

# Oracle Banking APIs

Extensibility API Toolkit Guide

Release 18.2.0.0.0

Part No. E97823-01

June 2018

**ORACLE®**

Extensibility API Toolkit Guide

June 2018

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/](http://www.oracle.com/financialservices/)

Copyright © 2018, 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

<b>1. Preface.....</b>	<b>5</b>
1.1 Intended Audience .....	5
1.2 Documentation Accessibility .....	5
1.3 Access to Oracle Support .....	5
1.4 Structure.....	5
1.5 Related Information Sources.....	5
<b>2. Introduction.....</b>	<b>6</b>
<b>3. Technology Stack.....</b>	<b>7</b>
<b>4. Pre-requisites.....</b>	<b>8</b>
<b>5. Write your First service .....</b>	<b>9</b>
5.1 Obtain Toolkit .....	9
5.2 Set Environment Variables.....	9
5.2.1 Windows.....	9
5.2.2 Linux.....	11
5.2.3 IOS .....	11
5.3 Write JSON .....	12
5.4 Include javax.ws.rs-api-2.0.jar.....	14
5.5 Execute Gradle tasks .....	14
5.6 Deploy EARs .....	19
5.6.1 CZ EARS Deployment .....	19
5.6.2 REST IDM Deployment.....	19

<b>6. JSON Explained.....</b>	<b>21</b>
<b>7. FAQs.....</b>	<b>40</b>

# 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.

*Introduction* provides brief information on the overall functionality covered in the User Manual.

The subsequent chapters provide information on transactions covered in the User Manual.

Each transaction is explained in the following manner:

- Introduction to the transaction
- Screenshots of the transaction
- The images of screens used in this user manual are for illustrative purpose only, to provide improved understanding of the functionality; actual screens that appear in the application may vary based on selected browser, theme, and mobile devices.
- Procedure containing steps to complete the transaction- The mandatory and conditional fields of the transaction are explained in the procedure.

If a transaction contains multiple procedures, each procedure is explained. If some functionality is present in many transactions, this functionality is explained separately.

## 1.5 Related Information Sources

For more information on Oracle Banking APIs Release 18.2.0.0.0, refer to the following documents:

- Oracle Banking APIs Licensing Guide
- Oracle Banking APIs 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 ear files.
- generates various database scripts required for the functioning of the services.

### 3. Technology Stack

<b>Software</b>	<b>Version</b>
Java	Java JDK or JRE version 8
Gradle	gradle-4.7
OB APIs	18.2

## 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(refer Appendix A for Gradle installation)
- User must provide “**javax.ws.rs-api-2.0.jar**” in the <APIToolkit\_HOME>/output/lib/provided-compile folder.

## 5. Write your First service

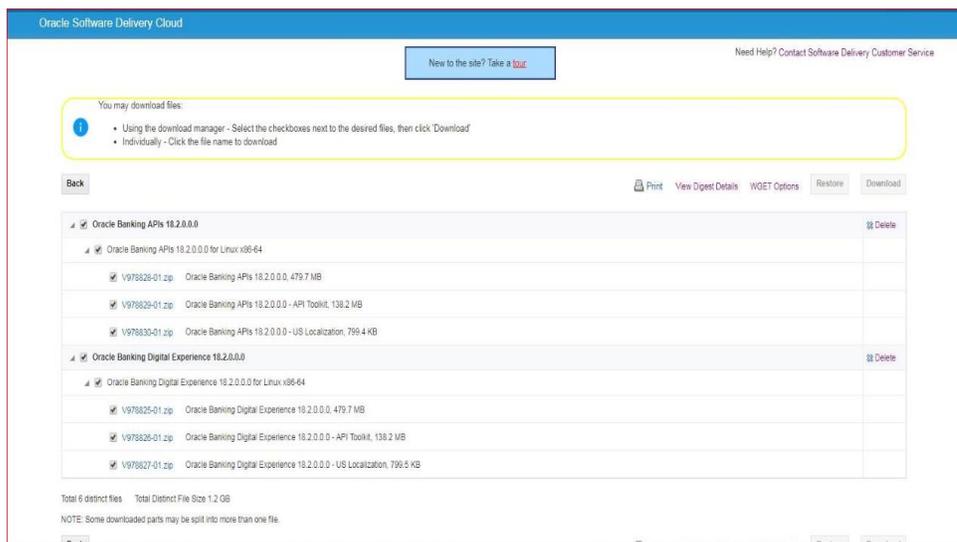
Let us start with a Hello World Service. This section will help you in writing an end to end Hello World service. The following needs to be done:

