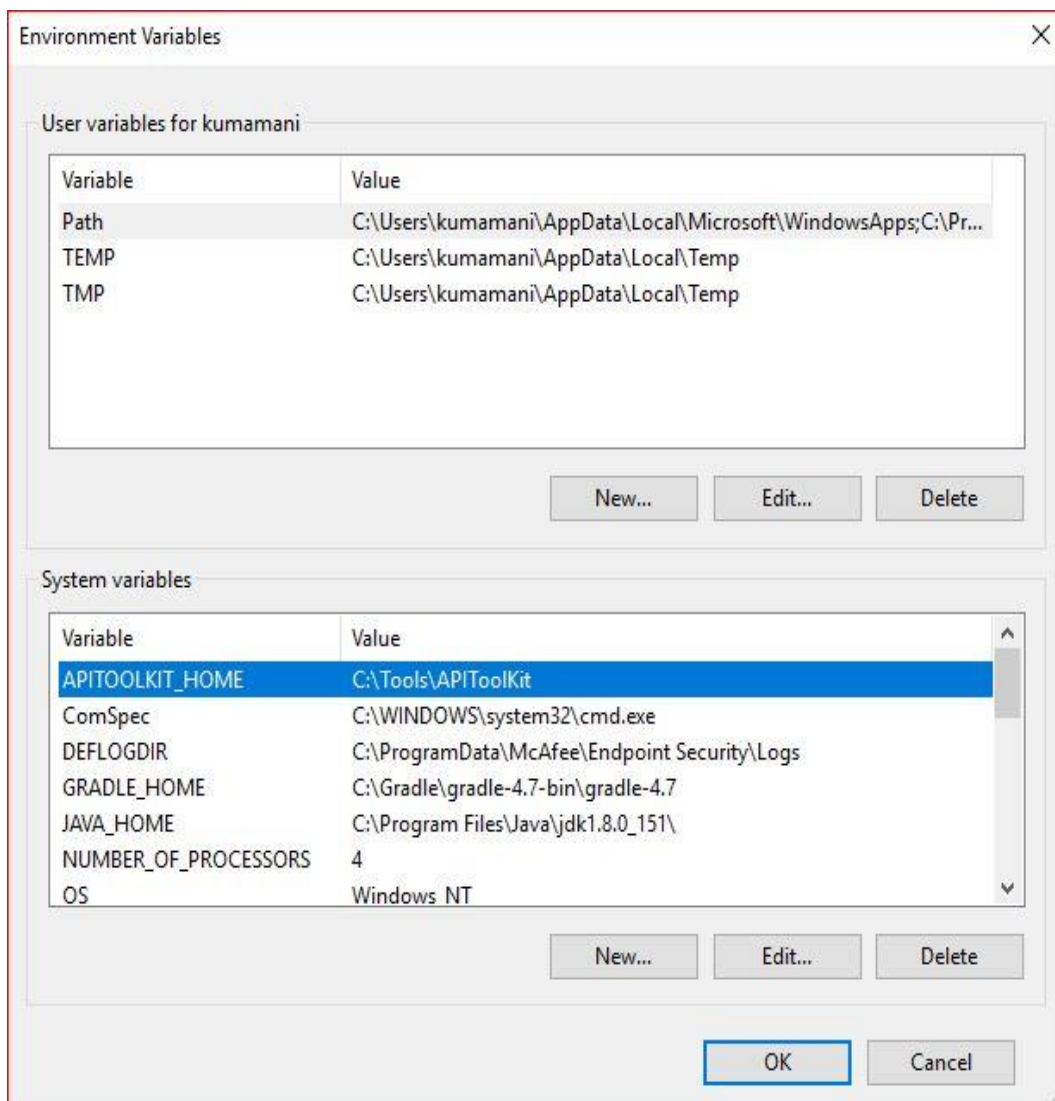
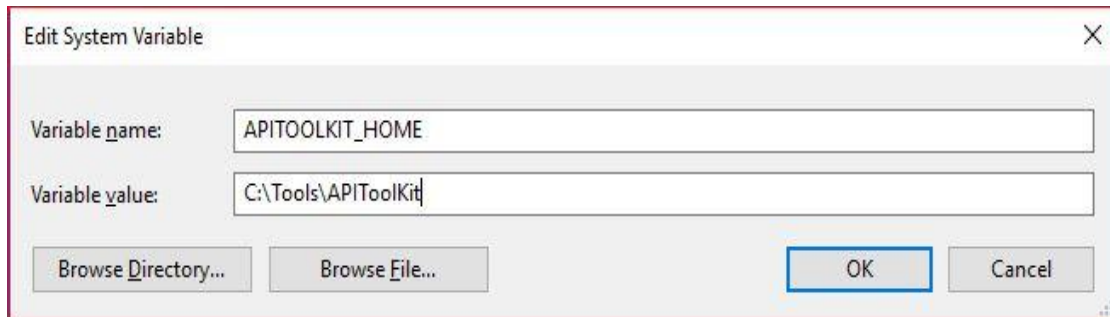
User should download the API Toolkit zip from the Oracle Software Delivery Cloud portal



5.2 Set Environment Variables

5.2.1 Windows

Right click on This PC and click → Properties → Advanced System Settings → Environment Variables. Under System Variables select Path, then click Edit.



5.2.2 Linux

User should run the following command on the terminal:

```
export APIToolKIT_HOME=<Location of APIToolKit folder>
```

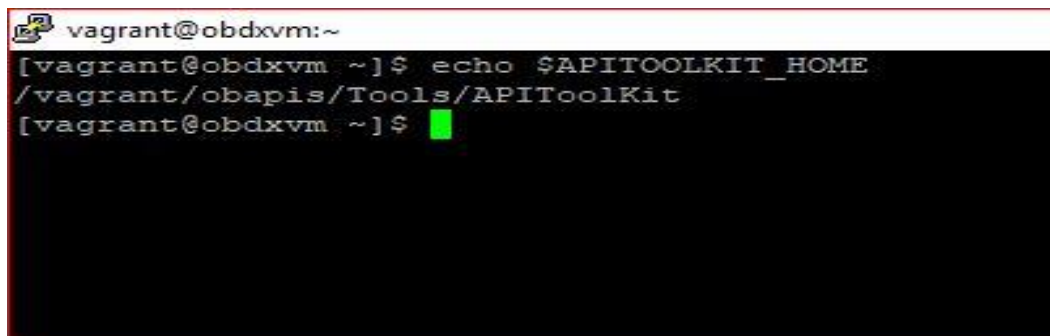
e.g.

```
export APIToolKIT_HOME=/vagrant/obdx/Tools/APIToolKit
```

The above mentioned step is depicted below:



```
vagrant@obdxvm:~  
[vagrant@obdxvm ~]$ export APIToolKIT_HOME=/vagrant/obapis/Tools/APIToolKit  
[vagrant@obdxvm ~]$
```



```
vagrant@obdxvm:~  
[vagrant@obdxvm ~]$ echo $APIToolKIT_HOME  
/vagrant/obapis/Tools/APIToolKit  
[vagrant@obdxvm ~]$
```

5.2.3 IOS

User should run the following command on the terminal:

```
export APIToolKIT_HOME=<Location of APIToolKit folder>
```

e.g.

```
export APIToolKIT_HOME=/vagrant/obdx/Tools/APIToolKit
```

The above mentioned step is depicted below:


```

dhcp-in-wifi-clear-10-120-54-3:/ obdxuser$ export APIToolKit_HOME=/vagrant/obdx/Tools/APIToolKit
dhcp-in-wifi-clear-10-120-54-3:/ obdxuser$ printenv
TERM_PROGRAM=Apple_Terminal
TERM=xterm-256color
SHELL=/bin/bash
TMPDIR=/var/folders/53/110hlz792z9gkqctfnm7rctr0000gn/T/
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.Ujht44psUw/Render
TERM_PROGRAM_VERSION=400
OLDPWD=/Users/obdxuser/Documents/uiworkbench
TERM_SESSION_ID=08048E8-7280-4E25-BF44-0022000FDE58
APIToolKit_HOME=/vagrant/obdx/Tools/APIToolKit
USER=obdxuser
SSH_AUTH_SOCK=/private/tmp/com.apple.launchd.8nKEshq7SK/Listeners
PATH=/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
PWD=/
XPC_FLAGS=0x0
XPC_SERVICE_NAME=0
HOME=/Users/obdxuser
SHLVL=1
LOGNAME=obdxuser
LC_CTYPE=UTF-8
_=/usr/bin/printenv
dhcp-in-wifi-clear-10-120-54-3:/ obdxuser$

```

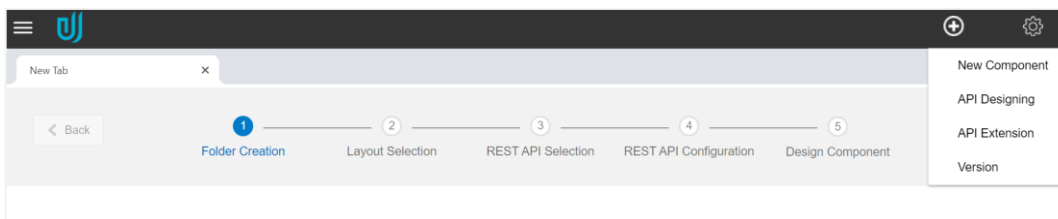
5.3 API Generation using UI Toolkit

You need to setup the UI Toolkit first. Refer to UI Toolkit setup documentation.

