TEMENOS
The software specialist for banking and finance

IRIS R18 - User Guide

FF 95594731

Table Of Contents

Introduction

Purpose of the Guide

This document helps the user to design models and resources for the data service which is generated from the designed models to communicate with T24. This allows the model to be used for all T24 users. This document also helps to configure and deploy IRIS R18 provider and publisher components of T24 infrastructure.

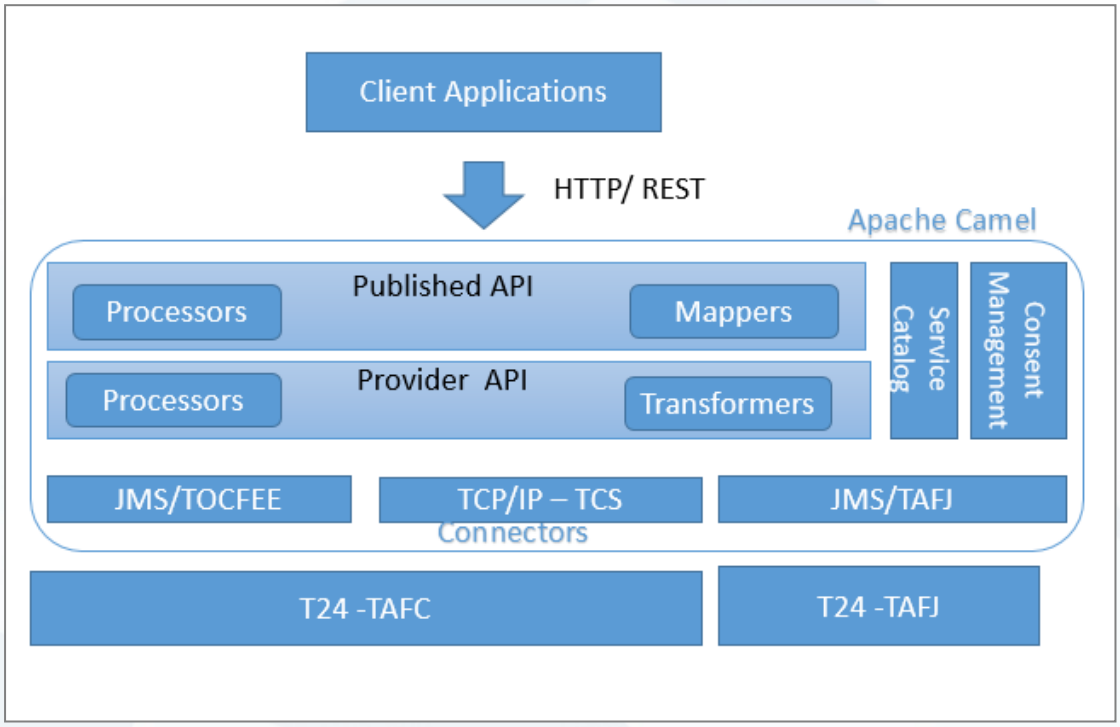
Intended Audience

This User Guide is intended for the use of External and Partner teams to deploy IRIS R18 Module.

Overview

IRIS R18 is a lightweight, REST standards based solution that uses OFS message format to communicate with T24. This allows the solution to be used for all T24 customers across release versions.

The below screenshot shows the overview of the solution and its components.



- Provider API Layer : hosts REST APIs for T24 modules
- Published API: hosts Published API. Used to transform provider API to an interface external applications or standards message formats.
- Connectors: Standard connectors available to connect to T24 based on the T24 deployment (TAFC/TAFJ).
- Consent Management: management layer to apply access control while servicing API requests from client application.
- Service Catalogue: hosts SWAGGER API documentation for Provider and Publisher APIs.
- Apache Camel: is an open source frameworks used to define rest end points, transforms data and route messages in Publisher and Provider layer.

Prerequisites

The following tools should be available for development and testing purpose:

Item	Link
Apache Camel	http://camel.apache.org/
Maven	https://maven.apache.org/
Spring	https://projects.spring.io/spring-framework/
Open API Spec, fka Swagger	https://swagger.io/
Postman	https://www.getpostman.com/
Eclipse	http://www.eclipse.org/
Junit	http://junit.org

Maven, Java

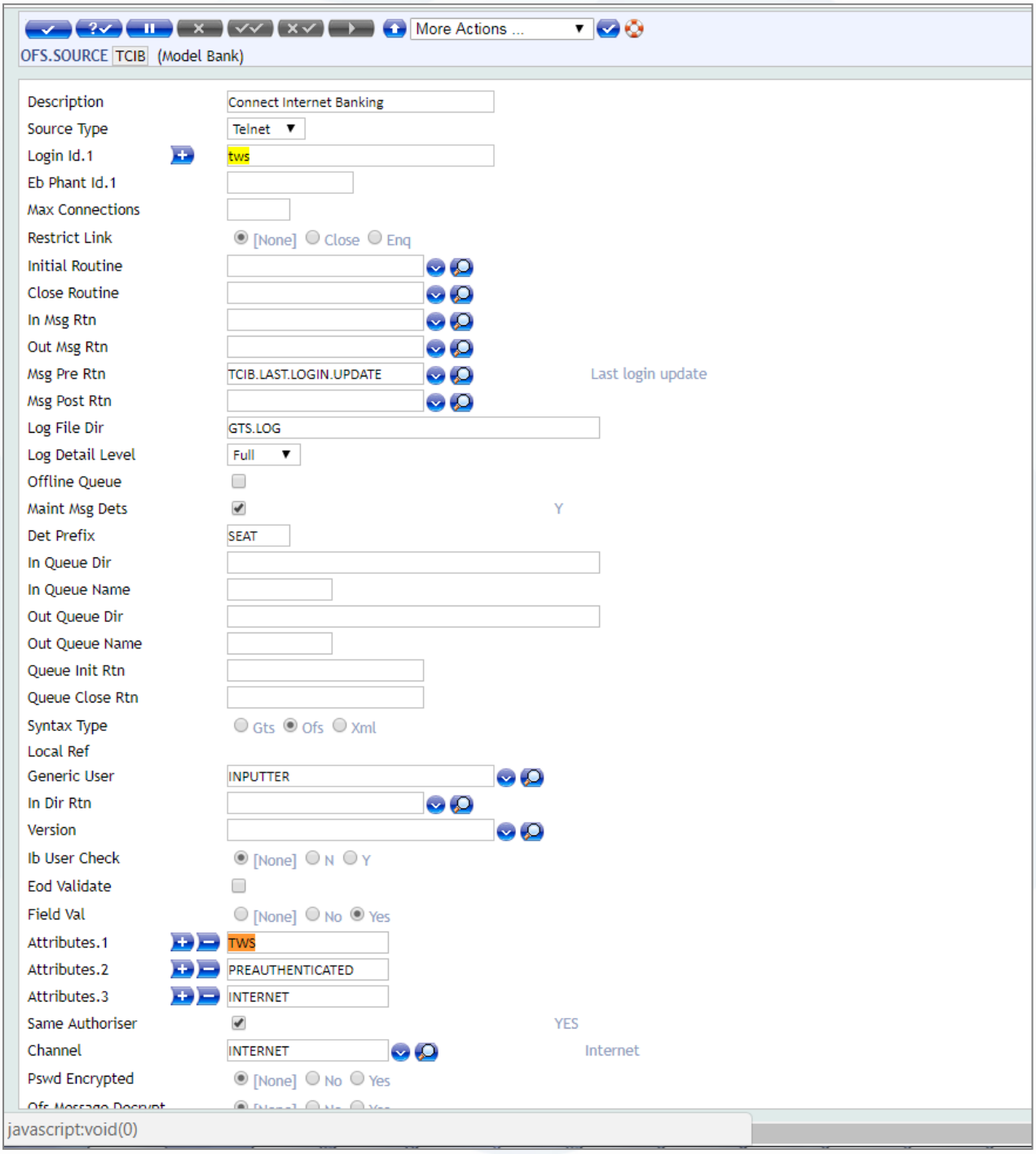
Maven and Java 8 are required. You can download the other dependencies from the Maven repository.

Key Application configuration

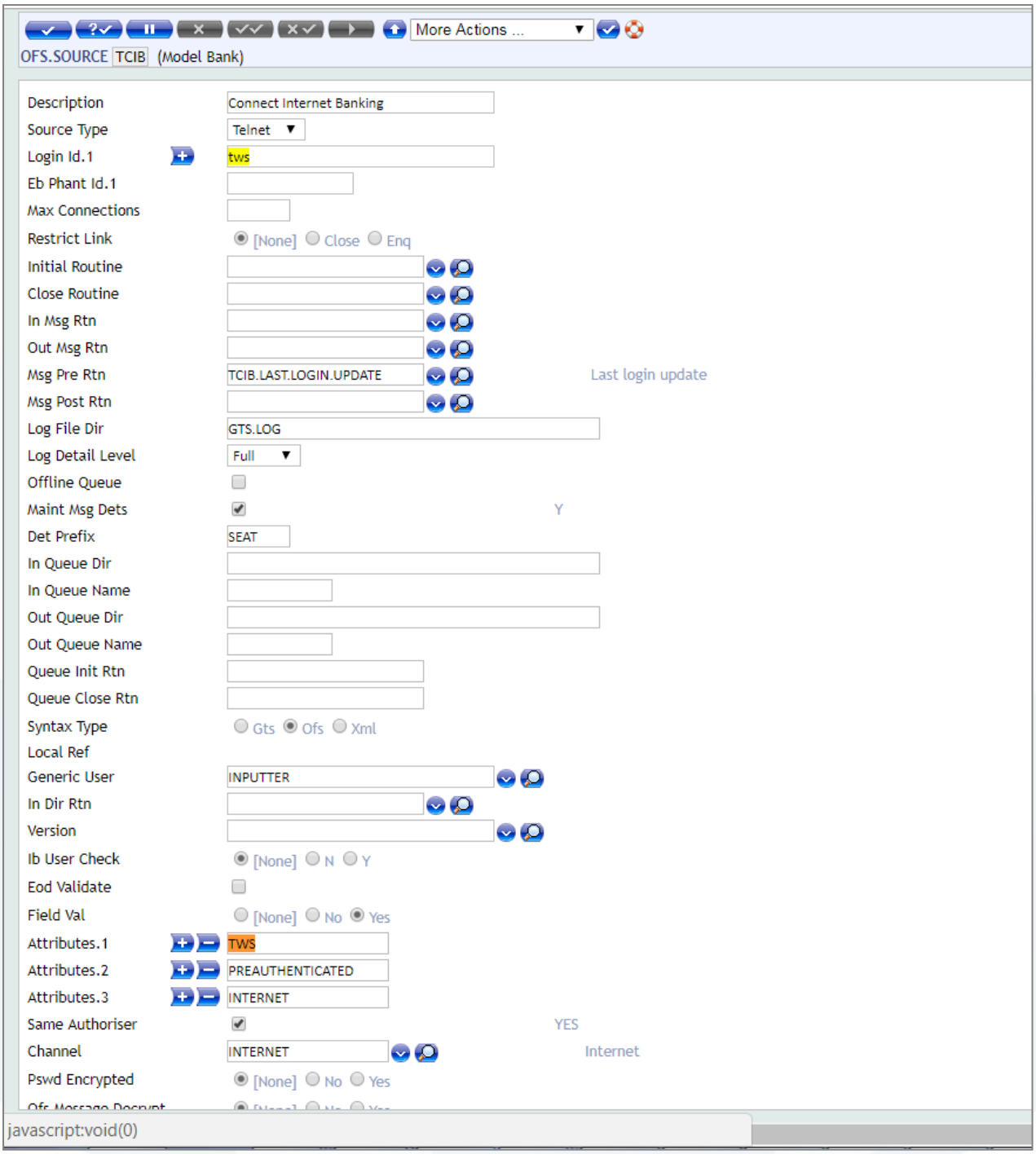
T24 Application Configuration

OFS.SOURCE Configuration

1. **FIELD Val** in order to enable Field validation for OFS Messages specifically for AA activities, configure the attribute **Field Val** as Yes of OFS.SOURCE



2. Remove the TWS attribute in the OFS.Source



Note: Restart the T24 Application to apply the configuration for the application runtime.

API Dev

An API defined in Swagger 2.0 is imported into an API project. Import process will create the Camel routes, mock responses and wire the routes to the mock responses. This is done using a Maven plugin, either from your IDE or from the command line.