- 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 (EARs) and the database scripts (The order in which the tasks are executed must be maintained as provided in the explanation)
- Carry out necessary modifications in the REST IDM EARs. Deploy the EARs 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

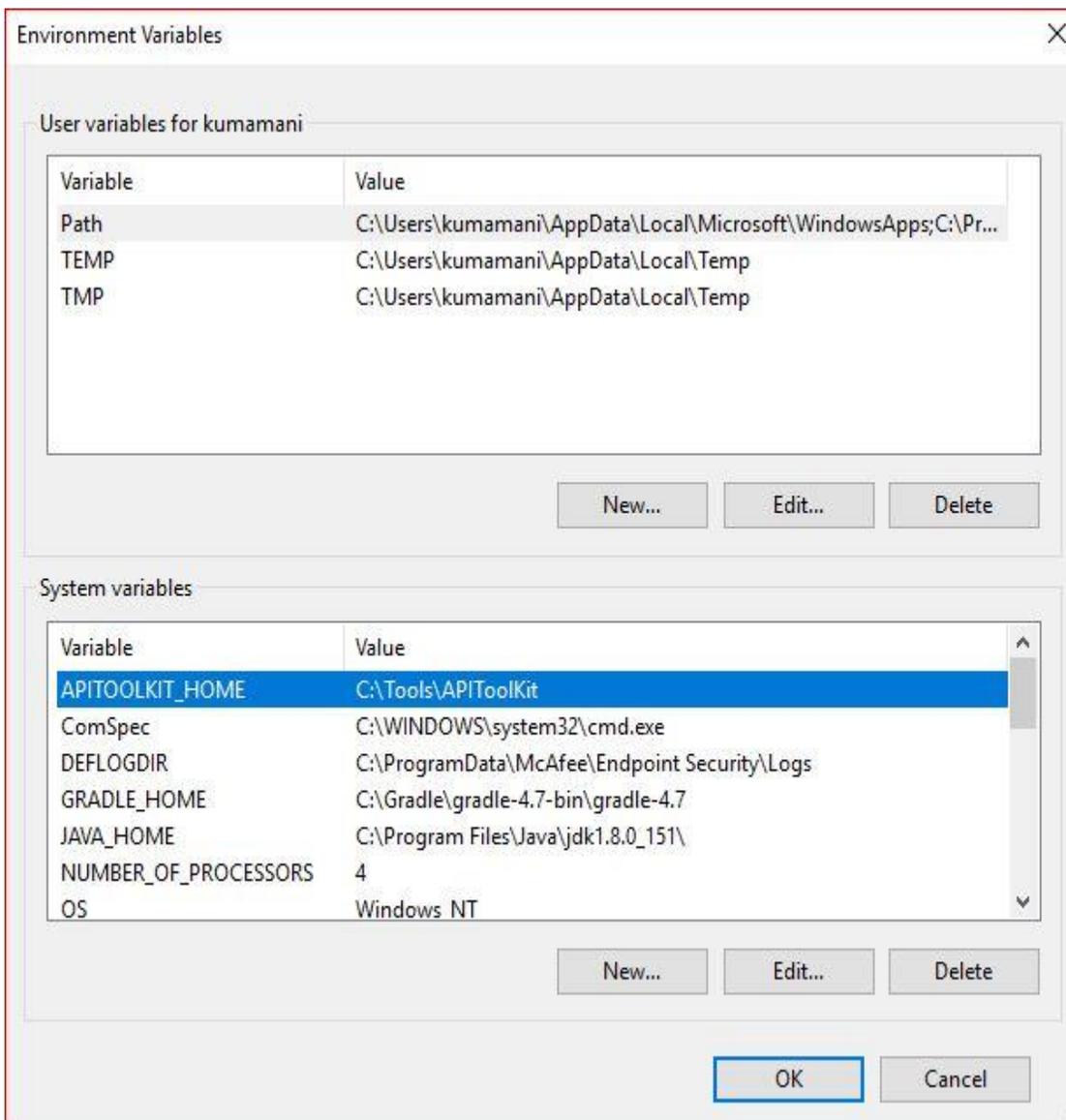
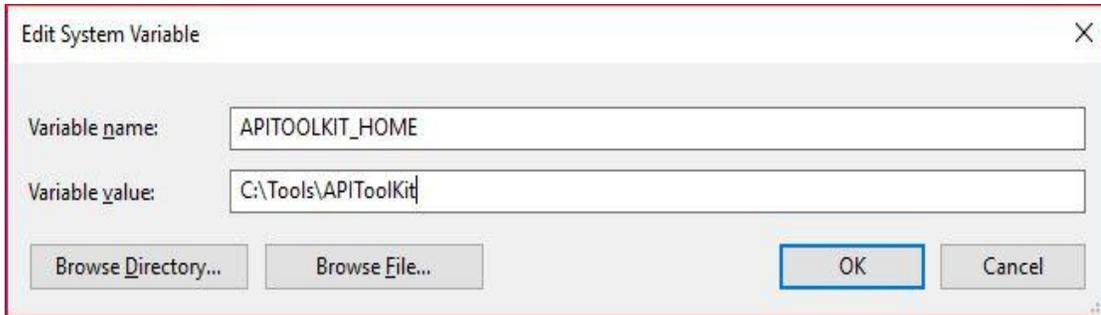
User should download the API ToolKit zip from the Oracle Software Delivery Cloud portal