API Toolkit is integrated with this tool one simple has to perform all the pre requisites mentioned in the API toolkit document and then click on the plus icon  on the header and select API designing to use the API Toolkit

Once UI Toolkit is up and running, Users needs to complete this following steps.

Open API Designing in UI Toolkit.



Step 1: Define your API Toolkit, Provide the API name and the module name

The screenshot shows the 'API Definition' step in the UI Toolkit interface. The progress indicator shows four steps: 1. API Definition (active), 2. Methods Declarations, 3. Declare Type Details, and 4. Download. The main content area is titled 'API Definitions' and contains the following form fields:

- API Name:
- Module Name:
- Sub Module Name:

At the bottom of the form, there is a blue 'Next' button.

Step 2:

Declare your methods, give a name for your method without changing the initial letter i.e create, read, list etc.

Click on Add and provide the attributes and entitlements for your method.

Methods Declarations

URI: /bitcoin

Module Code: BI

Methods: Create

Method Name: create

| Method Name | Operation Type | Task Name | Account Type | Task Type | Task Aspects |
|-------------|----------------|-----------------|--------------|-----------------------|--------------|
| create | CREATE | Create Payments | | FINANCIAL_TRANSACTION | |

Back

Method Details

Task

Task Name: Create Payment

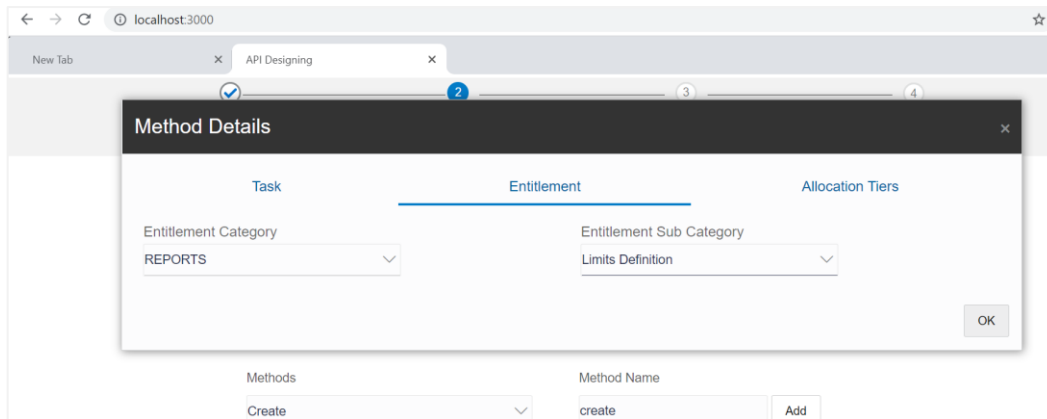
Aspects:

Module Type: TERM DEPOSIT

Entitlement

Account Types:

Task Type: Financial Transaction

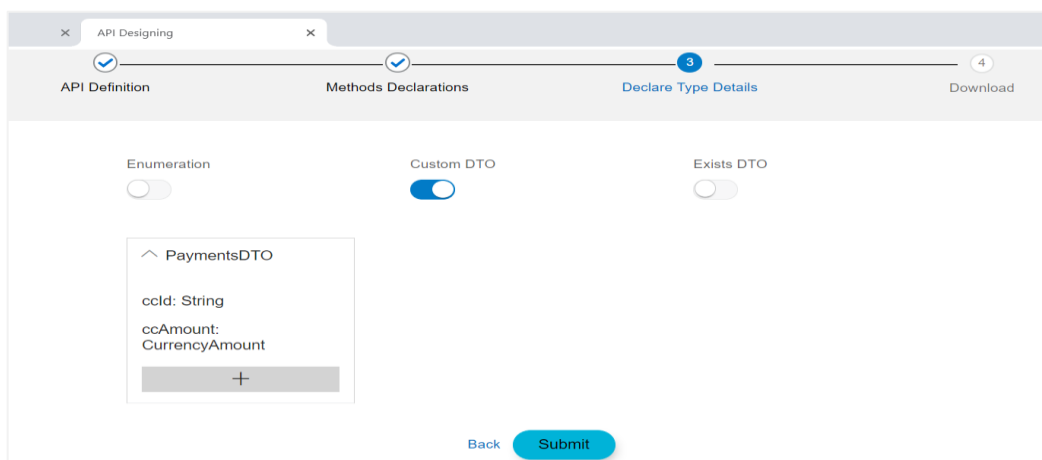
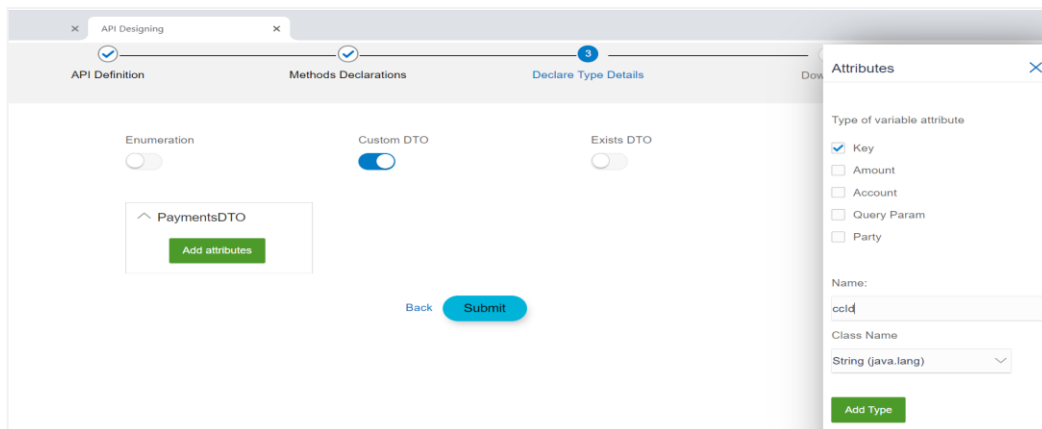


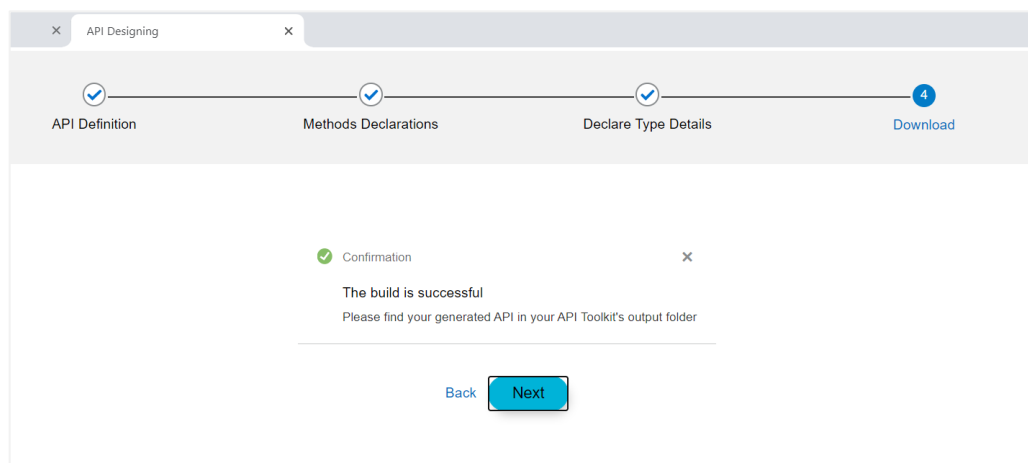
Step 3:

Define your DTO and its attribute in the third step.

Click on Add Attributes select the type of attribute (**One key attribute is must**) give the attribute name and its datatype.

After successful completion Click on submit to generate the code.