This module covers the following topics.

Getting Started

The table below describes the technical terminologies used in the solution.

Terms	Description
Swagger	Swagger is an open source software framework backed by a large ecosystem of tools that helps developers design, build, document, and consume RESTful Web services.
Apache Camel	Apache Camel is an open source framework for message-oriented middleware with a rule-based routing and mediation engine that provides a Java object-based implementation of the Enterprise Integration Patterns using an application programming interface (or declarative Java domain-specific language) to configure routing and mediation rules.
Archetype	It is defined as an original pattern or model from which all other things of the same kind are made.
Maven	Maven is a build automation tool used primarily for Java projects.
Inventory	It is a file that provides the information of an API in core banking terms. It helps in creating the service implementation by generating swagger specification of an API.
payload	The actual information or message in transmitted data, as opposed to automatically generated metadata.
Deployment Descriptor	A deployment descriptor refers to a configuration file for an artifact that is deployed to some container.

The following steps are required for API development:

- Creation of Artefacts in T24- This section explains how data is retrieved from T24 system and provided to IRIS to expose to external requests. This in turn reduces the data loads being retrieved and simplifies the whole process. Also, it can be referred as filtering of T24 data by T24 being expose to the external request.
- Creation of Service Project- It contains the default directories and set of model data. It basically acts as a platform where developers can build and run the required APIs. It also contains a set of archetypes which gives developers options of creating desired APIs.
- Creation of Provider API- Provider APIs expose core banking capabilities as RESTful APIs. The key concept is that each Provider API is driven from an inventory that defines the contents of the API in core banking terms. The inventory is used to create the Swagger specification of the API. Together, the inventory file and the generated swagger specification are used to create the service implementation.
- Creation of Published API- Publisher APIs provide the following REST resources:

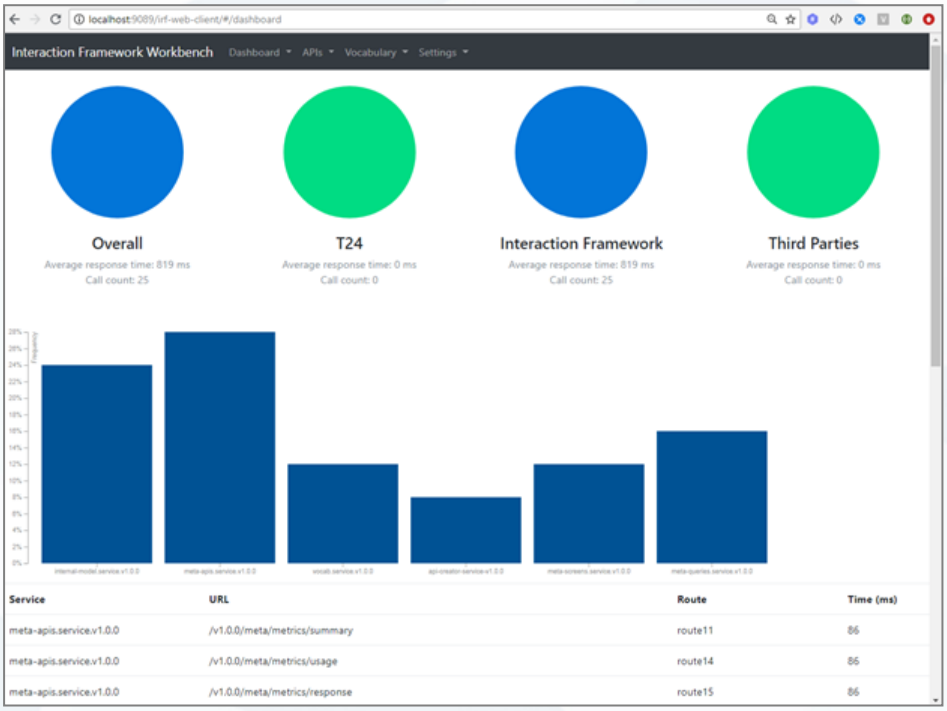
- Login
- Logout
- Add API, Update API, Remove API, Copy an API
- Validation roles

Note: When you access any API other than the login and logout APIs through an external REST client, first invoke the login API to ensure that user is authenticated. When the login API is invoked, the system stores the generated session cookie in a file, which we use in the next API invocations.

Workbench Installation

IRIS R18 Workbench

IRIS R18 Workbench is a Licensed Design-Time Tool that can be effectively used to create Provider APIs as well as Publisher APIs. A separate request is required to make to the Distribution to acquire the workbench which facilitates you to create the APIs on your own.



IRIS Workbench Packaging

For acquiring the package, a request should be made for IRIS R18 design time package which will follow the naming convention IRISR18_Design_vxx.xx.xx.zip where xx denotes the release version.

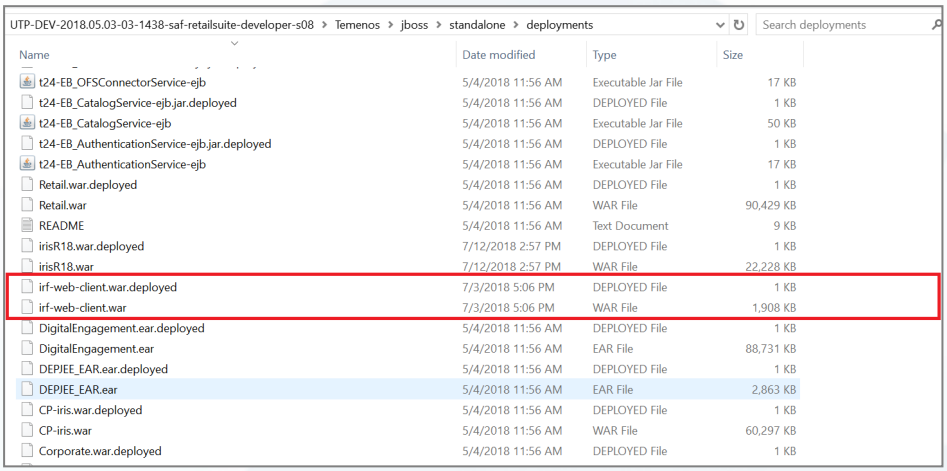
This package will mainly consist of the following components :

- irf-web-client.war - This is the Workbench war which needs to extracted from the zip package and can be deployed in any of the application servers(jboss, weblogic, websphere). It is required to connect to T24 via jetty server or default IRIS war.
- irf-test-web.war - This is an out of box pre-configured IRIS R18 war which will by default contain PSD2 APIs. This war can be deployed in jboss and can be used by the Workbench to connect to T24 during Design Time (steps mentioned below to configure the workbench for it).
- irf-repo - This is the IRIS R18 Framework artefacts that forms the repository and are needed while generating an IRIS war from the container project.

IRIS Workbench Configurations Steps

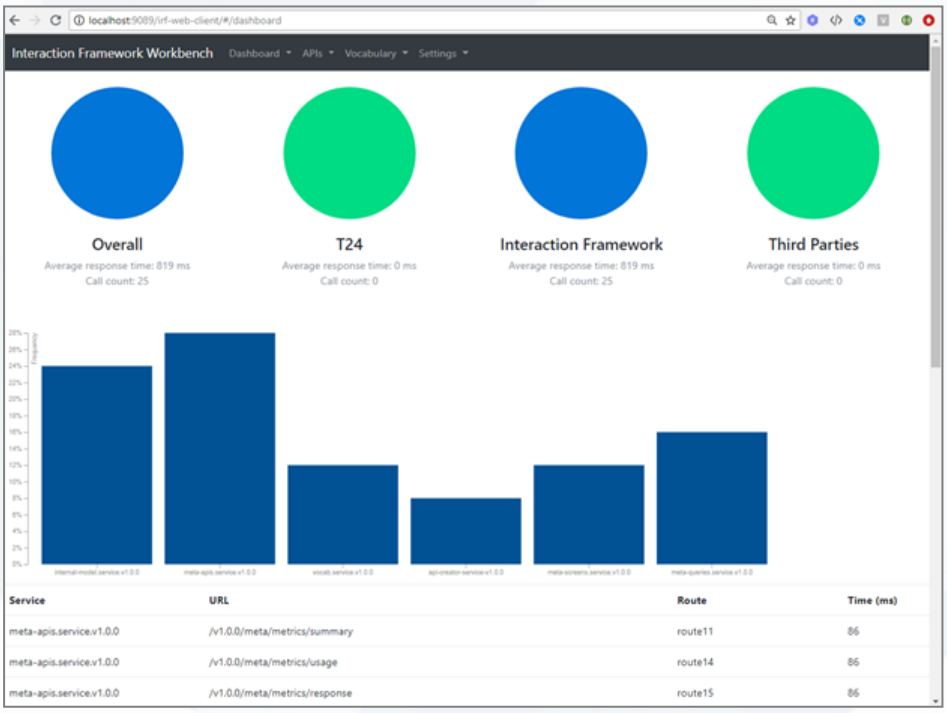
Step 1 : Extract the irf-web-client.war from the received Design time package IRISR18_Design_ vxx.xx.xx.zip.

Step 2 : Deploy the irf-web-client.war in deployment folder of Jboss Application Server.



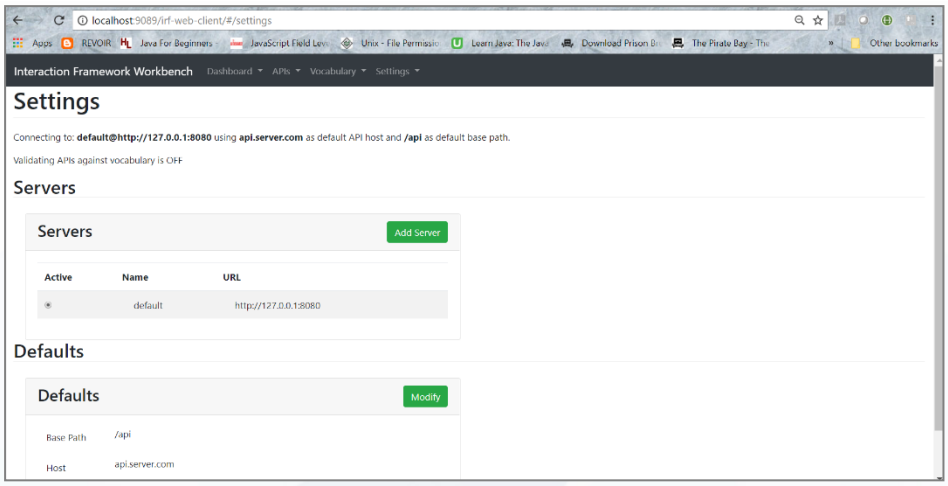
Step 3 : Verify that the irf-web-client.war is deployed successfully and Workbench is up and running.

- Open New Browser Window & type http://localhost:9089/irf-web-client/

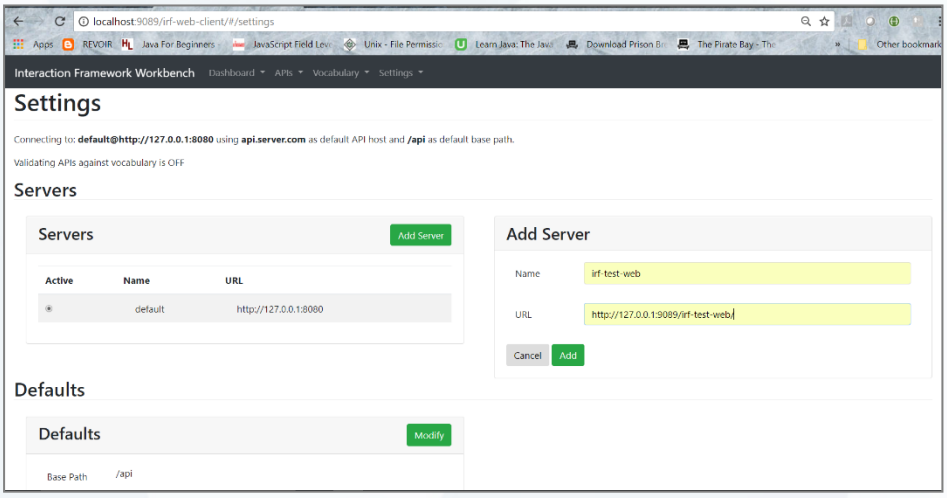


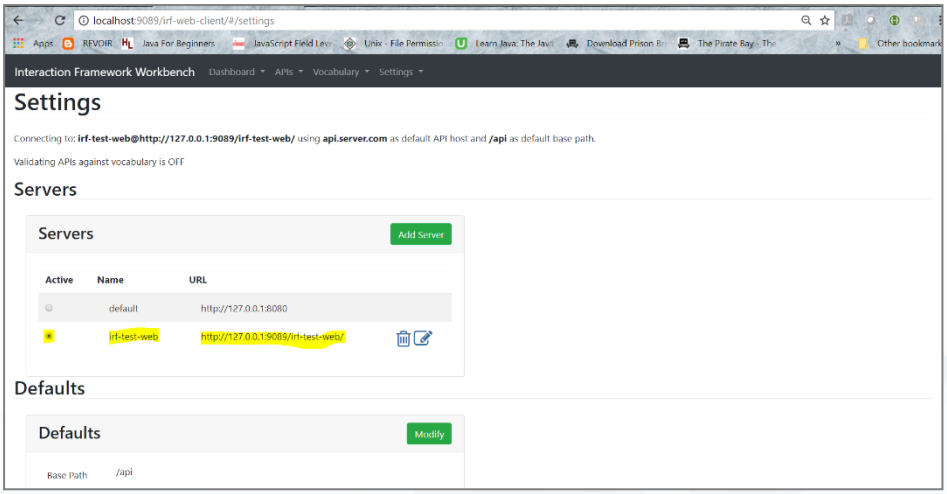
Step 4 : To start creating Provider APIs, you are required to connect to T24 to get a list of artefacts for which you intend to generate an API service. There are two ways to do this :

- First way is using jetty server started from the created container project. For that, ensure that the setting in the workbench has been properly configured and T24 is up and running. By Default , jetty runs on 8080 port.