### 5.2 Set Environment Variables

#### 5.2.1 Windows

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



## 5.2.2 Linux

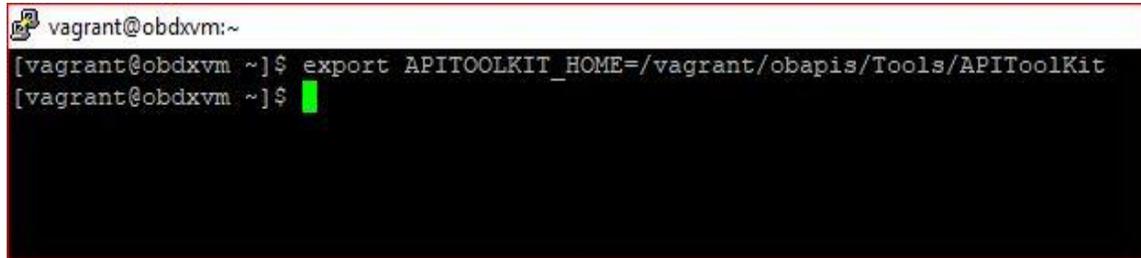
User should run the following command on the terminal:

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

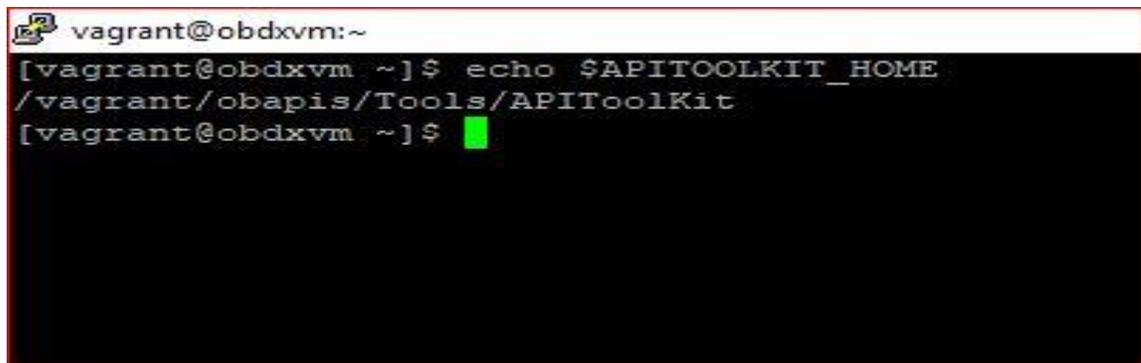
e.g.

```
export APITOOLKIT_HOME=/vagrant/obapis/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/obapis/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/53/110hlz792z9gkqctfnm7rcr0000gn/T/
Apple_PubSub_Socket_Render=/private/tmp/com.apple.launchd.Ujht44psUw/Render
TERM_PROGRAM_VERSION=400
OLDPWD=/Users/obdxuser/Documents/uiworkbench
TERM_SESSION_ID=485C48B7-7200-4E2E-BF44-0022000FE58
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 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 :

```

{
  "domainName": "HelloWorld",           — 1
  "moduleName": "hello",                 — 2
  "subModuleName": "world",              — 3
  "moduleCode": "HW",                    — 4
  "uri": "/hello",                       — 5
  "methods": [{                           — 6
  "fields": [{                             — 7
}

```

For details of each and every field of the input JSON please refer section 6(JSON Explained). It is known that every functionality in OB API 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’, ‘update’ which is provided in the JSON as an array of methods and it consists of various fields. The fields (7) and methods (6) details put in the HelloWorld input JSON is given below:

```

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

```

The above snippet is of the fields or variables to be declared in the HelloWorld. It is an array in which each element represents one field or variable. You have 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. For details of preparing fields array please refer 'fields' explanation in the JSON Explained section(Section 6).