After successful build you project will get generated in **APIToolKIT_HOME > output > com.ofss.digx.cz.module.{moduleName}**

System (C:) > APIToolKit > output

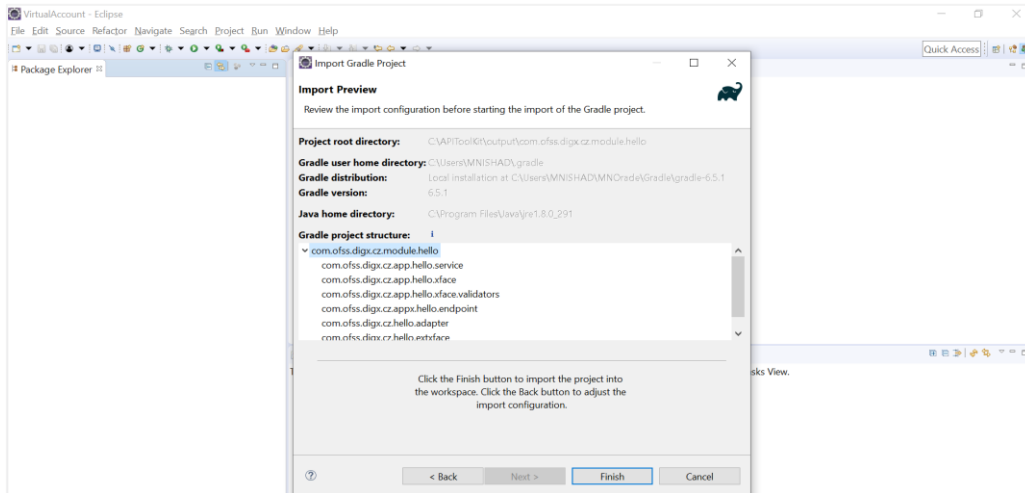
| Name | Date modified | Type |
|-------------------------------|-------------------|-------------|
| .gradle | 6/11/2021 3:36 PM | File folder |
| adapter-impl | 6/11/2021 3:35 PM | File folder |
| buildSrc | 6/11/2021 3:36 PM | File folder |
| com.ofss.digx.cz.module.hello | 6/11/2021 3:36 PM | File folder |
| deployables | 6/9/2021 5:26 PM | File folder |

Note: In order to generate a module for obdx kernel mode the user needs to add an xml in the APIToolKIT_HOME which can be found in the svn repository. The module is generated according to the latest modularized framework. It won't be able to generate files for old OBDX frameworks.

It will create a skeleton adapter-impl for your module in output > adapter-impl folder in which you can implement the adapter's which you want to use.

It will also generate an ext-xface implementation for your module in output > ext-xface > ext-xface-impl folder

The generated module is a gradle project hence you can import it into any IDE.



5.3.1 Gradle Repository

API Toolkit now imports the jars from gradle/maven repositories.

Hence the user outside OBDX framework needs to maintain a gradle/maven repositories in order to import the OBDX framework jars.

User simply has to provide the repository URL in **APIToolkit_HOME > output > artifactory.properties with “,” separated values**

| | | | |
|------------------------|------------------|-----------------|------|
| artifactory.properties | 6/9/2021 5:26 PM | PROPERTIES File | 1 KB |
| build.gradle | 6/9/2021 5:26 PM | GRADLE File | 2 KB |
| settings.gradle | 6/9/2021 5:26 PM | GRADLE File | 1 KB |

`urls=https://artifacthub-phx.oci.oraclecorp.com/obdx-maven-dev-local/,http://whf00mja.in.oracle.com:8082/artifactory/obdx-gradle-dev-local/`

5.4 Write JSON

This is the most important part of toolkit. The toolkit takes this JSON as input in the form of JSON file. It is where you need to write the details pertaining to the Service to be generated.

The details for the “Hello World” service to be put in the JSON is given below :

```

1  {
2    "domainName":"HelloWorld",           - 1
3    "moduleName":"hello",                - 2
4    "subModuleName":"world",              - 3
5    "moduleCode":"HW",                    - 4
6    "uri":"/hello",                       - 5
7    "methods":[ ],                        - 6
8    "typeDetailsDTOs":[ ],                - 7
9    "type":null                            - 8
10 }
```

For details of each and every field of the input JSON please refer section 6(JSON Explained). It is known that every functionality in OBDX runs on underlying domain.

This JSON for the Hello World has a domain name “HelloWorld” which belongs to a submodule “world” of the “hello” module. You’ll be able to reach “HelloWorld” resource at path “/hello”.

HelloWorld functionality allows various operations such as ‘create’, ‘read’ which is provided in the JSON as an array of methods and it consists of various fields which are to be provided in typeDetailsDTO’s. The type(8) represents the type of domain name. If it is not provided by the user, then it is deduced from domain name, module name and sub module name. The typeDetailsDTO’s (7) and methods (6) details put in the HelloWorld input JSON is given below:


```

"typeDetailsDTOs": [{
  "fields": [
    {
      "type": {
        "packageName": "java.lang",
        "className": "String"
      },
      "name": "id",
      "varAttrs": {
        "key": true
      }
    },
    {
      "type": {
        "packageName": "java.util",
        "className": "List"
      },
      "name": "greetings",
      "genericType": [
        {
          "packageName": "java.lang",
          "className": "String"
        }
      ]
    },
    {
      "type": {
        "className": "BirthdayWishes"
      },
      "name": "birthdayWishes"
    }
  ]
},
]

```

The above snippet is of the typeDetailsDTO's in which fields or variables to be declared in the HelloWorld. It is an array in which each element consists of fields, type, typeAttrDTO and enumeration. User has to define one of these fields as key of the HelloWorld domain. Here the key is 'id' variable (as can be seen its attributes 'key' carries 'true' value). Another field is a list of strings followed by 'birthdayWishes' of type BirthdayWishes. For details of preparing fields array please the JSON Explained section(Section 6).

```

{
  "fields": [
    {
      "type": {
        "packageName": "com.ofss.fc.datatype",
        "className": "Date"
      },
      "name": "birthDate"
    },
    {
      "type": {
        "packageName": "java.lang",
        "className": "String"
      },
      "name": "birthPlace"
    }
  ],
  "type": {
    "className": "BirthdayWishes"
  },
  "typeAttrDTO": {
    "custom": true,
    "exists": false
  },
  "enumeration": null
},
"type": null,

```

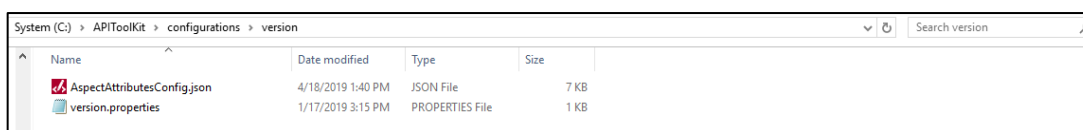
The above snippet is of another element of typeDetailsDTO array. It represents the type BirthdayWishes comprising of the fields 'birthDate' and 'birthPlace'. The typeDetailsDTO has certain attributes like typeAttrDTO and enumeration

```

"methods": [
  {
    "name": "create",
    "operationType": "CREATE",
    "methodAttributes": {
      "task": {
        "taskType": "MAINTENANCE",
        "taskAspects": [
          "approval",
          "audit",
          "2fa",
          "limit"
        ],
        "moduleType": "COMMON",
        "txnType": "PAYMENTS"
      },
      "entitlements": {
        "entitlementCategory": "EC",
        "entitlementSubCategory": "ENNT"
      }
    }
  },
  {
    "name": "read",
    "operationType": "READ",
    "methodAttributes": {
      "task": {
        "taskType": "MAINTENANCE",
        "taskAspects": [
          "approval",
          "audit",
          "2fa",
          "limit"
        ],
        "moduleType": "COMMON",
        "txnType": "PAYMENTS"
      },
      "entitlements": {
        "entitlementCategory": "EC",
        "entitlementSubCategory": "ENNT"
      }
    }
  }
]

```

The above snippet is of the 'methods' in the HelloWorld, which again is an array, each element representing one method. In HelloWorld we have two methods create and read hence two elements. The methods has certain attributes like taskType, moduleType and txnType. If the 'entitlements' is provided empty value the tool will generate default entitlements for it. For details regarding the possible values in there attributes please refer [section : JSON Explained](#).



| Name | Date modified | Type | Size |
|-----------------------------|-------------------|-----------------|------|
| AspectAttributesConfig.json | 4/18/2019 1:40 PM | JSON File | 7 KB |
| version.properties | 1/17/2019 3:15 PM | PROPERTIES File | 1 KB |

The above snippet is of the version folder which contains AspectAttributesConfig.json and version.properties. version.properties contains the OBDX version(for which the artifacts are to be generated). Based on the version, the corresponding aspectAttributes are selected from the AspectAttributesConfig.json. This JSON consists of different transaction types with their corresponding attributes. With the help of this JSON and txn type mentioned in the methods of input JSON, the domainName#AspectAttributes.json is generated which is shown in the below Figure. Now ,the user has to fill the corresponding attributes in domainName#AspectAttributes.json from input JSON. For that purpose user has to check the fields (declared in the domain) corresponding to each attributes. Aspect attributes are mandatory.

```

1 {
2   "aspectAttributesMap":{
3     "accountType#AccountType": "",
4     "accountId": "",
5     "accountPartyId": "",
6     "partyName": "",
7     "amount:approvalAttr": "",
8     "payee.id": "",
9     "payee.name": "",
10    "valueDate": "",
11    "currencyAmount:limitAttr": "",
12    "partyId": ""
13  }
14 }