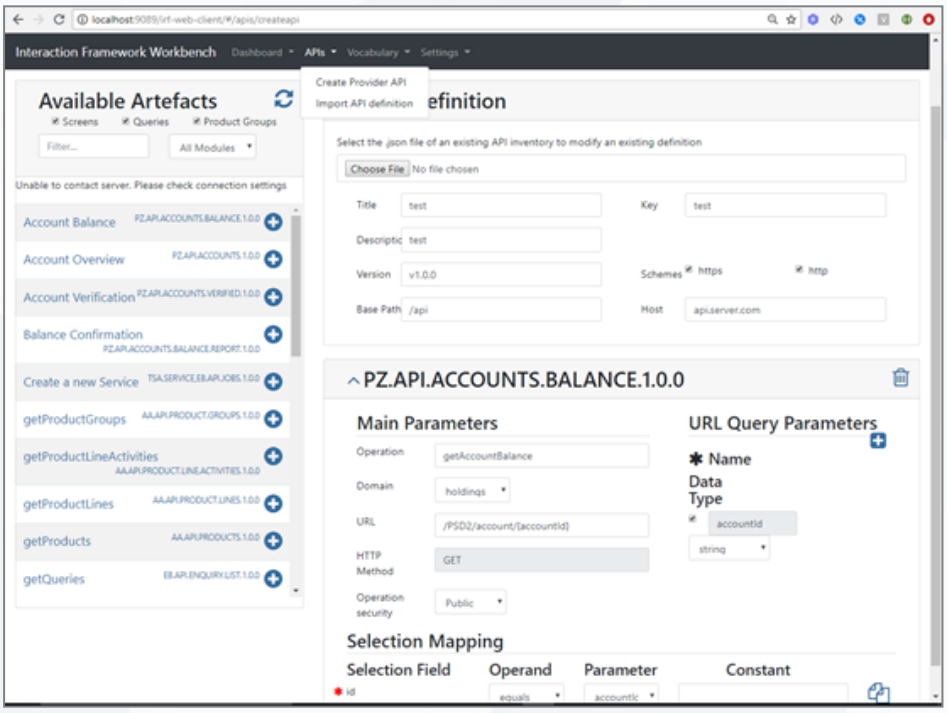


- Second way is to connect via irf-test-web.war provided as a part of the IRIS Design time package. For that you need to extract the irf-test-web.war from the package and deploy it in the jboss server. Once successfully deployed, you need to configure the WorkBench by adding a new server in the settings as shown below to connect to T24 via pre-configured irf-test-web.war





Step 5 : Once properly configured, you should hover to APIs → Create Provider API and the Workbench list you all the available artefacts in T24\. You are ready to go and create your own APIs.



Creation of Artefacts in T24

In order for T24 Version and Enquiry artefacts to be used in the Interaction Framework, certain rules must be followed. These rules provide a simple and effective governance framework that

- Clearly identifies that a given T24 artefact is used in an API
- Allows simple versioning control to be applied to the T24 artefact following semantic versioning (refer http://semver.org/)
- Provides data type meta data to ensure clean conversion of data from T24 to internet standard data types
- Provides meaningful operation names

The following rules must be followed:

ID Naming Convention

For Enquiry, the following ID naming convention must be followed:

```
XX.API.COLLECTIONID.SUBCOLLECTIONID.Major.Minor.Patch
```

Examples: PZ.API.ACCOUNTS.BALANCES.1.0.0, PZ.API.ACCOUNTS.TRANSACTIONS.1.0.0

For Version the following ID naming convention must be followed:

```
TABLE, XX.API.VERB.RESOURCEID.Major.Minor.Patch
```

Example: FUNDS.TRANSFER, AC.API.CREATE.TRANSFER.1.0.0

Field data typing

Each Enquiry column and Version field must have a data type defined.

For Enquiry, this is the FIELD.DISP.TYPE field,

For VERSION the ATTRIBS field is used.

Fields are considered to be alphanumeric unless they are set as **Date** or **Amount**. Amount and date fields must be unformatted, e.g. 20170123 and 1234.56

S.No	T24 Data Type	Swagger Schema Data Type
1	ALPHANUMERIC	string
2	AMOUNT	number
3	DATE	Date
4	TIMESTAMP	Date-time
5	BOOLEAN	Boolean

Description

Each Enquiry and Version must have a description. For Enquiry, this is the DESCRIPT field, for Version the DESCRIPTION field is used.

This is used to describe the operation supported and should follow verb/resource/subresource, e.g. getAccountTransactions, getAccountBalances, createAccountTransfer

Restrictions on Enquiry

Header fields should not be used in API enquiries.

- Turn SEL.LABEL into API style labels
- remove HEADER
- Turn FIELD.LABEL into API style labels
- Use FIELD.DISPLAY.TYPE for data type (Alphanumeric / amount, date)
- remove any static labels - where OPERATION is a literal and COLUMN is set
- Set ATTRIBS to ''
- Remove drill down information (ENQUIRY.NAME,SEL.CRIT,LABEL.FIELD,NXT.DESC)
- Set DESCRIPT to be the operation, e.g. getAccountBalances

Naming should be in standard vocabulary

The standard Temenos vocabulary exists to bring consistent naming across all API initiatives. Field / main resource / sub resource names should exist in the standard vocabulary.

Click here to know **how the T24 enquiry definition affects the JSON response**.

> Note: If there is an existing Service Project, you can skip Create Service Project step and navigate to Provider API step.

Creation of Service Container

In order to run any API projects, you should have a service container. A service container hosts one or more service projects. It is also a runtime container that hosts the camel run time and configuration required to connect to downstream systems, i.e. T24

You can create the service container using the supplied Maven Archetype, either from your IDE or from the command line.

Using the command line

mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-container-archetype

-DarchetypeVersion=1.0

Supply values for various arguments, either interactively or as parameters from the command line:

Define value for property 'groupId': com.temenos.training.irf

Define value for property 'artifactId': training-container

Define value for property 'version' 1.0-SNAPSHOT: :

Define value for property 'package' com.temenos.training.irf::

Confirm properties configuration:

groupId: com.temenos.training.irf

artifactId: training-container

version: 1.0.0-SNAPSHOT

package: com.temenos.training.irf

As a single command:

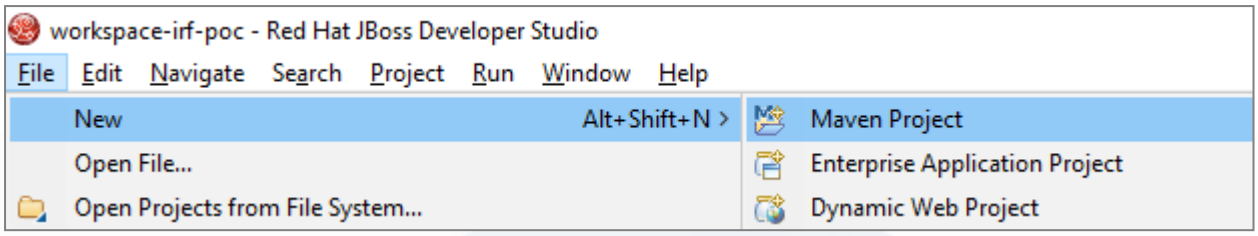
mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-container-archetype

-DarchetypeVersion=1.0 -DartifactId=training-container

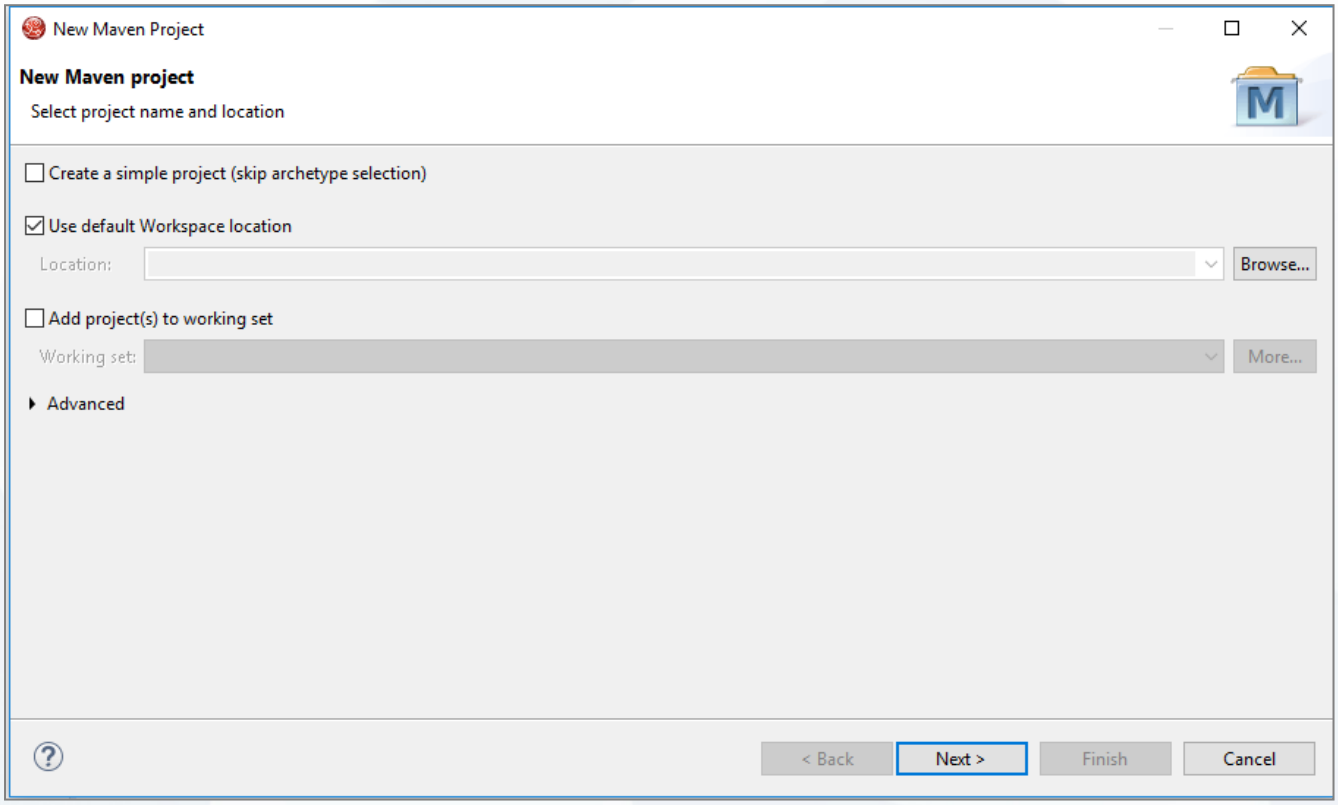
-DgroupId=com.temenos.irf.training -Dpackage=com.temenos.irf.training -Dversion=SNAPSHOT-0.0.1