```

"methods": [
  {
    "name": "create",
    "operationType": "CREATE",
    "methodAttributes": {
      "task": {
        "taskType": "COMMON",
        "moduleType": "COMMON"
      },
      "entitlements": {}
    }
  },
  {
    "name": "read",
    "operationType": "READ",
    "methodAttributes": {
      "task": {
        "taskType": "COMMON",
        "moduleType": "COMMON"
      },
      "entitlements": {}
    }
  }
]

```

```

{
  "name": "update",
  "operationType": "UPDATE",
  "methodAttributes": {
    "task": {
      "taskType": "COMMON",
      "moduleType": "COMMON"
    },
    "entitlements": {
    }
  }
}
],

```

The above snippet is of the 'methods' in the HelloWorld, which again is an array, each element representing one method. In HelloWorld we have three methods create, read and update hence three elements. The methods has certain attributes like taskType and moduleType. Since 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 6 JSON Explained](#).

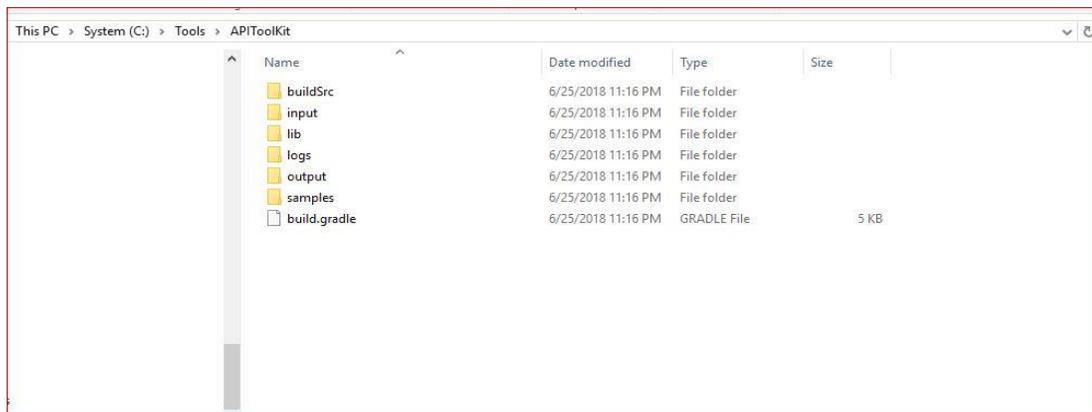
## 5.4 Include javax.ws.rs-api-2.0.jar

As part of the prerequisite for using this tool the "javax.ws.rs-api-2.0.jar" must be placed at the <APITOOLKIT\_HOME>/output/lib/provided-compile folder.