```

If any operation added has a task Aspect “approval” or “limit” then user has to fill the corresponding data in this file else not required.

accountType#AccountType : Value of Enumeration of AccountType (e.g #CSA)

accountId : attribute used to define the accountId

accountPartyId : attribute used to define the account Party Id

partyName.fullName : Name of the Party from attributes

amount:approvalAttr: attribute of varAttr type amount which will be used as approval amount

5.5 Generate JSON through Swagger file

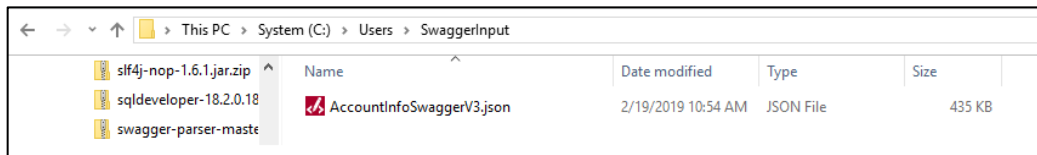
If the user has the required Swagger file, then its location must be provided.

```

1 {
2   "openapi": "3.0.0",
3   "info": {
4     "title": "Account and Transaction API Specification",
5     "description": "Swagger for Account and Transaction API Specification",
6     "termsOfService": "https://www.openbanking.org.uk/terms",
7     "contact": {
8       "name": "Service Desk",
9       "email": "ServiceDesk@openbanking.org.uk"
10    },
11    "license": {
12      "name": "open-licence",
13      "url": "https://www.openbanking.org.uk/open-licence"
14    },
15    "version": "v3.1-RCL"
16  },
17  "paths": {
18    "/account-access-consent": {
19      "post": {
20        "tags": [
21          "Account Access"
22        ],
23        "summary": "Create Account Access Consents",
24        "operationId": "CreateAccountAccessConsents",
25        "parameters": [
26          {
27            "name": "x-fapi-financial-id-Param",
28            "in": "header",
29            "required": true,
30            "schema": {
31              "type": "string"
32            }
33          },
34          {
35            "name": "x-fapi-customer-last-logged-time-Param",
36            "in": "header",
37            "required": true,
38            "schema": {
39              "type": "string"
40            }
41          },
42          {
43            "name": "x-fapi-customer-ip-address-Param",
44            "in": "header",
45            "required": true,
46            "schema": {
47              "type": "string"
48            }
49          },
50          {
51            "name": "x-fapi-interaction-id-Param",
52            "in": "header",
53            "required": true,
54            "schema": {
55              "type": "string"
56            }
57          }
58        ]
59      }
60    }
61  }
62 }

```

Here we use " AccountInfoSwaggerV3.json" as a sample shown above.



The above snippet depicts the Swagger file location. Now user will be shown a list of URI's that is available in the file. The snippet of the list is shown below.

```

1. /account-access-consents
2. /account-access-consents/{ConsentId}
3. /accounts
4. /accounts/{AccountId}
5. /accounts/{AccountId}/balances
6. /accounts/{AccountId}/beneficiaries
7. /accounts/{AccountId}/direct-debits
8. /accounts/{AccountId}/offers
9. /accounts/{AccountId}/party
10. /accounts/{AccountId}/product
11. /accounts/{AccountId}/scheduled-payments
12. /accounts/{AccountId}/standing-orders
13. /accounts/{AccountId}/statements
14. /accounts/{AccountId}/statements/{StatementId}
15. /accounts/{AccountId}/statements/{StatementId}/file
16. /accounts/{AccountId}/statements/{StatementId}/transactions
17. /accounts/{AccountId}/transactions

```

He will be allowed to choose the URI's to be onboarded. Each URI will correspond to a particular domain. After selecting the URIs, respective JSON files will get created at the file location:

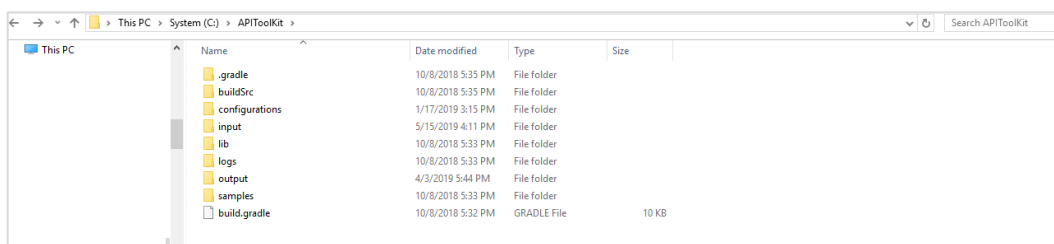
APITOOLKIT_HOME/input/json/<DomainName>.json, which can be used further to generate the respective artifacts generation.

5.6 Execute Gradle tasks

Now that you are ready with your input JSON and the pre-requisites are done you are eligible to run required gradle tasks. There is a list of gradle tasks which needs to be run in the provided order. These tasks will generate source code, furnish eclipse projects for the generated code, build the source and provide the deployable WARs in the <APITOOLKIT_HOME>/output/deployables folder.

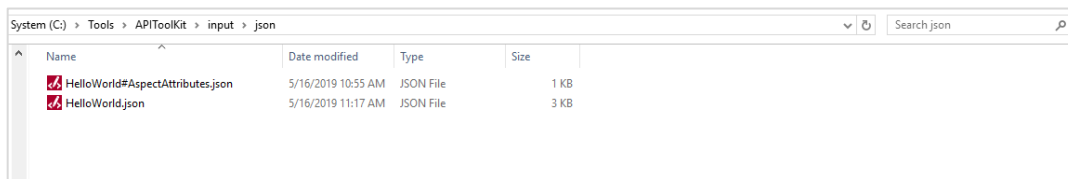
Before moving forward have a look at the folder structure of the toolkit.

The <APITOOLKIT_HOME> has input, output, logs folders that are going to be required in further steps. The <APITOOLKIT_HOME> is depicted below in the snippet.



- **Input folder**

It is where the input JSON has to be provided. The HelloWorld.json (input JSON for HelloWorld) can be seen below:

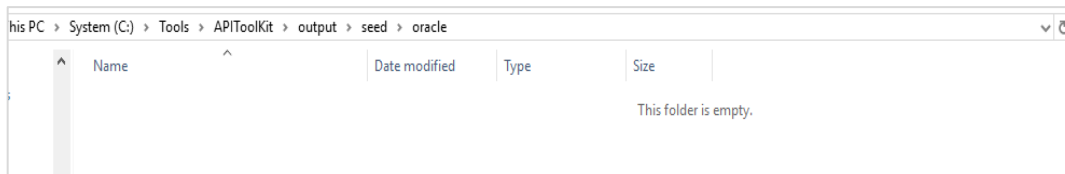


- **output folder**

Output folder has deployables, scripts, swagger, etc.

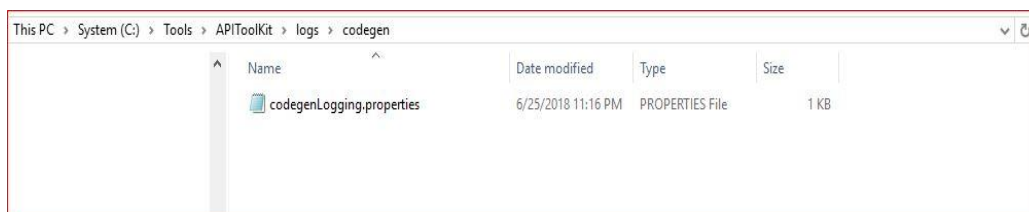
- **seed/oracle**

It is where the required SQL scripts will be generated.



- **logs folder**

It is where the logs of the toolkit will get generated.



- **gradle swaginput**

This command fetches the Swagger file from the specified location and generates the corresponding input JSON files.