Using Eclipse

From the File menu, select File > New > Maven Project

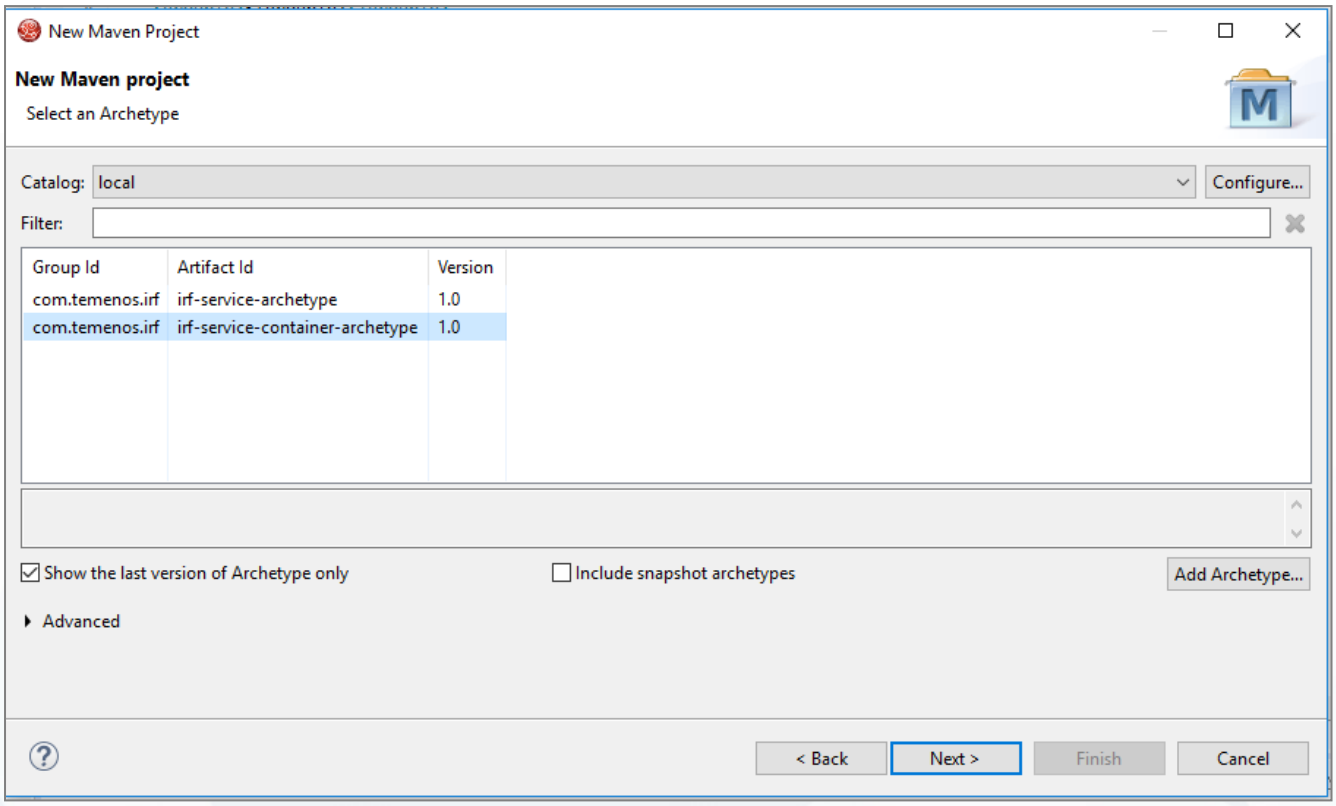


Click the Next button.



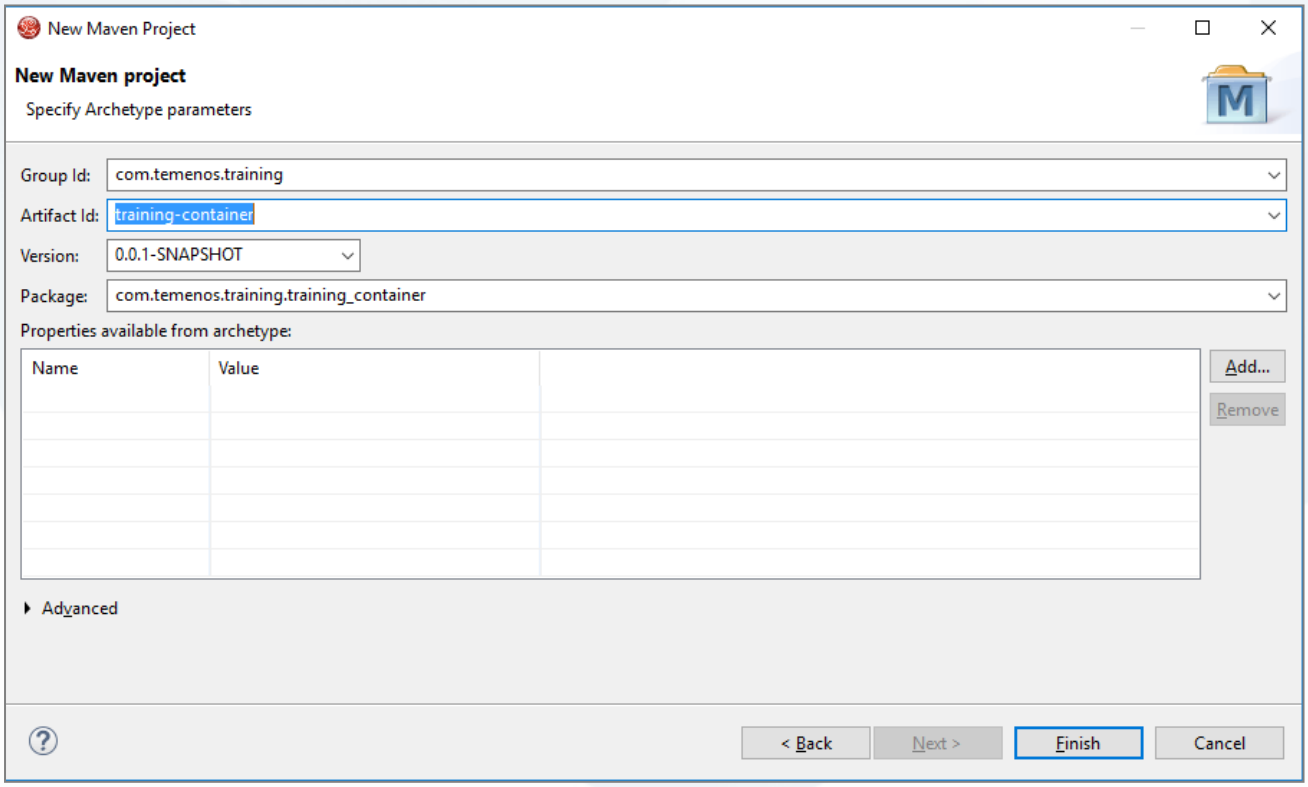
Select Archetype

Select the **irf-service-container-archetype** option from Artifact Id and click the **Next** button.



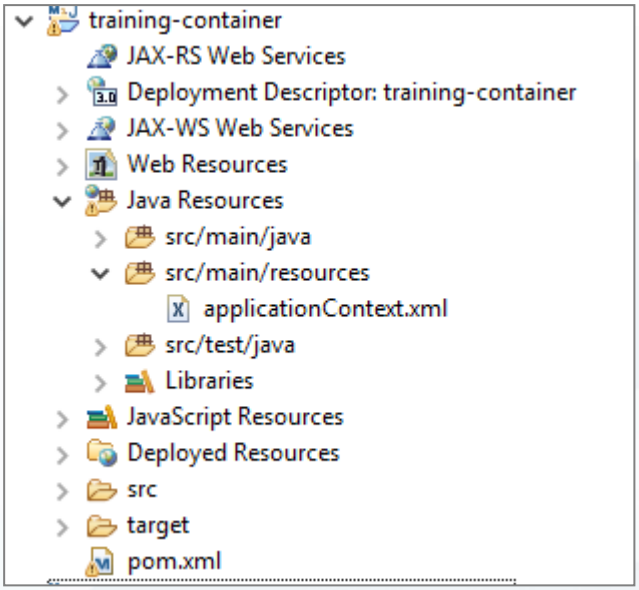
Provide Project Details

Fill in the **Group Id**, **Artifact Id** and **Version** fields, and click the **Finish** button.



Project Structure

You should see a project created with the following structure:



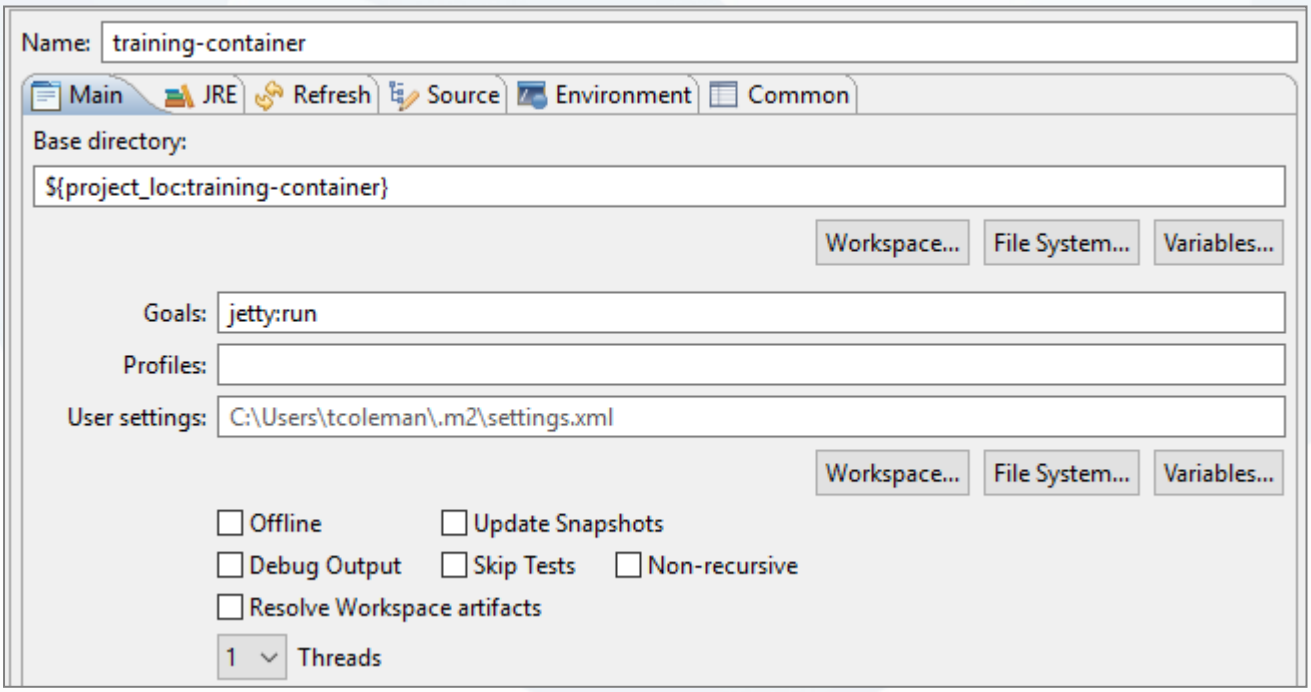
Start the server

You can now start the server, either from the command line or from your IDE using mvn jetty:run

mvn jetty:run

In Eclipse, right click the project and select Run As > Maven Build

In the dialog box, set the Goals to jetty:run



Creation of Service Project

Create a service project using the irf-service-archetype.