## 5.5 Execute Gradle tasks

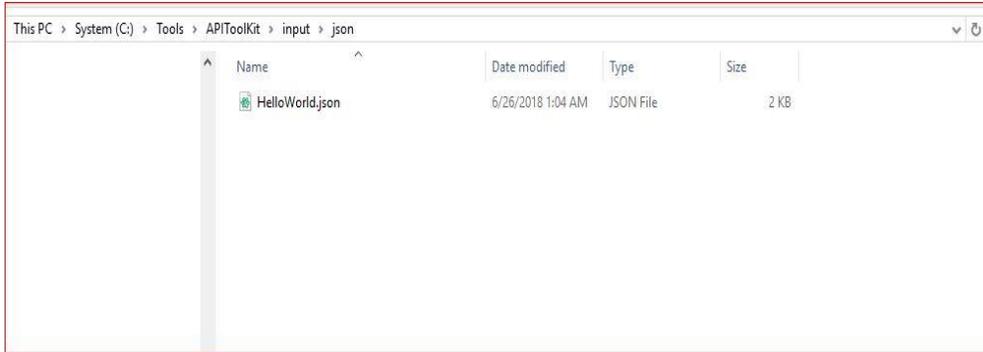
Now that you are ready with your input JSON and the pre-requisites are done you are eligible to run required gradle tasks. There is a list of gradle tasks which needs to be run in the provided order. These tasks will generate source code, furnish eclipse projects for the generated code, build the source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

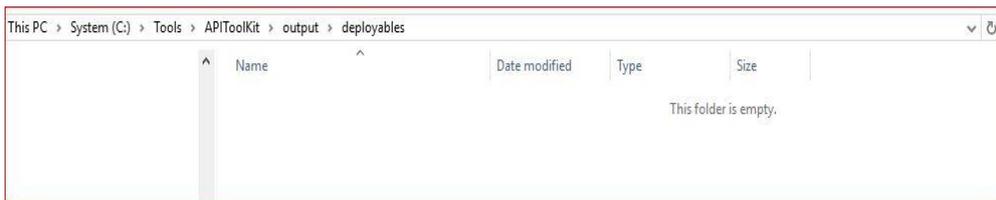


- output folder

Output folder has deployables, scripts, swagger, etc.

- deployables

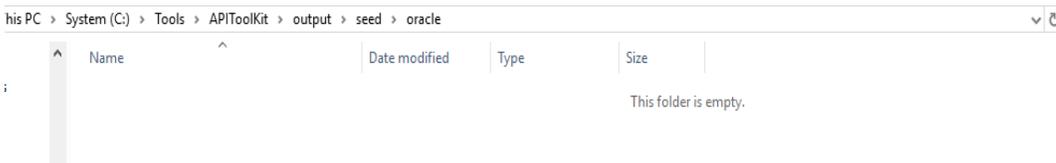
It is where the deployables (EARs) will be generated. The exact location for the generated deployables can be seen below:



It is currently empty since gradle tasks are not executed to generate it.

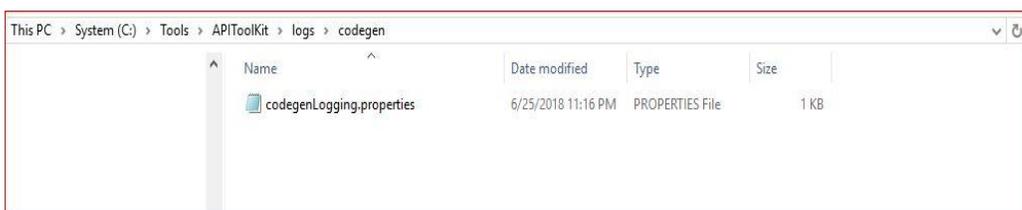
- seed/oracle

It is where the required sql scripts will be generated.



- logs folder

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



- gradle 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 and have the external library "javax.ws.rs-api-2.0.jar" at <APIToolKIT\_HOME>/output/lib/provided-compile