- **gradle precodegen**

It generates the `domainName#AspectAttributes.json` as per the transaction type and `AspectAttributes.json` in the `<APIToolKit HOME>/input/json` location

- **gradle codegen(*)**

It generates the source code as per the input provided as JSON. So before you execute this task make sure you have the prepared input JSON placed at <APIToolKIT_HOME>/input/json

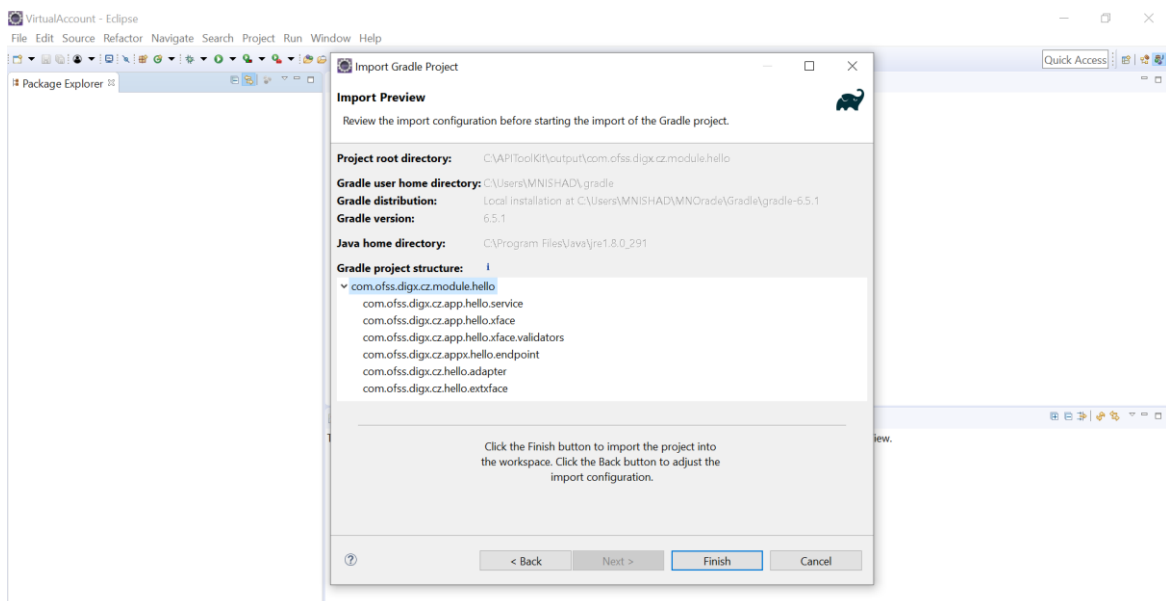
```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Tools\APIToolKit>gradle codegen
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 1m 0s
2 actionable tasks: 2 executed
C:\Tools\APIToolKit>
```

- **gradle build(on generated module)**

You can directly import your gradle project into an IDE or else you can execute gradle build command on command line



```

C:\APIToolKit\output\com.ofss.digx.cz.module.hello>gradle build

> Configure project :
swagger.resources file generated successfully!!!

> Configure project :com.ofss.digx.cz.appx.hello.endpoint
base.resources file generated successfully!!!

> Task :com.ofss.digx.cz.app.hello.service:compileJava
Note: C:\APIToolKit\output\com.ofss.digx.cz.module.hello\com.ofss.digx.cz.app.hello.service\src\main
.java uses unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

> Task :com.ofss.digx.cz.app.hello.xface.validators:generateValidatorSources
generated validator sources invoked

Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.5.1/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 8s
15 actionable tasks: 15 executed

```

- **gradle thirdpartygen**

This task will get you the third party deployables. It basically generates the simulation MDB WARs which mocks the third-party host.



```

C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

C:\Tools\APIToolKit>gradle codegen
Starting a Gradle Daemon (subsequent builds will be faster)

BUILD SUCCESSFUL in 1m 0s
2 actionable tasks: 2 executed
C:\Tools\APIToolKit>gradle eclipse

BUILD SUCCESSFUL in 12s
2 actionable tasks: 2 executed
C:\Tools\APIToolKit>gradle build

> Task :output\obdx.cz.app.domain.module:com.ofss.digx.cz.module.hello:compileJava
Note: C:\Tools\APIToolKit\output\obdx.cz.app.domain.module\com.ofss.digx.cz.module.hello\src\com\ofss\digx\cz\app\hello\service\world\ext\HelloWorldExtExecutor.java use
s unchecked or unsafe operations.
Note: Recompile with -Xlint:unchecked for details.

BUILD SUCCESSFUL in 13s
2 actionable tasks: 2 executed
C:\Tools\APIToolKit>gradle thirdpartygen

BUILD SUCCESSFUL in 9s
3 actionable tasks: 3 executed
C:\Tools\APIToolKit>

```

- **gradle swagger (on generated module)**

It generates a swagger document that comprises details of the generated REST API.

This document must be hosted and that URL must be provided to UI Workbench. Details regarding how to use this document is mentioned in the user guide of UI Workbench.

The swagger will get generated on the particular path **APIToolKit_HOME > output > com.ofss.digx.module.{moduleName} > com.ofss.digx.appx.{moduleName}.endpoint > build**


```

C:\APIToolKit\output\com.ofss.digx.cz.module.hello>gradle swagger

> Configure project :
swagger.resources file generated successfully!!!







> Configure project :com.ofss.digx.cz.appx.hello.endpoint
base.resources file generated successfully!!!

> Task :swagger
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
Jun 11, 2021 4:43:20 PM com.ofss.fc.infra.config.ConfigurationManager constructConfigProvid
SEVERE: Found incorrect preference setup for jdbcpreference
Jun 11, 2021 4:43:20 PM com.ofss.fc.infra.config.ConfigurationManager constructConfigProvid
SEVERE: Found incorrect preference setup for DocumentConfig
Jun 11, 2021 4:43:20 PM com.ofss.fc.infra.config.ConfigurationManager constructConfigProvid
SEVERE: Found incorrect preference setup for DayOneConfig
Open API Specification file generated at C:\APIToolKit\output\com.ofss.digx.cz.module.hello
Deprecated Gradle features were used in this build, making it incompatible with Gradle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.5.1/userguide/command_line_interface.html#sec:command_line_wa

BUILD SUCCESSFUL in 3s
9 actionable tasks: 2 executed, 7 up-to-date

```

Desktop > APIToolKit > output > com.ofss.digx.cz.module.hello > com.ofss.digx.cz.appx.hello.endpoint > build

| <input type="checkbox"/> Name | Date modified | Type | Size |
|--|-------------------|-------------|------|
|  classes | 6/10/2021 2:08 PM | File folder | |
|  generated | 6/10/2021 2:08 PM | File folder | |
|  libs | 6/10/2021 2:08 PM | File folder | |
|  resources | 6/10/2021 2:08 PM | File folder | |
|  tmp | 6/10/2021 2:08 PM | File folder | |
|  swagger-hello.json | 6/10/2021 5:29 PM | JSON File | |

Note: In case the user needs to change the input JSON as per changed requirement please execute the below task before performing the above-mentioned steps for the updated JSON:
gradle clean

6. JSON Explained

User should prepare the JSON with right set of values of the keys/nodes. A sample JSON is depicted below in the Figure3.1. The JSON is set with sample values for a mutual fund holdings functionality under payments. As part of the explanation, some of the JSON nodes or fields or keys may differ from the sample JSON provided (It has been done to explain functionalities which cannot be covered using just sample JSON) . Each and every key/node of the JSON is explained below:

```

{
  "domainName": "Holdings",           - 1
  "moduleName": "mutualfund",         - 2
  "subModuleName": "holdings",        - 3
  "moduleCode": "MF",                 - 4
  "uri": "/holdings",                 - 5
  "methods": [ ],                     - 6
  "typeDetailsDTOs": [ ],             - 7
  "type": null                         - 8
}

```

- **domainName(*)**: User has to input the name of the domain entity which represents one business use case. e.g. Holdings i.e. mutual fund holdings is a domain(a business use case under payment transactions)
- **moduleName(*)**: User should provide the name of the module to which this domain functionality belongs to.eg. Holdings domain belongs to “mutualfund” module.

```
"moduleName": "mutualfund",           — 2
```

- **subModuleName**: It is common to have module constituting of sub-modules. User should set the appropriate name of the sub-module (depending upon the domain functionality).

```
"subModuleName": "holdings",         — 3
```

- **moduleCode(*)**: It is basically an abbreviated unique identifier of the module to which the service belongs . It is formed by taking the initials of the module name. “moduleCode” has to be decided by the user. Here “MF” is formed by taking initials of the “mutualfund” module

e.g.

```
"moduleCode": "MF",                  — 4
```

Here “MF” is formed by taking initials of the “mutualfund” module.

- **uri(*)**: It is the path of the REST resource. It is the path exposed to the end user, of the product channel (as REST API). This is the URI which will appear in the swagger JSON created using the tool.

```
"uri": "/holdings",
```

This is the URI which will appear in the swagger JSON created using the tool.

- **methods(*)**: User should provide the methods which are expected in the domain. A combination of “name” and “operationType” nodes uniquely defines a method. The “methods” node is an array of methods defined in the domain. Part of the JSON depicting methods is presented below:

```
"methods": [
  {
    "name": "create",
    "operationType": "CREATE",
    "methodAttributes": {
  },
  {
    "name": "read",
    "operationType": "READ",
    "methodAttributes": {
  }
],
```

- **name(*)**: User should provide the name of the methods. A method name depicts the operation that the user wants to perform by calling this method. For example
- **operationType(*)** : Operation type is synonymous to REST operation type. So the method with operationType 'CREATE' will result in creation of a new resource at the specified URL (similar to what REST “POST” operation does).