Using the command line

mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-archetype

-DarchetypeVersion=1.0 -DarchetypeCatalog=local

Supply values for various arguments, either interactively or as parameters from the command line:

Define value for property 'groupId': com.temenos.training.irf

Define value for property 'artifactId': training-api

Define value for property 'version' 1.0-SNAPSHOT: :

Define value for property 'package' com.temenos.training.irf::

Confirm properties configuration:

groupId: com.temenos.training.irf

artifactId: training-api

version: 1.0-SNAPSHOT

package: com.temenos.training.irf

As a single command:

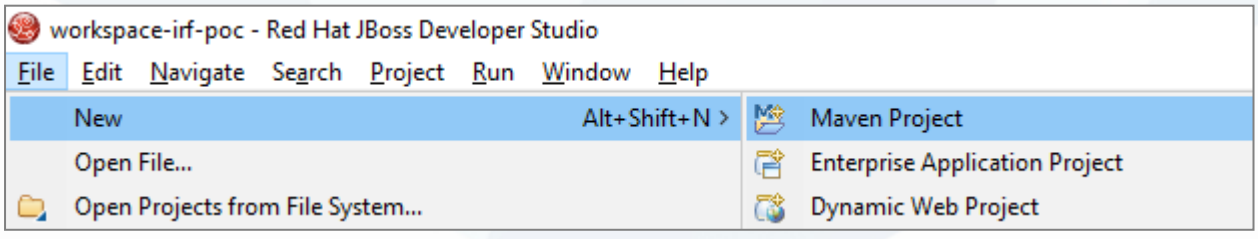
mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-archetype

-DarchetypeVersion=1.0 -DartifactId=training-api

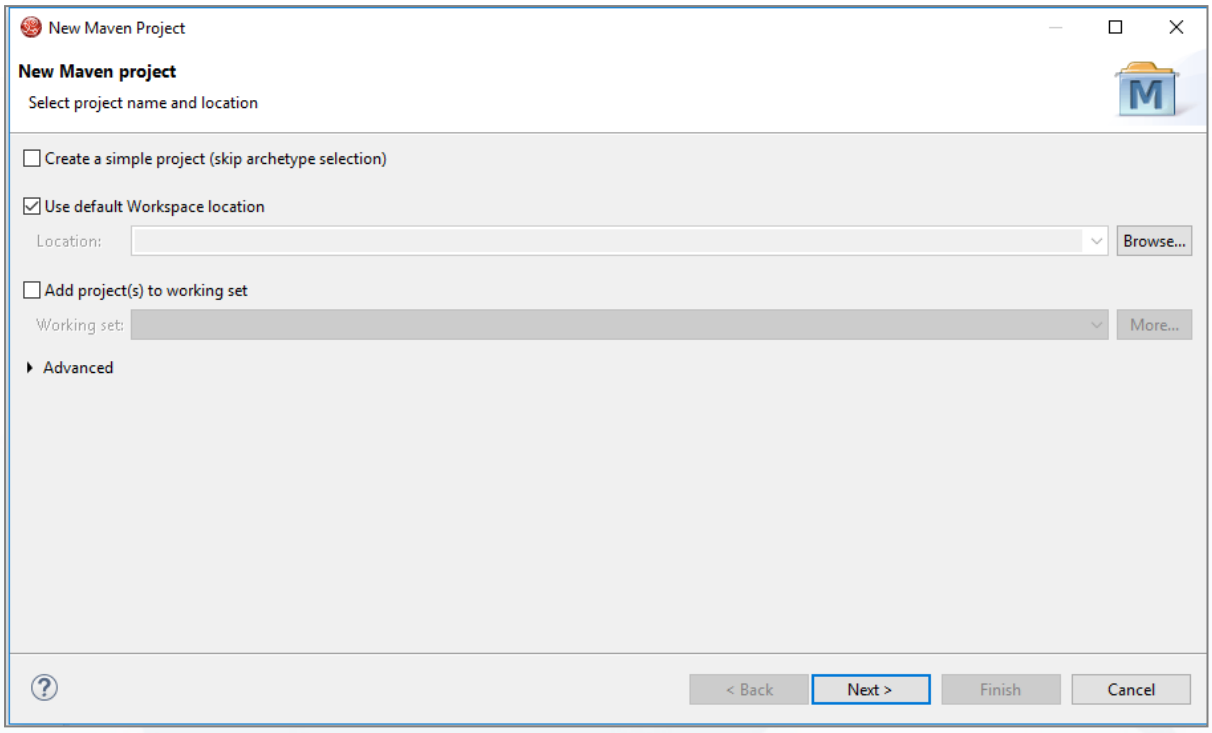
-DgroupId=com.temenos.irf.training -Dpackage=com.temenos.irf.training -Dversion=SNAPSHOT-0.0.1

Using Eclipse

From the File menu, select **File** > **New** > **Maven Project**

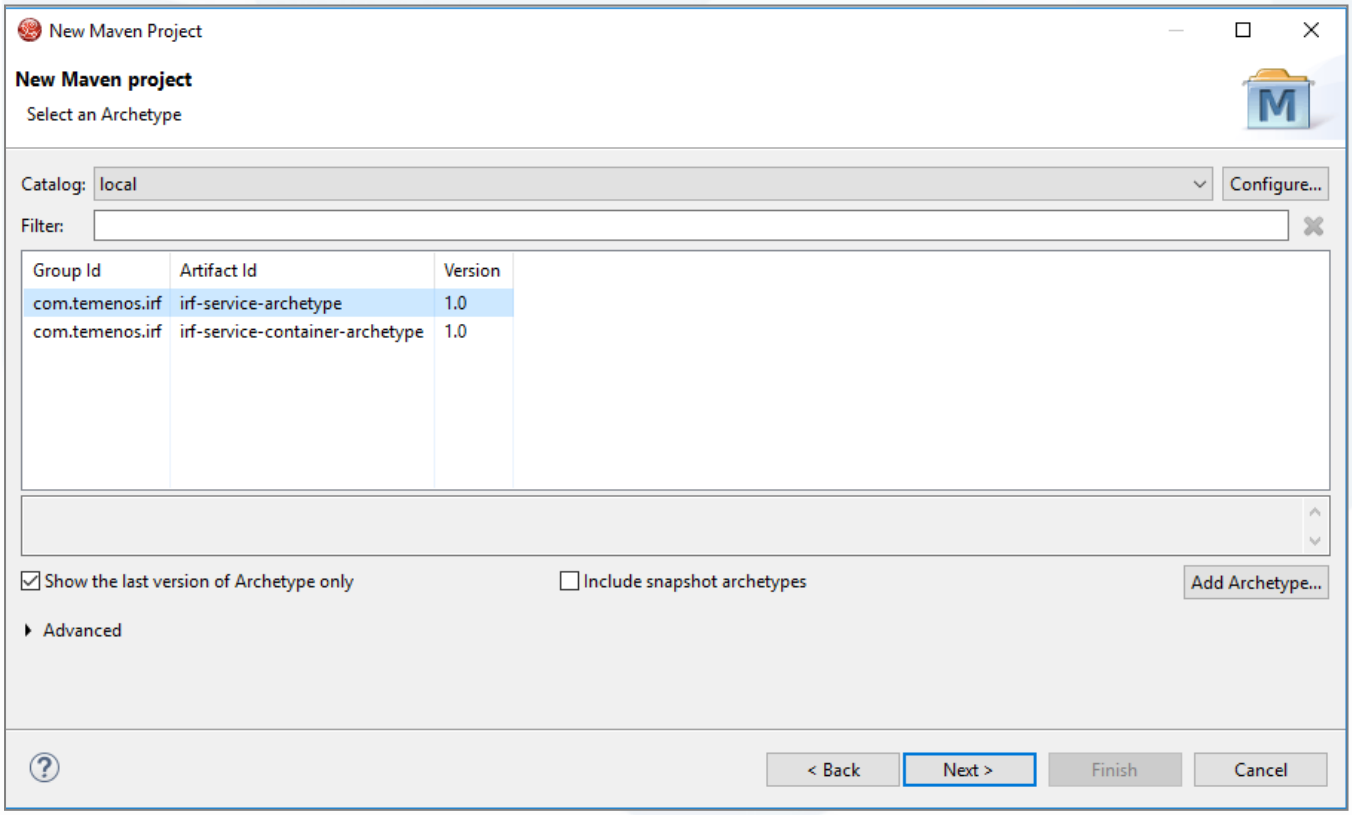


Click the **Next** button.



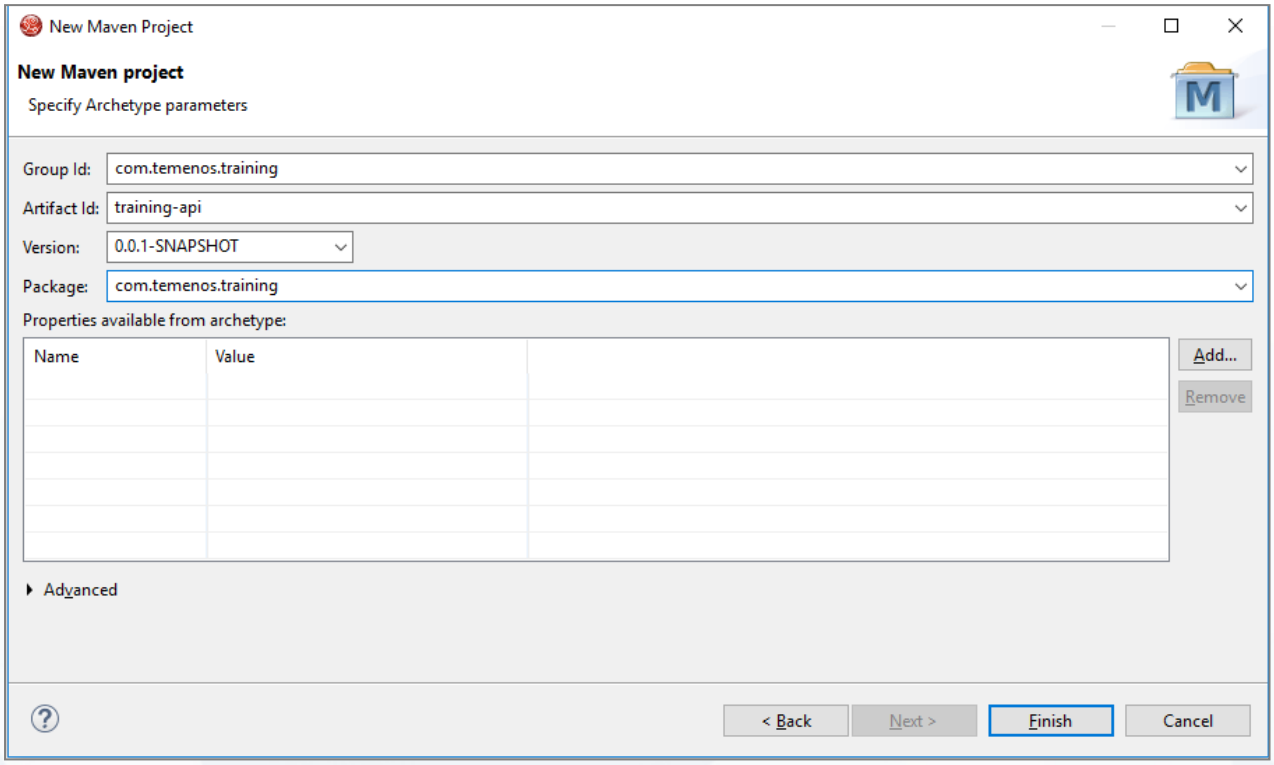
Select archetype

Select the **irf-service-archetype** option from Artifact Id column and click the **Next** button.



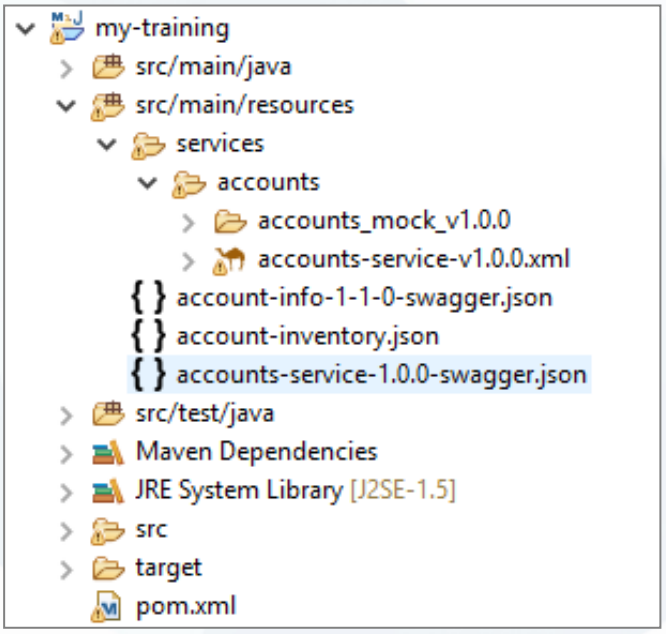
Provide Project details

Fill in the **Group Id**, **Artifact Id** and **Version** fields, and click the **Finish** button.