```
ca. 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

This task will get you the eclipse projects for the generated source code. It generates the required eclipse project and settings file for the purpose. It is not mandatory to execute this task. One needs to execute this task if he/she wants to check the code in the eclipse IDE or do some modifications in the generated code.

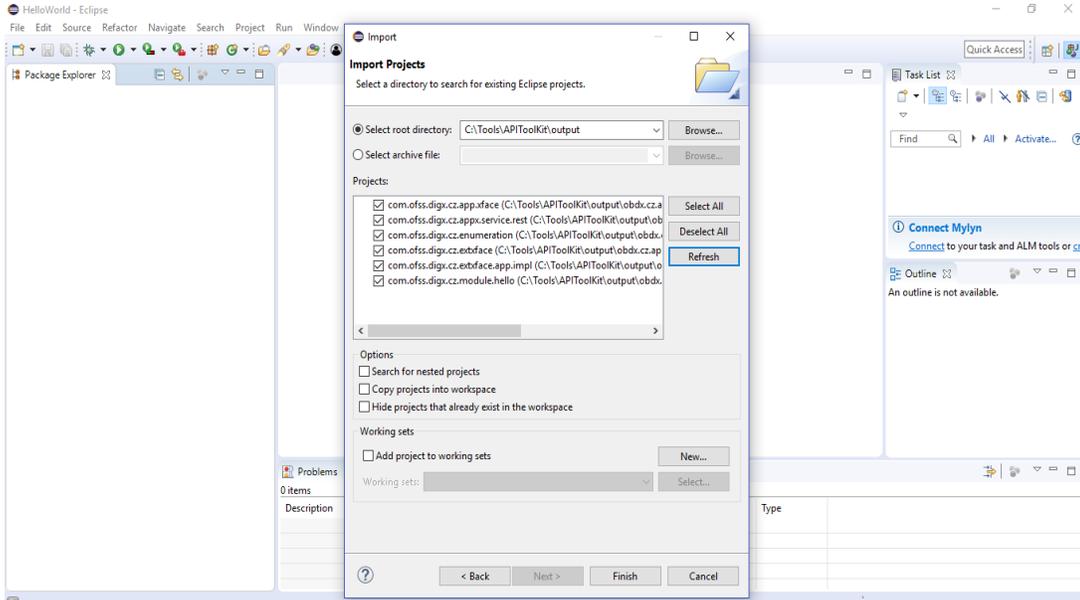
```
ca. C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.16299.15]
(c) 2017 Microsoft Corporation. All rights reserved.

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

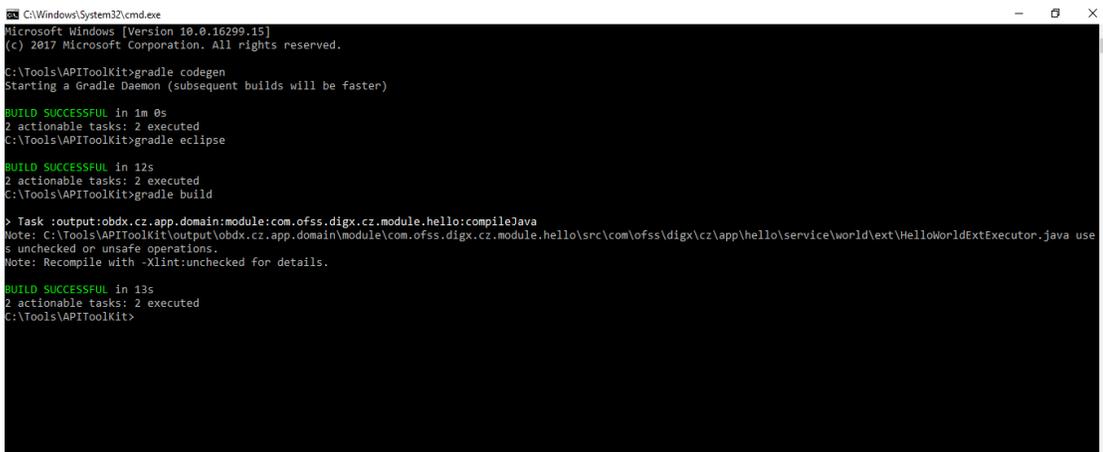
BUILD SUCCESSFUL in 1m 0s
2 actionable tasks: 2 executed
C:\Tools\APIToolKit>gradle eclipse

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

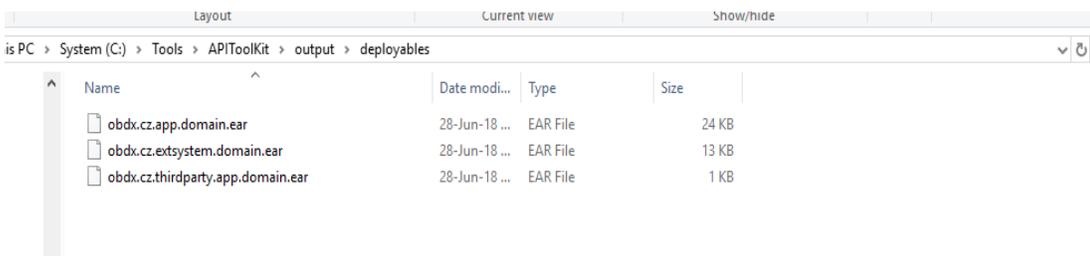
Now the project can be imported into eclipse IDE from the <APIToolKit\_HOME>/ output folder.



- gradle build(\*)



Browse to the <APIToolKIT\_HOME>/ output/deployables folder.



Here are the EARs generated through the build task. Currently it is missing the thirdparty EARs which will be generated through thirdparty scripts.

- gradle thirdpartygen

This task will get you the third party deployables. It basically generates the simulation mdb EARs 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 docgen(\*)

You will get the swagger JSON here which can be consumed for generating swagger API.