User must provide the functionality the method is going to perform. It should be one of the following list of strings: 'CREATE', 'READ', 'UPDATE'.

If the method performs creation task it should have operation type as “CREATE”.

- **methodAttributes**: Method or Operation defined in the application carries various properties which defines its behavior. For example an operation or a method can go for approval. In order to ensure that the declared method has the property to go through approval process the task aspects of the task attribute must have a value “approval”. These properties which attaches various behaviors to the application, facilitates various application requirements such as swagger UI generation, etc. are taken as input by the Toolkit in the form of “methodAttributes”.

It has various attributes such as 'task', 'entitlements', 'swaggerAnnotations', 'allocation'

User should provide various attributes of the methods such as tasks associated with them, entitlements allocated to them and swagger details.

```

"methodAttributes":      - 6.1.3
{
  "task":                - 6.1.3.1
  {
    entitlements":       - 6.1.3.2
    {
      swaggerAnnotations": - 6.1.3.3
      {
        "allocationTiers": ["REST","SERVICE","DOMAIN","REPOSITORY"] - 6.1.3.4
      }
    }
  }
}

```

- **task:** User should provide the details of the task associated with the method

```

"task": {
  "taskName": "Create Forex Deal",           - 6.1.3.1.1
  "accountTypes": ["CSA"],                  - 6.1.3.1.2
  "taskType": "MAINTENANCE",               - 6.1.3.1.3
  "taskAspects": [                          - 6.1.3.1.4
    "approval",
    "audit",
    "2fa",
    "limit"
  ],
  "moduleType": "COMMON",                  - 6.1.3.1.5
  "txnType": "PAYMENTS"                    - 6.1.3.1.6
},

```

- **taskName:** User should declare name of the task associated with the current method
- **accountTypes:** User should provide the supported account types by the current transacting method. The set of all possible values that a user can provide are given below in the table with its description:

For detailed values to be input in the “accountTypes” please refer below table

| Sr. No. | AccountTypes | Description |
|---------|--------------|---|
| 1 | CSA | “CSA” refers current and savings accounts |
| 2 | LON | "LON" refers loan accounts |
| 3 | TRD | "TRD" refers term Deposits |
| 4 | CCA | "CCA" refers Credit card accounts |

- **taskType:** User should provide the type of the transaction that this method will be using. It can be a maintenance transaction or can be an auth-admin transaction, etc. The set of possible values that a user can provide is given below in the table

| Sr. No. | TaskType | Description |
|---------|--------------------------|--|
| 1 | FINANCIAL TRANSACTION | Transactions involving account and amount i.e. monetary transactions |
| 2 | NONFINANCIAL TRANSACTION | Transactions involving no monetary value. For e.g. Request a cheque book |
| 3 | INQUIRY | Inquiry of transactions |
| 4 | ADMINISTRATION | ADMINISTRATION |
| 5 | MAINTENANCE | MAINTENANCE |
| 6 | COMMON | COMMON |

- **taskAspects:** User should provide the details of the aspects of the transaction associated with the current method. An aspect of the transaction defines a feature inherently provided by the framework. For example “approval” is feature of the transaction which it is going to need while execution. Similarly, there are various features like limit, two-factor authentication(2fa), audit, etc. which the transaction can be dependent upon (based on use case of the transaction).The set of possible values that taskAspects can carry are :

| Sr. No. | TaskAspects | Description |
|---------|----------------|---|
| 1 | limit | Applicability of limits for a transaction |
| 2 | account-access | Applicability of access for a transaction |
| 3 | approval | Applicability of approval rules for a transaction |
| 4 | working-window | Applicability of working window i.e. cutoff for a transaction |
| 5 | blackout | Applicability of blackout or outage for a transaction |
| 6 | 2fa | Applicability of two factor authentication for a transaction |
| 7 | audit | Applicability of audit log for a transaction |
| 8 | grace-period | Applicability of grace period for a transaction |

| Sr. No. | TaskAspects | Description |
|---------|----------------------|---|
| 9 | eReceipt | Applicability of e-Receipt for a transaction |
| 10 | purpose-mapping | Applicability of purpose mapping to transaction |
| 11 | currency-config | Applicability of currency mapping to transaction |
| 12 | account-relationship | Applicability of access on the basis of account relationships |

- **moduleType**: User should provide the type of the module.

| Sr. No. | moduleType | Description |
|---------|------------|------------------|
| 1 | TD | TERM_DEPOSIT |
| 2 | CH | CASA |
| 3 | PI | PARTY |
| 4 | LN | LOAN |
| 5 | OR | ORIGINATION |
| 6 | PC | PAYMENTS |
| 7 | AT | CHANNELS |
| 8 | CC | CREDIT_CARD |
| 9 | SMS | SMS |
| 10 | FL | FINANCIAL_LIMITS |
| 11 | WA | WALLET |
| 12 | BO | BACK_OFFICE |
| 13 | FU | FILE_UPLOAD |
| 14 | AP | APPROVALS |
| 15 | NM | NOMINEE |
| 16 | AL | ALERTS |
| 17 | PFM | PFM |

| Sr. No. | moduleType | Description |
|---------|------------|-------------------------|
| 18 | RT | REPORTS |
| 19 | LC | LETTEROFCREDIT |
| 20 | BL | BILL |
| 21 | CM | COMMON |
| 22 | BM | BENEFICIARYMAINTAINANCE |
| 23 | GR | GENERICREST |
| 24 | MT | ADMIN_MAINTENANCE |
| 25 | FX | FOREXDEAL |
| 26 | MB | MOBILE |

- **txn type:** User should provide the type of the transaction
- **entitlements:**

Entitlements are meant to group transactions. Transactions can be grouped into categories and further into sub-categories.

For example a user onboarding transaction can be grouped into “*Admin Maintenance*” category and further into “*User Management*” sub-category. User should provide the details of category into which the currently executing method can be grouped with a further sub-category level grouping. e.g.

```

"entitlements": {
  "entitlementCategory": "FOREX",
  "entitlementSubCategory": "Forex_Deal_Booking"
},

```

There is a list of groups and sub groups provided in the table below.

| Sr. No. | Entitlement Category | Description |
|---------|----------------------|---|
| 1 | MT | Entitlement Category constant for 'ADMIN MAINTENANCE' |
| 2 | RP | Entitlement Category constant for 'REPORTS' |
| 3 | CS | Entitlement Category constant for 'CUSTOMER SERVICING'. |

| Sr. No. | Entitlement Category | Description |
|---------|----------------------|--|
| 4 | CASA | Entitlement Category constant for 'CURRENT SAVING ACCOUNT' |
| 5 | TD | Entitlement Category constant for 'TERM DEPOSIT' |
| 6 | LN | Entitlement Category constant for 'LOAN' |
| 7 | PC | Entitlement Category constant for 'PAYMENTS' |
| 8 | FU | Entitlement Category constant for 'FILE UPLOAD' |
| 9 | TF | Entitlement Category constant for 'CREDIT CARDS' |
| 10 | CC | Entitlement Category constant for 'CREDIT CARDS' |
| 11 | PFM | Entitlement Category constant for 'PERSONAL FIANANCE MANAGEMENT' |
| 12 | FX | Entitlement Category constant for 'FOREX'. |
| 13 | EBP | Entitlement Category constant for 'ELECTRONIC BILL PAYMENT' |
| 14 | MB | Entitlement Category constant for 'MOBILE APP MAINTENANCE' |
| 15 | PRL | Entitlement Category constant for 'PRE LOGIN' |
| 16 | OR | Entitlement Category constant for 'Originations' |
| 17 | ENUMP | Entitlement Category constant for 'ENUMERATION' |
| 18 | AUTH | Entitlement Category constant for Authentication |

If there are no groups or sub-groups into which the current transaction can be put then the user needs to add a new category and sub-category.