Project Structure

You should see a project created with the following structure:



Using Workbench to Define Provider API

The easiest mechanism to create a Provider API is to use the Interaction Framework Workbench. Select "APIs > Create Provider API":

Step 1 - Start Service Container

Make sure that, Service Container is up and running.

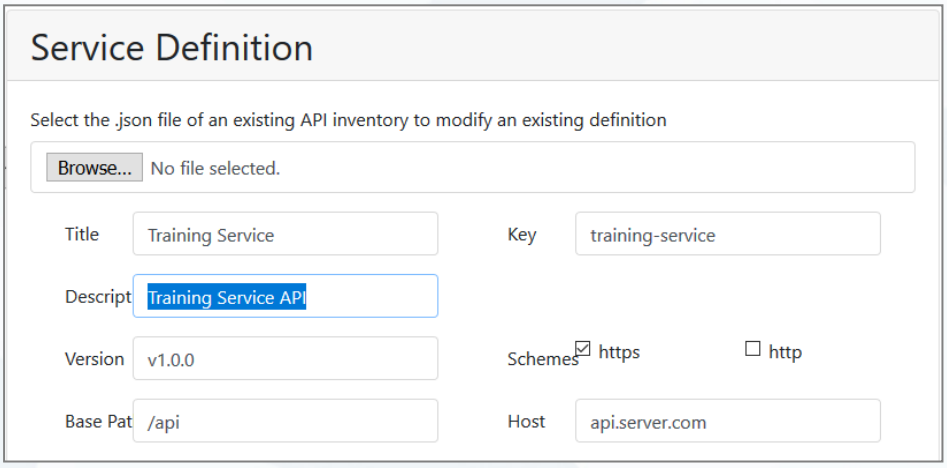
http://localhost:8080/api/v1.0.0/meta/apis

If it is not running, start the server using the following command

```
jetty:run   from maven command prompt
```

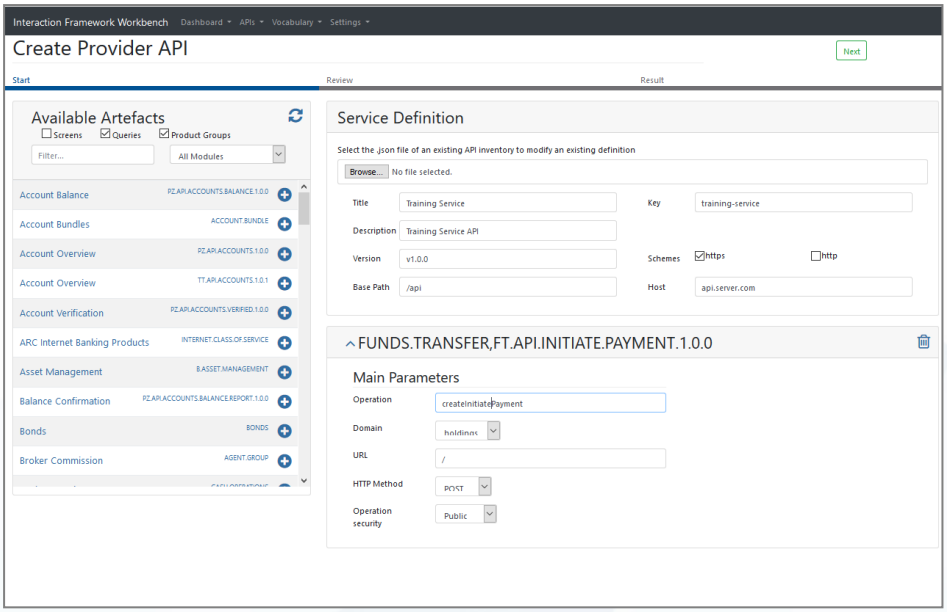
Step 2 - Service Definition

Provide the Title, service Key, Description , etc in the Service Definition section



Step 3 - Select Artefacts to include

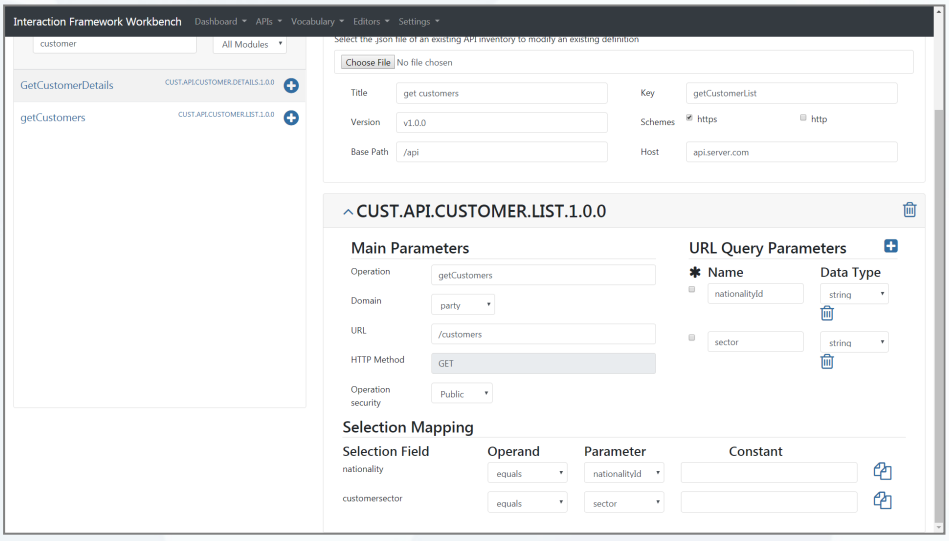
Artefacts may be filtered to aid in discovery. Clicking on an artefact will add an endpoint to the service, and create a URL based on the artefact definitions. Review the URL and Operation to ensure that this matches the API design. An end point may be removed by clicking on the trash icon in the top right of the endpoint definition.



Step 4 - Map URL parameters to Query Selection

Where a T24 VERSION is selected, only the method is required to be set. The payload will be configured in Swagger directly from the T24 Model. POST should be used to create new resources, PUT to amend and existing resource, GET for viewing a resources and DELETE for removing a resource.

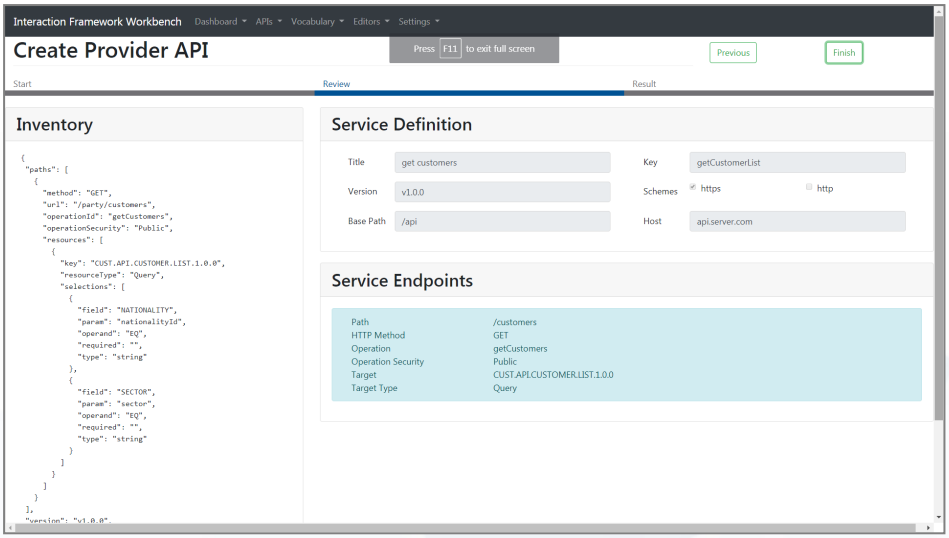
Where a T24 ENQUIRY is selected, the URL parameters need to be mapped to the ENQUIRY selection fields.



A selection field can be mapped from mulitple URL query or path parameters. In the above example, fromDate and toDate are both mapped to the valueDate selection field.

Click Next in the top right fo the screen to review the provider API

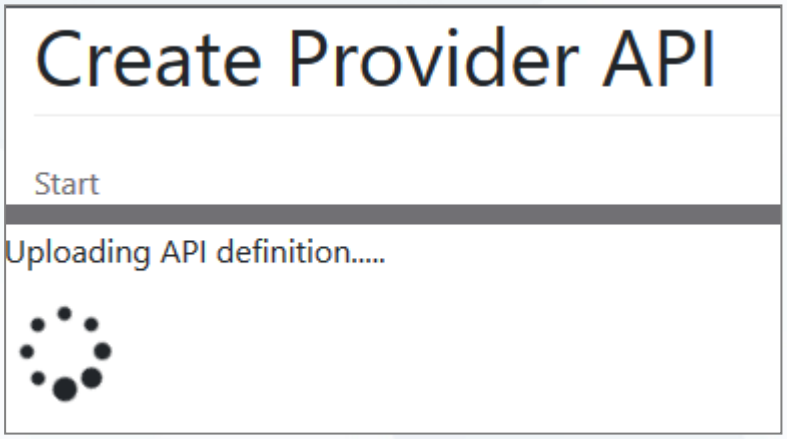
Step 5 - Review



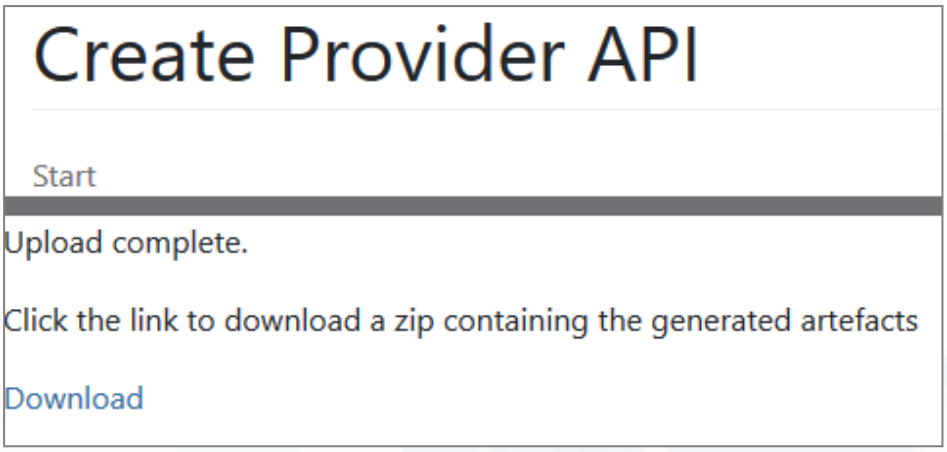
The inventory file can be copied directly from the Inventory pane and pasted into an IDE. Or if preferred click Finish to create a zip file file containing the swager spec, Camel routes, mappings and mock responses.

Step 6 (optional) - Submit

Click Finish the top right fo the screen to submit the provider API



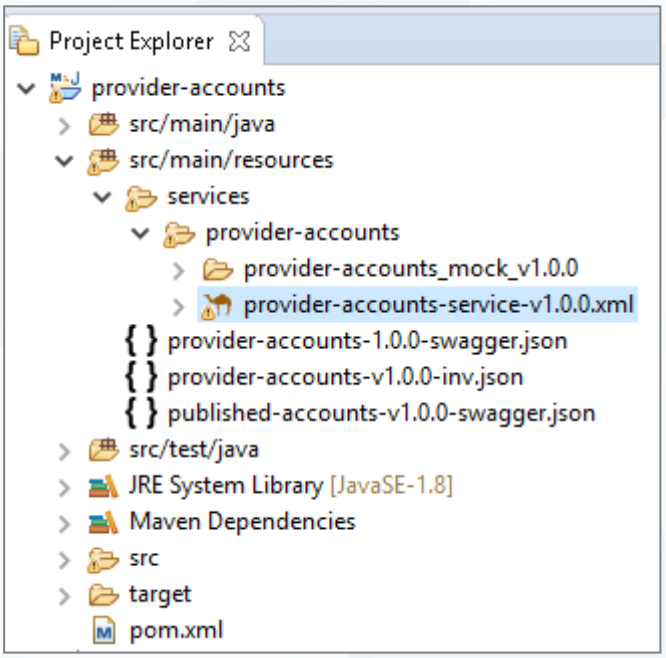
The result provides a link to download a zip file containing the swager spec, Camel routes, mappings and mock responses.



Step 7 Importing generated Artifacts

Extract the content of downloaded zip into <service-provider> project/ resources folder.