```

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 docgen

BUILD SUCCESSFUL in 16s
1 actionable task: 1 executed
C:\Tools\APIToolKit>

```

#### 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

## 5.6 Deploy EARs

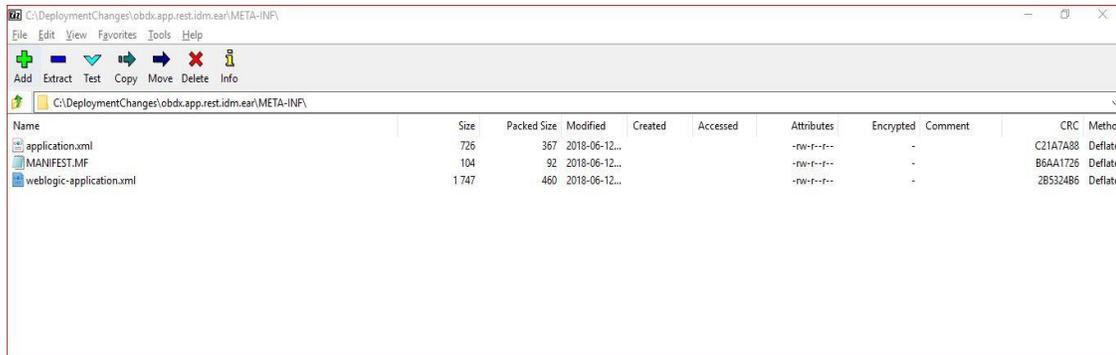
### 5.6.1 CZ EARS Deployment

### 5.6.2 REST IDM Deployment

The REST IDM (“obdx.app.rest.idm.ear”) already deployed on the server must be modified in order to get the CZ EARs deployment working.

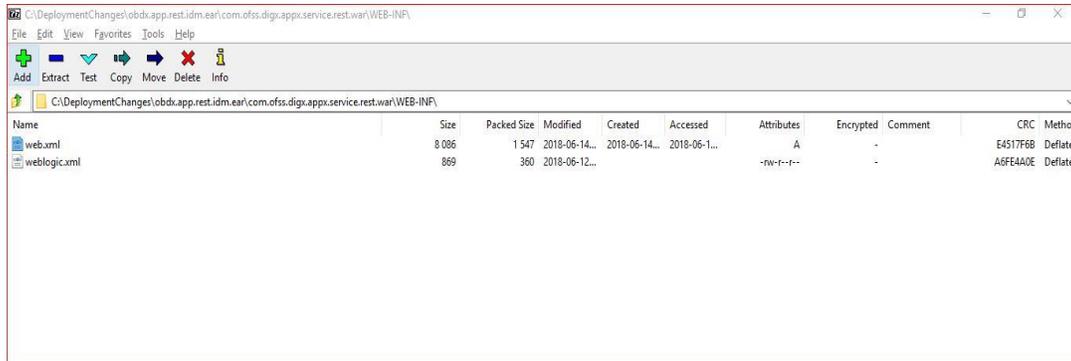
For this purpose two modifications must be done:

1. Add the CZ EARs in the weblogic-application.xml file located at obdx.app.rest.idm.ear/META-INF/weblogic-application.xml. Changes are depicted below:



2. Add CZ Servlet and the servlet mapping in the web.xml file located at obdx.app.rest.idm.ear/com.ofss.digx.appx.service.rest.war/WEB-INF/web.xml. The changes are depicted below:

## Write your First service



web.xml file already has so many servlets, do not change them, just add the CZ Servlet.

```
<!-- CZServlet -->
<servlet>
  <description>Custom domain servlet</description>
  <servlet-name>CZServlet</servlet-name>
  <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
  <!--Register resources and providers under my.package. -->
  <init-param>
    <param-name>javax.ws.rs.Application</param-name>
    <param-value>com.ofss.digx.appx.Application</param-value>
  </init-param>
  <!-- Register my custom provider -->
  <init-param>
    <param-name>jersey.config.server.provider.classnames</param-name>
    <param-value>com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider;org.glassfish.jersey.jackson.JacksonFeature;org.glassfish.jersey.media.multipart.MultiPartFeature</param-value>
  </init-param>
  <init-param>
    <param-name>obdx.rest.resource.filename</param-name>
    <param-value>cz.resources</param-value>
  </init-param>
  <init-param>
    <param-name>openApi.configuration.prettyPrint</param-name>
    <param-value>true</param-value>
  </init-param>
  <load-on-startup>3</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>CZServlet</servlet-name>
  <url-pattern>/v1/cz/*</url-pattern>
</servlet-mapping>
```

In the above CZ servlet the “Register my custom provider” part of init-params tag is depicted below:

```
<!-- Register my custom provider -->
<init-param>
  <param-name>jersey.config.server.provider.classnames</param-name>
  <param-value>com.fasterxml.jackson.jaxrs.json.JacksonJaxbJsonProvider;org.glassfish.jersey.jackson.JacksonFeature;org.glassfish.jersey.media.multipart.MultiPartFeature</param-value>
</init-param>
```

## 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
  "fields": [                         ——— 7
}

```

- 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.

eg

```
"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). In the example of deal booking functionality, the path exposed is shown below:

```
"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

eg

```

— 6.1.3.1
"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": ["approval", "audit", "limit", "2fa"], — 6.1.3.1.4
  "moduleType": "FOREXDEAL" — 6.1.3.1.5
},

```

- 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 Appendix A.