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|-------------------------------|
| 1 | SC | System Configuration |
| 2 | SR | System Rules |
| 3 | LMD | Limits Definition |
| 4 | UL | User Limits |
| 5 | LMP | Limits Package |
| 6 | TG | Task Group |
| 7 | SPC | Spend Category |
| 8 | SPCM | Spend Category Maintenance |
| 9 | GOC | Goal Category |
| 10 | PYP | Payment Purpose |
| 11 | PYR | Payee Restrictions |
| 12 | BCM | Bill Category Maintenance |
| 13 | BCMUBS | Bill Category Maintenance UBS |
| 14 | UM | User Management |
| 15 | MOB | Merchant Onboarding |
| 16 | AUTH | Authentication |
| 17 | MSS | Manage Security Settings |
| 18 | PPL | Password Policy |
| 19 | TXB | Transaction Blackout |

| Sr. No. | Entitlement Sub Category | Description |
|----------------|---------------------------------|------------------------------------|
| 20 | WW | Working Window |
| 21 | UGSM | User Group Subject Mapping |
| 22 | UGM | User Group Management |
| 23 | ALM | Alert Maintenance |
| 24 | BR | Brand Management |
| 25 | DSHBD | Dashboard Management |
| 26 | AL | Audit Log |
| 27 | AUL | Audit Logging |
| 28 | ABM | ATM Branch Maintenance |
| 29 | PDM | Product Mapping |
| 30 | ML | Mailers |
| 31 | MGB | Manage Brands |
| 32 | TXA | Transaction Aspects |
| 33 | PPI | Password Print Information |
| 34 | OWC | Origination Workflow Configuration |
| 35 | PL | Party to Party Linkage |
| 36 | PAC | Party Account Access |
| 37 | UAC | User Account Access |
| 38 | LUAC | Linked User Account Access |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|----------------------------------|
| 39 | LPAC | Linked Party Account Access |
| 40 | FIM | File Identifier Maintenance |
| 41 | UFIM | User FI Mapping |
| 42 | AWC | Approvals Workflow Configuration |
| 43 | SVR | Service Request |
| 44 | PP | Party Preference |
| 45 | RM | Rule Management |
| 46 | URM | User Report Mapping |
| 47 | RPM | Reports |
| 48 | RPV | Reports View |
| 49 | RPC | Corp Admin Reports |
| 50 | RPU | User Reports |
| 51 | MN | Mailbox Notifications |
| 52 | MAIL | Mails |
| 53 | ALT | Alerts |
| 54 | NO | Notification |
| 55 | PF | Profile |
| 56 | SS | Session Summary |
| 57 | LOC | ATM Branch Locator |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|----------------------|
| 58 | SCS | Security Setting |
| 59 | HELP | Help |
| 60 | CSAAD | CASA Account Details |
| 61 | CSASM | CASA Statement |
| 62 | CBR | Cheque Book |
| 63 | DC | Debit Card |
| 64 | CSACA | CASA Calculators |
| 65 | TDAD | TD Account Details |
| 66 | TDSM | TD Statement |
| 67 | TDTSN | TD Transactions |
| 68 | TDCA | TD Calculators |
| 69 | LNAD | Loan Account Details |
| 70 | LNSM | Loan Statement |
| 71 | LNTXN | Loan Transactions |
| 72 | LNCA | Loan Calculators |
| 73 | CCAD | CC Account Details |
| 74 | CCSM | CC Statement |
| 75 | CCTXN | CC Transactions |
| 76 | PYT | Payments Transfers |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|-----------------------------|
| 77 | ADPY | Adhoc Payment |
| 78 | IntPaye | Internal Payee |
| 79 | DP | Domestic Payee |
| 80 | DPR | Domestic Payer |
| 81 | IP | International Payee |
| 82 | DDP | Demand Draft Payee |
| 83 | BM | Biller Maintenance |
| 84 | CBM | Customer Biller Maintenance |
| 85 | DD | Demand Draft |
| 86 | BP | Bill Payment |
| 87 | RMT | Remittance |
| 88 | UP | Upcoming Payments |
| 89 | RF | Request Funds |
| 90 | FT | Favorite Transactions |
| 91 | PPP | Peer To Peer Payee |
| 92 | GO | Goal |
| 93 | SP | Spends |
| 94 | BD | Budget |
| 95 | AS | Alert Subscription |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|--|
| 96 | TPC | Third Party Consent |
| 97 | LM | Limits |
| 98 | FUTXN | File upload transactions |
| 99 | FUT | File Uploads |
| 100 | FUA | File Upload |
| 101 | TFLOC | Letter Of Credit |
| 102 | TFLOCA | Letter Of Credit Amendment |
| 103 | BAC | Bills And Collection |
| 104 | OWG | Outward Guarantee |
| 105 | OWGA | Outward Guarantee Amendment |
| 106 | GL | General |
| 107 | ACL | Activity log |
| 108 | PLT | Pre Login Transactions |
| 109 | SRFB | Service Request - Form Builder Sub Category, under category 'Admin Maintenance'. |
| 110 | FDM | Feedback Maintenance sub category under category 'Admin Maintenance' |
| 111 | NM | Nominee sub category under category Customer Servicing. |
| 112 | FXDB | Forex deal booking sub category under category 'Forex'. |
| 113 | PMNT | Payment maintenance sub category |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|---|
| 114 | RTM | Role Transaction Mapping sub category under category 'Admin Maintenance'. |
| 115 | EBR | Favorites sub category of EBPP under category 'Electronic Bill Payment'. |
| 116 | EBL | Bills sub category of EBPP under category 'Electronic Bill Payment'. |
| 117 | AGR | Favorites sub category of Account Aggregate under category 'Account Aggregation'. |
| 118 | EPY | Manage Billers sub category of EBPP under category 'Electronic Bill Payment' |
| 119 | MCL | Mobile Client sub category under category 'Mobile Application' |
| 120 | BFM | Manage Billers sub category of EBPP under category 'Electronic Bill Payment' |
| 121 | MSWPIN | Manage Sweep-in sub category under category 'Customer Servicing'. |
| 122 | FXDM | Forex Deal Maintenance sub category under category 'Admin Maintenance'. |
| 123 | ARM | Account Relationship Mapping sub category under category 'Admin Maintenance'. |
| 124 | FAM | 2 factor task auth maintenance. |
| 125 | ORP | Origination products sub category under category 'Pre-Login' |
| 126 | ANN | Account Nick name sub category under category 'CUSTOMER_SERVICING'. |
| 127 | BU | Business entities sub category under category 'CUSTOMER_SERVICING'. |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|---|
| 128 | ACPU | Access Point sub category under category 'CUSTOMER_SERVICING'. |
| 129 | FR | Fund Request sub category under category 'PAYMENTS'. |
| 130 | ANUM | Enumeration sub category under category Enumeration. |
| 131 | FCL | FATCA Compliance sub category under category 'CUSTOMER_SERVICING' |
| 132 | FDB | Feedback sub category under category 'CUSTOMER_SERVICING'. |
| 133 | SAD | Service Advisor sub category under category 'CUSTOMER_SERVICING' |
| 134 | ORC | Origination sub category under category 'Originations'. |
| 135 | AP | Approvals sub category under category 'CUSTOMER_SERVICING'. |
| 136 | ACPM | Access Point Maintenance sub category under category 'CUSTOMER_SERVICING'. |
| 137 | HDS | Help Desk Session sub category under category 'Admin Maintenance'. |
| 138 | APA | Access Point Account Access under category 'CUSTOMER_SERVICING'. |
| 139 | SECQUE | Security Question |
| 140 | TASK | Task |
| 141 | SB | SMS Banking |
| 142 | SRFL | Service Request - Form Listing Sub Category, under category 'Customer Servicing'. |
| 143 | UPS | Preferences of the user. |

| Sr. No. | Entitlement Sub Category | Description |
|---------|--------------------------|-----------------------------|
| 144 | BC | Base Configuration |
| 145 | FUFT | File Upload Funds Transfer. |
| 146 | FUP | File Upload Payee |
| 147 | GR | Generic Rest |

If there are no groups or sub-groups into which the current transaction can be put then the user needs to add the new category and sub-category. Guide to add new categories and sub-categories can be found here.

- **swaggerAnnotations:**

User should provide the details required for the swagger specification. In swagger documentation for each path(@path Annotation in REST) multiple operations(HTTP methods) can be defined . Swagger defines a unique operation as a combination of a path and an HTTP method. For each operation(HTTP method) there is a summary section and details section in the documentation. On clicking the operation box in swagger one can get details section. The details section consists of description, parameters, request body and responses. Further swagger details can be found here.

“swaggerAnnotations” takes the values relevant for generating the required swagger document. It takes summary, description, tags and API responses.

e.g.

```

6.1.3.3
"swaggerAnnotations": {
  "summary": "create method of class ForexDeal",      6.1.3.3.1
  "description": "Creates forex deal.",              6.1.3.3.2
  "tags": "",                                       6.1.3.3.3
  "apiResponses": [{                                6.1.3.3.4
  }
}

```

- **summary:** The value input in this field appears in the operation summary of the swagger documentation. The required content in the summary section for the current method(i.e. REST resource or REST URL +) should be provided here.
- **description:** User should provide the description of the method(i.e. REST resource).
- **tags:** Swagger uses tags to group the displayed operations. User should provide the relevant details of grouping and its description.
- **apiResponses:** For the current API operation all the possible responses are listed here.

e.g.

```

"apiResponses": [
  {
    "responseCode": "201",
    "description": "ForexDeal Created Successfully.",
    "content": {
  },
},
  {
    "responseCode": "400",
    "description": "Validation Failure",
    "content": {
  },
},
  {
    "responseCode": "500",
    "description": "Internal Server Error",
    "content": {
  },
}
]

```

- responseCode: Response code or HTTP status code of the REST response should be provided by the user.

e.g. "201" - if the request has been fulfilled and has resulted in one or more new resources being created.
 "400" – if the server cannot or will not process the request due to something that is perceived to be a client error.