Project Structure

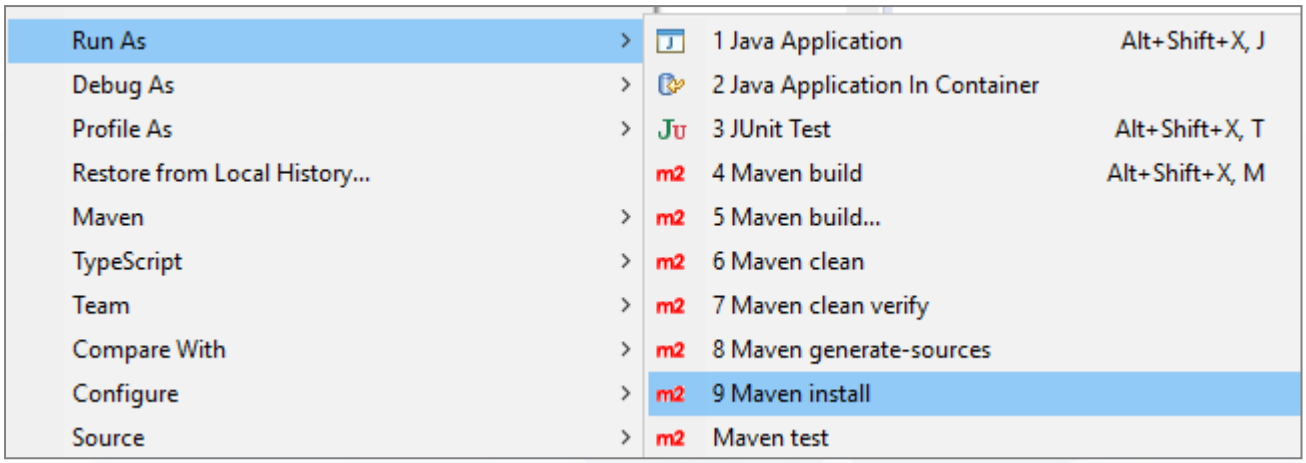


Step 8 - Build The Provider API Project

Run the project as Maven > Install to build the project

Build the Service Project

Right click the project and choose Run As > Maven install to build the project.


Run As
Debug As
Profile As
Restore from Local History...
Maven
TypeScript
Team
Compare With
Configure
Source
1 Java Application Alt+Shift+X, J
2 Java Application In Container
3 JUnit Test Alt+Shift+X, T
4 Maven build Alt+Shift+X, M
5 Maven build...
6 Maven clean
7 Maven clean verify
8 Maven generate-sources
9 Maven install
Maven test

Creation of Provider API

Provider APIs expose core-banking capabilities as RESTful APIs. The key concept is that each Provider API is driven from an **inventory** that defines the contents of the API in core banking terms. The inventory is used to create the Swagger specification of the API. Together, the inventory file and the generated swagger specification are used to create the service implementation.

Creating an Inventory using Interaction Framework Workbench

The easiest mechanism to create a Provider API is to use the Interaction Framework Workbench. Select "**APIs > Create Provider API**":

Step 1 - Start Service Container

Make sure that, Service Container is up and running.

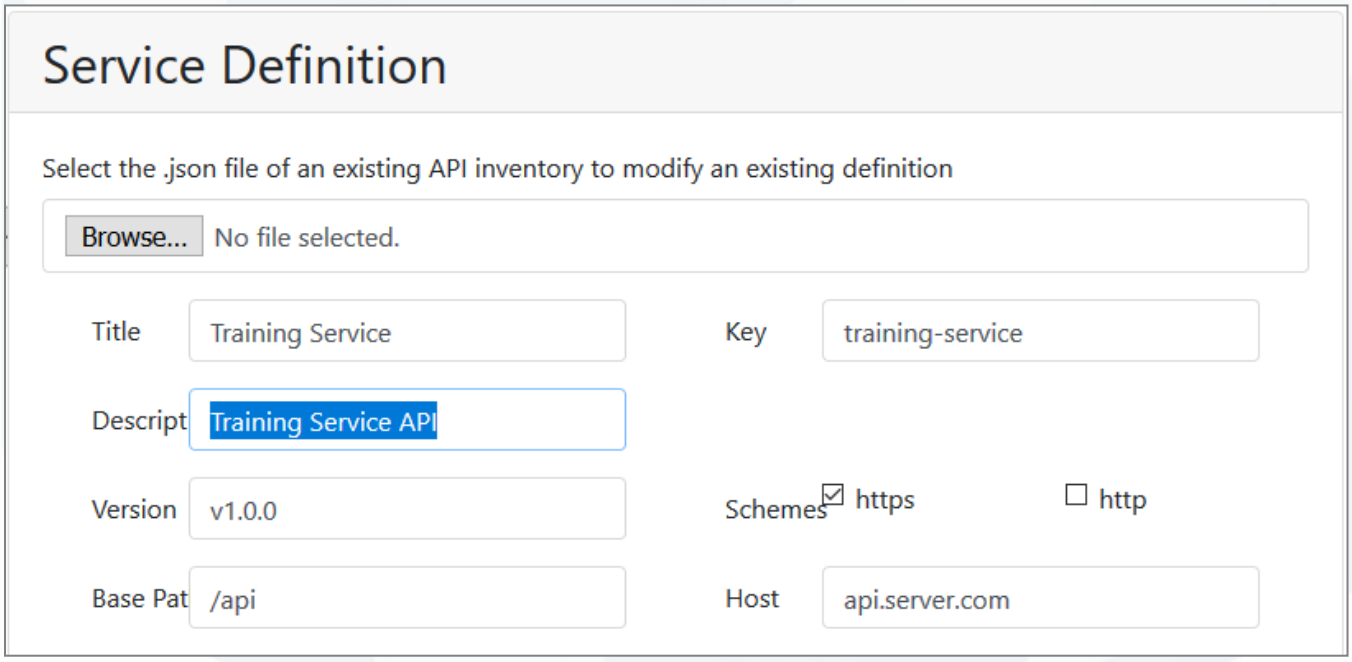
http://localhost:8080/api/v1.0.0/meta/apis

If it is not running, start the server using the following command

```
jetty:run from maven command prompt
```

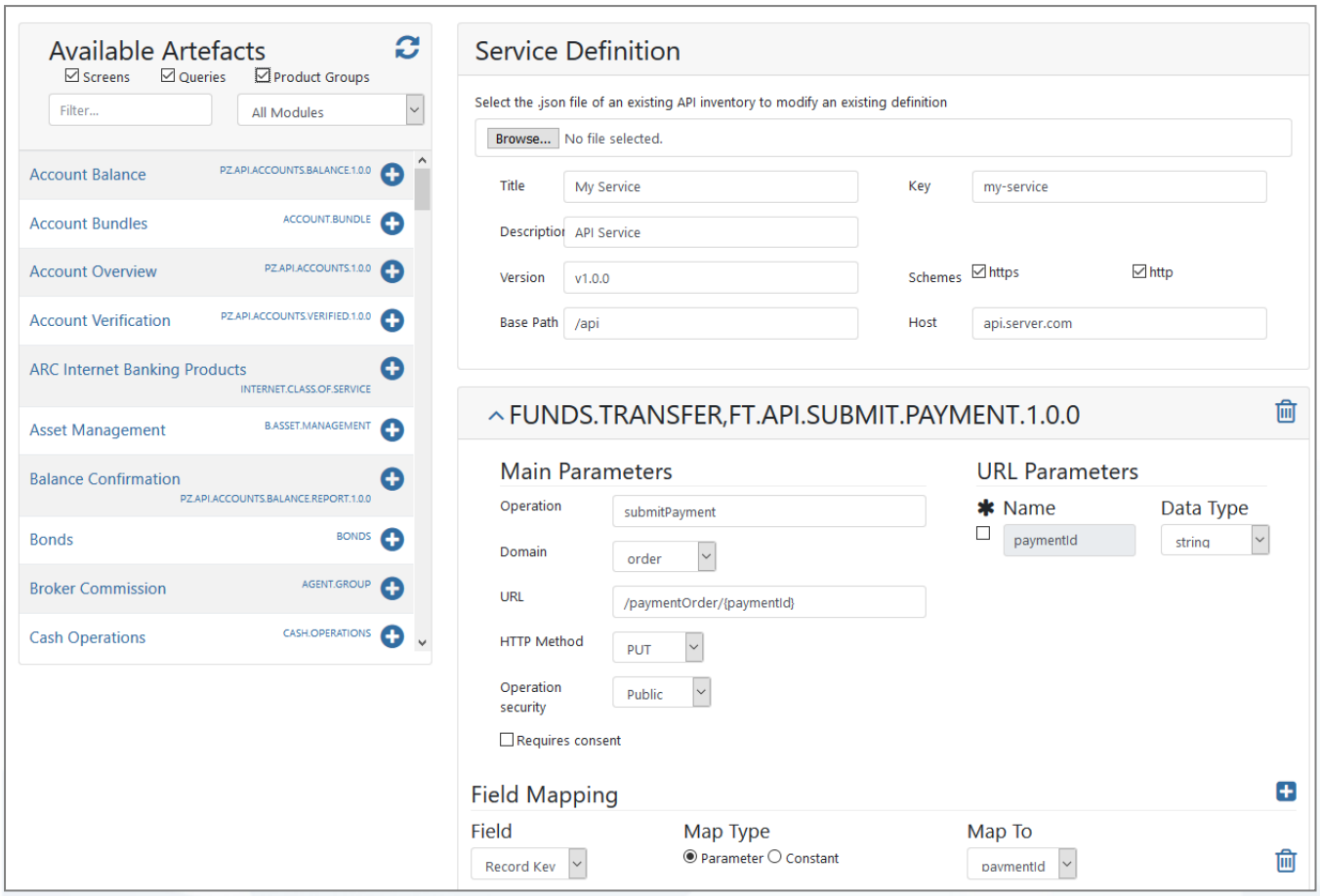
Step 2 - Service Definition

Provide the Title, service Key, Description, etc in the Service Definition section.



Step 3 - Select Artefacts to include

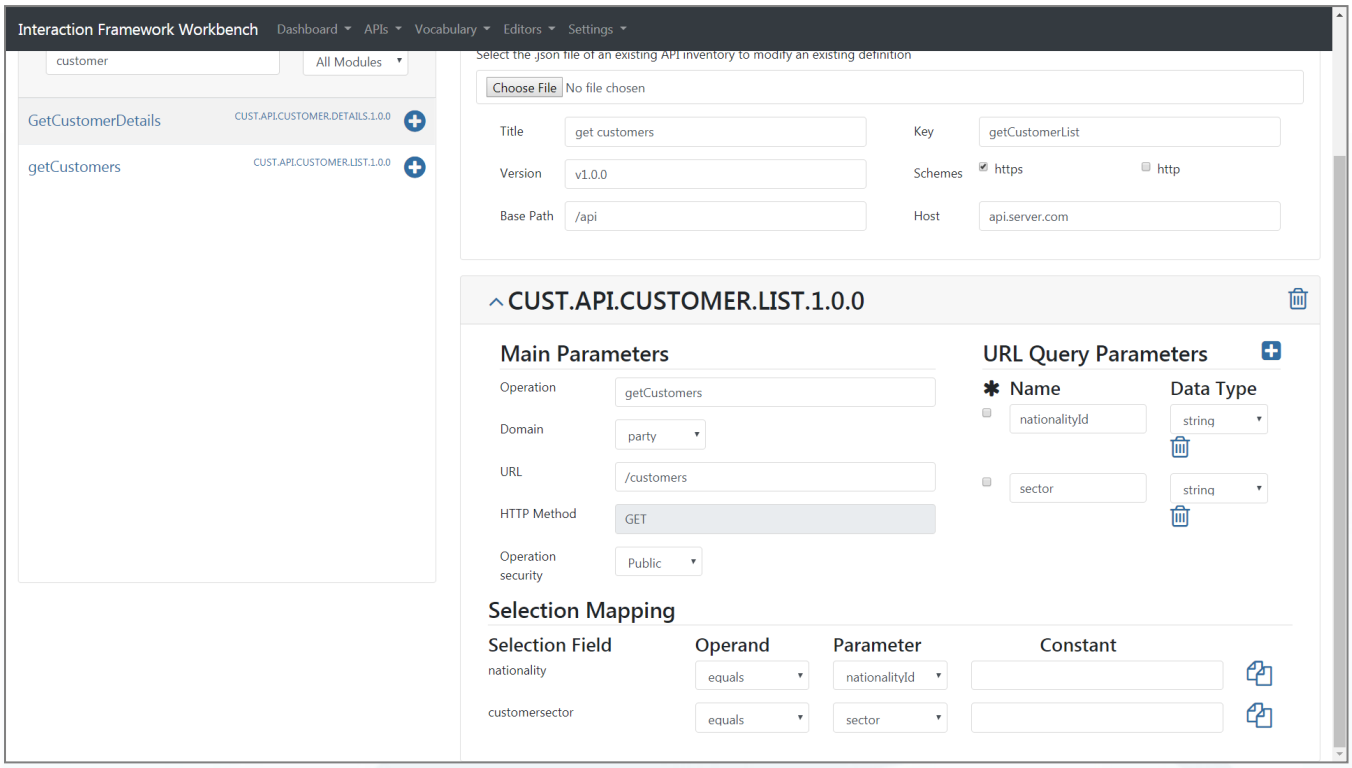
Artefacts may be filtered to aid in discovery. Clicking on an artefact will add an endpoint to the service, and create a URL based on the artefact definitions. Review the URL and Operation to ensure that this matches the API design. An end point may be removed by clicking on the trash icon in the top right of the endpoint definition.