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
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
18	RT	REPORTS
19	LC	LETTEROF CREDIT
20	BL	BILL
21	CM	COMMON
22	BM	BENEFICIARY MAINTAINANCE

23	GR	GENERICREST
24	MT	ADMIN_MAINTENANCE
25	FX	FOREXDEAL
26	MB	MOBILE

- 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'.
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'

Sr.No	Entitlement Category	Description
10	CC	Entitlement Category constant for 'CREDIT CARDS'
12	PFM	Entitlement Category constant for 'PERSONAL FIANANCE MANAGEMENT'
13	FX	Entitlement Category constant for 'FOREX'.
14	EBP	Entitlement Category constant for 'ELECTRONIC BILL PAYMENT'
15	MB	Entitlement Category constant for 'MOBILE APP MAINTENANCE'
16	PRL	Entitlement Category constant for 'PRE LOGIN'
17	OR	Entitlement Category constant for 'Originations'
18	ENUMP	Entitlement Category constant for 'ENUMERATION'
19	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
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
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
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
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 Maintainance
84	CBM	Customer Biller Maintainance
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
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

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'.
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 Quesiton
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.
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

```

6.1.3.3.4
"apiResponses": [
  { 6.1.3.3.4.1
    "responseCode": "201", 6.1.3.3.4.1.1
    "description": "ForexDeal Created Successfully.", 6.1.3.3.4.1.2
    "content": { 6.1.3.3.4.1.3
  }
},
  { 6.1.3.3.4.2
    "responseCode": "400", 6.1.3.3.4.2.1
    "description": "Validation Failure", 6.1.3.3.4.2.2
    "content": { 6.1.3.3.4.2.3
  }
},
  {
    "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.

eg

"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

```

- **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:

## HelloWorld Fields:

```

"fields": [ 7
{ 7.1
  "type": { 7.1.1
    "name": "dealType", 7.1.2
    "attrs": { 7.1.3
    "detailsDTO": [{ 7.1.4
  },
  { 7.2
    "type": { 7.2.1
      "name": "forexDealKey", 7.2.2
      "attrs": { 7.2.3
      "detailsDTO": [{ 7.2.4
    },
  },
}

```

- **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:

```

"type": { 7.1.1
  "packageName": "com.ofss.digx.enumeration.forexdeal", 7.1.1.1
  "className": "DealType", 7.1.1.2
  "enumeration": true 7.1.1.3
},

```

```

"type": { 7.2.1
  "packageName": "", 7.2.1.1
  "className": "ForexDealKey" 7.2.1.2
},

```

- **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 (can be seen in 2nd fig 7.2.1.1)
- **className:** User must provide the class name of the variable to be declared.
- **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.

- **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
```

- **attributes** : 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. Also the variable declared can be a key of the domain. All these parameters can be defined using the below nodes/keys:

```

      — 7.1.3
"attrs": {
  "custom": true,      — 7.1.3.1
  "exists": false     — 7.1.3.2
},

```

```

      — 7.2.3
"attrs": {
  "custom": true,      — 7.2.3.1
  "exists": false,    — 7.2.3.2
  "key": true         — 7.2.3.3
},

```

- **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 field).

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.

- **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 forms the composite key.

- **detailsDTO:** 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. Part of the JSON with sample values of this node are depicted below:

```

"detailsDTO": [ 7.1.4
  { 7.1.4.1
    "name": "SPOT", 7.1.4.1.1
    "mockValue": "S" 7.1.4.1.2
  },
  { 7.1.4.2
    "name": "FORWARD", 7.1.4.2.1
    "mockValue": "F" 7.1.4.2.2
  }
]

```

Here user should provide the details of the fields to be declared in this new domain field. It is same as the field (parent node) . It asks for type, name, attrs, mock value and further details of the field.

- name: User should provide the details of the variable to be declared in the new class or enumeration.
- mockValue: Same as mockValue of the fields node of the parent
- 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 detailsDTO node of the child.

e.g

```

{ 7.1.4.1
  "name": "SPOT", 7.1.4.1.1
  "mockValue": "S" 7.1.4.1.2
},

```

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

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.

## 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. Where are the EARs for the deployment generated?

One can get the built EARs after running respective gradle tasks("gradle build" and "gradle thirdpartygen") at <APIToolkit\_HOME>/ output/deployables.

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

No, User must run the gradle tasks at the <APIToolkit\_HOME>.

### 5. 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.ear which is deployed only in a simulation environment. In environments interacting with actual hosts this is not needed.

### 6. Where are the sql scripts generated?

At <APIToolkit\_HOME>/ output/seed/oracle.

### 7. 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.

### 8. 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.