- description: Description of the http response should be provided here.
- content: It consists of mainly two things mediaType and schema.
- MediaType: User should provide what media type the REST resource produces.

e.g.

"application/json" for JSON,

"application/xml" for XML.

The "content" node of the input JSON is depicted below in the image:

```

"content": { 6.1.3.3.4.1.3
  "mediaType": "application/json", 6.1.3.3.4.1.3.1
  "schema": { 6.1.3.3.4.1.3.2
    "implementation": { 6.1.3.3.4.1.3.2.1
      "packageName": "com.ofss.digx.app.forexdeal.dto", 6.1.3.3.4.1.3.2.1.1
      "className": "ForexDealCreateDTO" 6.1.3.3.4.1.3.2.1.2
    }
  }
}

```

- **schema:** It is used to describe the REST response body of the respective method. It can define an object or a primitive data type (for plain text responses) or a file. For the object or primitive data type in the response user must provide its `packageName` and `className`.
- **allocationTiers:** User should provide the details of the tiers (REST, SERVICE, DOMAIN, REPOSITORY) in which the method should be present. It is not a mandatory field. By default the method is created in all of the tiers

e.g.

```

"allocationTiers": ["REST","SERVICE", "DOMAIN", "REPOSITORY"] 6.1.3.4

```

- **MutualFunds typeDetailsDTOs:**

TypeDetailsDTOs array consists of multiple elements representing any Java type (i.e Class or Interface or enumeration) . Each element has child nodes “fields”, “type”, “typeAttrDTO” and “enumeration”.

```

"typeDetailsDTOs": [{
  "fields": [
    "typeAttrDTO": {
      "custom": true,
      "exists": false
    },
    "enumeration": true
  ]
}

```

- **fields(*):**

All the variables which user intends to declare in the domain are taken care of by this node/key. This node/key stands for the Java type (i.e. Class or Interface or enumeration) or variables or fields to be used in the domain (here ForexDeal) and its required details. This node has further child nodes “type” i.e. Java type for variable, “attrs” for attributes of the Java type or variable or field and “detailsDTO” for other details of the Java type or variables or fields. Other details are provided in case the field is in itself is a type containing further fields. Part of the JSON carrying the sample values of this node is depicted below:

```

"detailsDTOs": [
  {
    "typeDetailsDTOs": [
      {
        "fields": [
          {
            "type": {
              "packageName": "com.ofss.digx.enumeration.forexdeal",
              "className": "DealType"
            },
            "name": "dealType",
          },
          {
            "type": {
              "packageName": "",
              "className": "ForexDealKey"
            },
            "name": "forexDealKey",
            "varAttrs": {
              "key": true
            }
          }
        ]
      }
    ]
  }
]

```

- **type(*)**: “type” here is the Java type(Class or Interface or enumeration) which takes the qualified name (i.e package name and class name) as input. It consists of further child nodes depicted below:
 - packageName: User should provide the package name of the variable to be declared. It must be left empty if the variable is a domain key
 - className: User must provide the class name of the variable to be declared
- **name(*)**: User should provide the reference of the variable at this node/key as can be seen in the figure.
 - e.g

```
"name": "dealType", — 7.1.2
```

- **varAttrs** : A variable declared can be an amount, account, party, queryParam or key of the domain and the corresponding attribute has to be marked as ‘true’.
 - key: In a domain there is key which is a unique identifier for each domain object. User must provide this key value as “true” if this field is the key of the domain and “false” in case it is a normal variable.
 - A key can be a composite key i.e. multiple fields combined together to form key. In case of composite key user must provide the “key” attributes as true for all those fields which form the composite key.

```

"varAttrs": {
  .....
  "key": true
}

```

- amount: User must provide this amount value as “true” if a particular field represents an amount.

- **account:** User must provide this attribute as “true” if a particular field represents an amount.
 - **queryParams:** Query Parameter is used to sort/filter a specific resource. This variable attribute is marked as “true” if it represents a query parameter.
 - **party:** User must provide this attribute as “true” if a particular field represents a party.
 - **type:** ‘type’ is another node of typeDetailsDTOs. It is again the Java type(Class or Interface or enumeration) which takes the qualified name (i.e package name and class name) of the custom domain as an input.
 - **typeAttrDTO:** A variable can be a Java class of an API or can be a user defined class. If it is a user defined class or custom class it is possible that it is an existing domain or variable.
 - **custom:** A declared field can be a Java type (Class or Interface) which is not a class of existing Java packages instead are defined by the user. Those user defined fields are custom fields. e.g String class is a class of existing Java package java.lang so String fields are not custom fields but com.example. Student is a user defined Class (hence custom filed).

This node is a Boolean which takes values depending upon the variable declared is a class of an API or it is a user defined class. If this key carries “true” it means the class is user defined class.

 - **exists:** This is also a Boolean which takes values depending upon the variable declared in the domain is an existing domain or it is new domain at all.
- If user provides this value as “false” then a completely new domain is created by the tool. User should provide this as “true” in case the domain already exists.
- **enumeration:** This key is a Boolean. If this variable is an enumeration then user must provide “true” value to this JSON node. It is not a mandatory field. If user doesn’t provide any value the tool assumes it to be a Java type other than enumeration.

```

"typeAttrDTO": {
    "custom": true,
    "exists": false
},
"enumeration": true

```

```

"typeAttrDTO": {
    "custom": true,
    "exists": false
},
"enumeration": null

```

If the user is going to declare a variable which in itself is a domain or is another class (e.g enumeration) carrying fields then user must provide the details of the variables to be declared in that domain or class. A separate element should be added in the TypeDetailsDTOs array by the user to represent such domain. Part of the JSON with sample values are depicted below:

```

{
  "fields": [
    {
      "name": "SPOT",
      "mockValue": "S"
    },
    {
      "name": "FORWARD",
      "mockValue": "F"
    }
  ],
  "type": {
    "packageName": "com.ofss.digx.enumeration.forexdeal",
    "className": "DealType"
  },
  "typeAttrDTO": {
    "custom": true,
    "exists": false
  },
  "enumeration": true
},

```

Here user should provide the details of the fields to be declared in this new domain TypeDetailsDTO. If this new element is of enumeration type then 'enumeration' will carry 'true' as shown in the snippet. 'fields' asks for type, name, mock value and further details.

- name: User should provide the details of the variable to be declared in the new class or enumeration.
- mockValue: Mock value must be provided in case of enumeration. The user should provide the value to be obtained in each constant declared in the enumeration. It can be seen in the below snippet.
- genericType: The below snippet demonstrates creation of fields pertaining to classes supporting generics. The most frequently encountered example being list, let's see how to have a list of strings as a field in a domain.

```
"fields": [  
  {  
    "type": {  
      "packageName": "java.lang",  
      "className": "String"  
    },  
    "name": "id",  
    "varAttrs": {  
      "key": true  
    }  
  },  
  {  
    "type": {  
      "packageName": "java.util",  
      "className": "List"  
    },  
    "name": "greetings",  
    "genericType": [  
      {  
        "packageName": "java.lang",  
        "className": "String"  
      }  
    ]  
  }  
],  
}
```

Within the “type” of the field provide a generic type JSON array where each element of the array provides the package name and the class name of the parameterized types. In the example above the parameterized type is a String class of Java.

[Home](#)

7. FAQs

1. How do I specify which field is used for READ operation ?

To specify a field to be used in the READ operation the user must provide the value of the “key” of the “attributes” of the particular field in the input JSON as true. Please refer “fields” in the section 5 JSON explained.

2. Can we edit the generated source code ?

Yes, one can import the generated source code (after running gradle task “gradle eclipse”) from <APIToolkit_HOME>/output folder. Make sure the next tasks “gradle build” , “gradle thirdpartygen” are executed after it.

3. Can I run the gradle tasks at any windows location?

No, User must run the gradle tasks at the <APIToolkit_HOME>.

4. Is it mandatory to execute gradle thirdpartygen task?

The task thirdpartygen generates xml with mock values only for simulation purpose of third party setup. It further packages these xml into ExtxfaceSimulatorMDB.war which is deployed only in a simulation environment. In environments interacting with actual hosts this is not needed.

5. Where are the SQL scripts generated?

At <APIToolkit_HOME>/ output/seed/oracle.

6. Can I generate two different services in the same module?

Yes, user just need to place the two JSONs for respective services at <APIToolkit_HOME>/input/json.

7. Can I generate two different services in two different modules?

Yes, user just need to write two different JSONs with the required moduleNames in each JSON. Place the two JSONs for respective services at <APIToolkit_HOME>/input/json. That's it.

[Home](#)