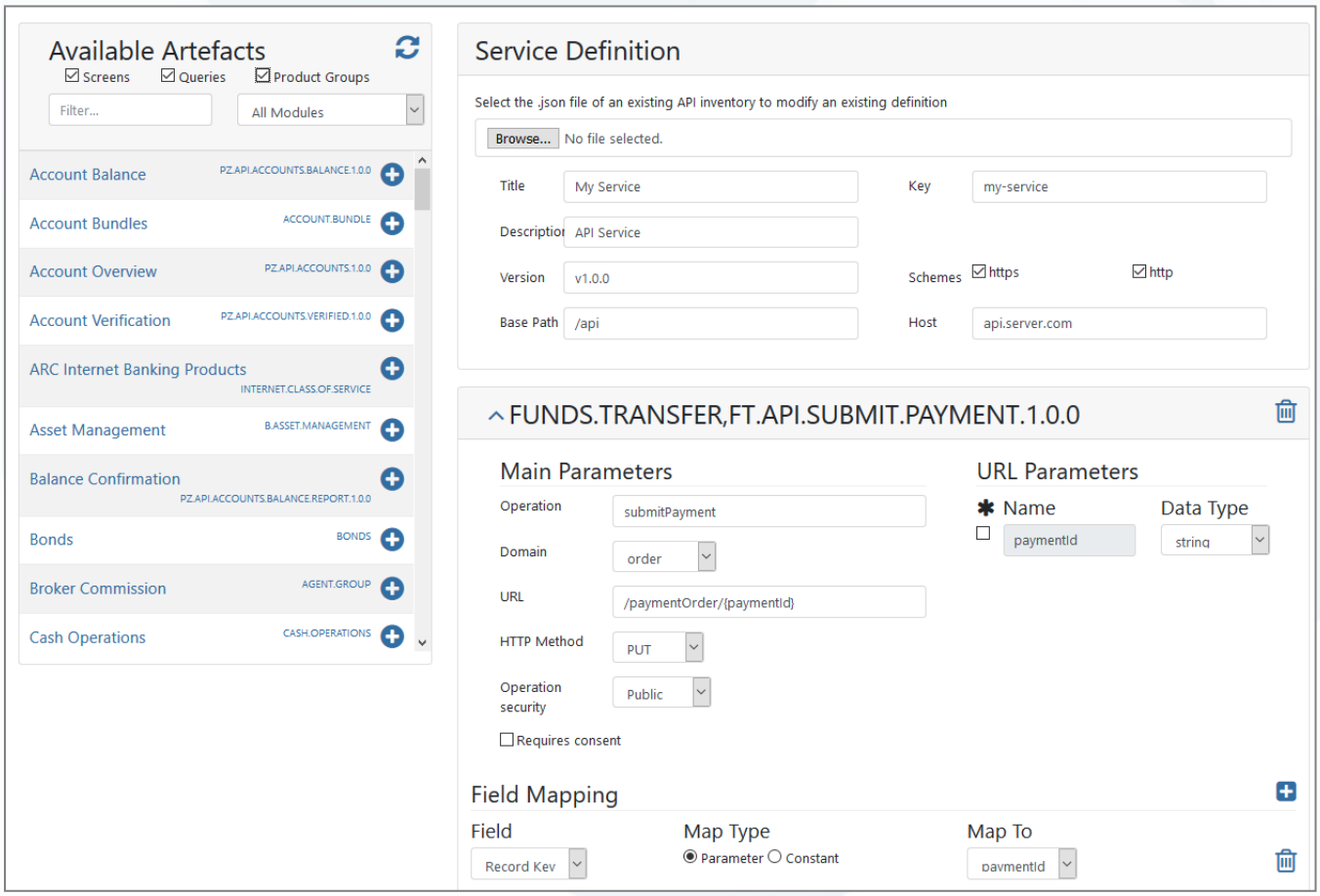


Step 4 - Map URL parameters to Query Selection/ Version Fields

Where a T24 VERSION is selected, only the method is required to be set. The payload will be configured in Swagger directly from the T24 Model. POST should be used to create new resources, PUT to amend an existing resource, GET for viewing a resource and DELETE for removing a resource. For extra URL mappings, the **Field Mapping** settings are to be used. To map a record key, **Field** should be set as **Record Key** and should be mapped to a parameter or constant. Other parameters in the URL can be mapped using this option.

Where a T24 ENQUIRY is selected, the URL parameters need to be mapped to the ENQUIRY selection fields.

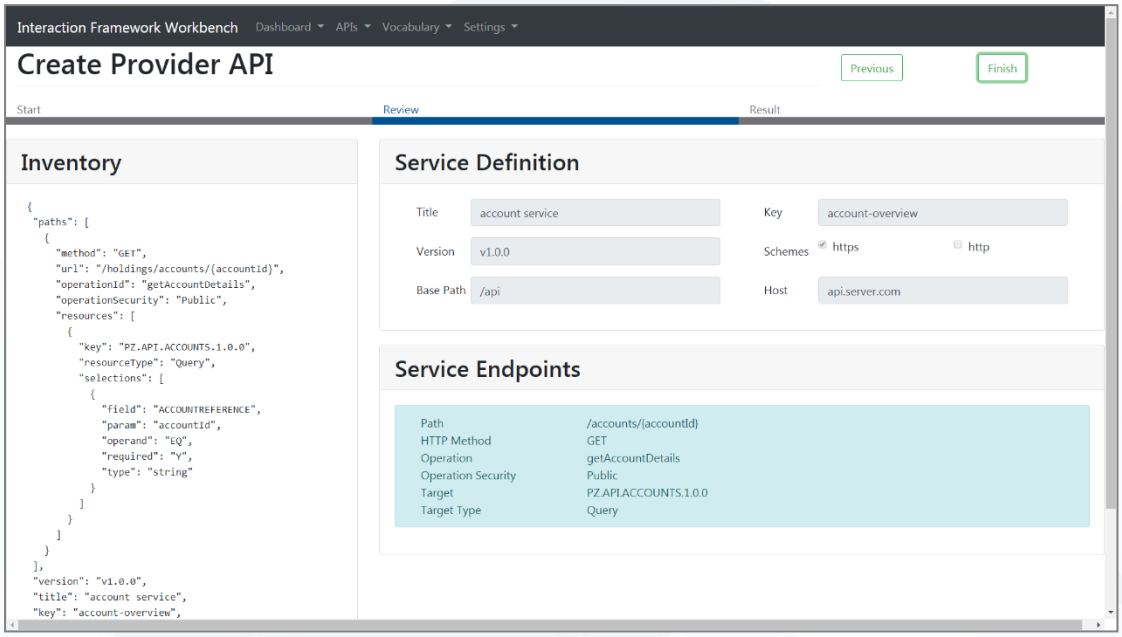

Interaction Framework Workbench
Dashboard
APIs
Vocabulary
Editors
Settings
customer
All Modules
GetCustomerDetails
CUST.API.CUSTOMER.DETAILS.1.0.0
getCustomers
CUST.API.CUSTOMER.LIST.1.0.0
Select the .json file of an existing API inventory to modify an existing definition
Choose File
No file chosen
Title
get customers
Key
getCustomerList
Version
v1.0.0
Schemes
https
http
Base Path
/api
Host
api.server.com
CUST.API.CUSTOMER.LIST.1.0.0
Main Parameters
Operation
getCustomers
Domain
party
URL
/customers
HTTP Method
GET
Operation security
Public
URL Query Parameters
Name
Data Type
nationalityId
string
sector
string
Selection Mapping
Selection Field
Operand
Parameter
Constant
nationality
equals
nationalityId
customersector
equals
sector


Available Artefacts
Screens
Queries
Product Groups
Filter...
All Modules
Account Balance
PZ.API.ACCOUNTS.BALANCE.1.0.0
Account Bundles
ACCOUNT.BUNDLE
Account Overview
PZ.API.ACCOUNTS.1.0.0
Account Verification
PZ.API.ACCOUNTS.VERIFIED.1.0.0
ARC Internet Banking Products
INTERNET.CLASS.OF.SERVICE
Asset Management
B.ASSET.MANAGEMENT
Balance Confirmation
PZ.API.ACCOUNTS.BALANCE.REPORT.1.0.0
Bonds
BONDS
Broker Commission
AGENT.GROUP
Cash Operations
CASH.OPERATIONS
Service Definition
Select the .json file of an existing API inventory to modify an existing definition
Browse...
No file selected.
Title
My Service
Key
my-service
Description
API Service
Version
v1.0.0
Schemes
https
http
Base Path
/api
Host
api.server.com
FUNDS.TRANSFER,FT.API.SUBMIT.PAYMENT.1.0.0
Main Parameters
Operation
submitPayment
Domain
order
URL
/paymentOrder/{paymentId}
HTTP Method
PUT
Operation security
Public
Requires consent
URL Parameters
Name
Data Type
paymentId
string
Field Mapping
Field
Map Type
Map To
Record Key
Parameter
Constant
paymentId

A selection field can be mapped from mulitple URL query or path parameters. In the above example, fromDate and toDate are both mapped to the valueDate selection field.

Click Next in the top right of the screen to review the Provider API.

Step 5 – Review



> Note: The inventory file can be copied directly from the Inventory pane and pasted into an IDE or if preferred click Finish to create a zip file containing the swagger spec, Camel routes, mappings and mock responses.

Click here to know how to create a zip file containing the swagger spec, Camel routes, mappings and mock responses.

Create Service Project

Create a service project using the irf-service-archetype.

Using the command line

mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-archetype

-DarchetypeVersion=1.0 -DarchetypeCatalog=local

Supply values for various arguments, either interactively or as parameters from the command line:

Define value for property 'groupId': com.temenos.training.irf

Define value for property 'artifactId': training-api

Define value for property 'version' 1.0-SNAPSHOT: :

Define value for property 'package' com.temenos.training.irf::

Confirm properties configuration:

groupId: com.temenos.training.irf

artifactId: training-api

version: 1.0-SNAPSHOT

package: com.temenos.training.irf

As a single command:

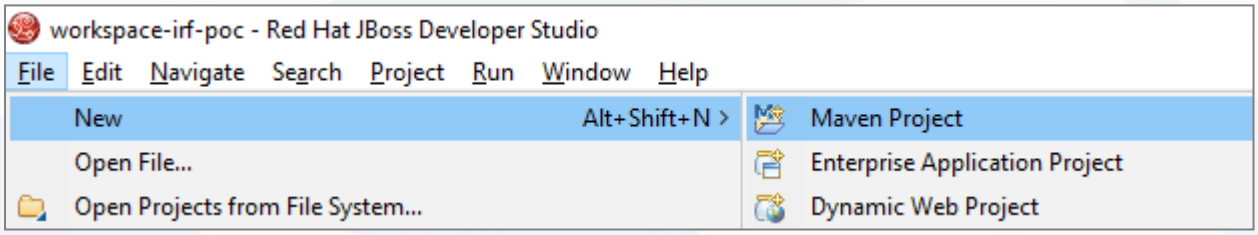
mvn archetype:generate -DarchetypeGroupId=com.temenos.irf -DarchetypeArtifactId=irf-service-archetype

-DarchetypeVersion=1.0 -DartifactId=training-api

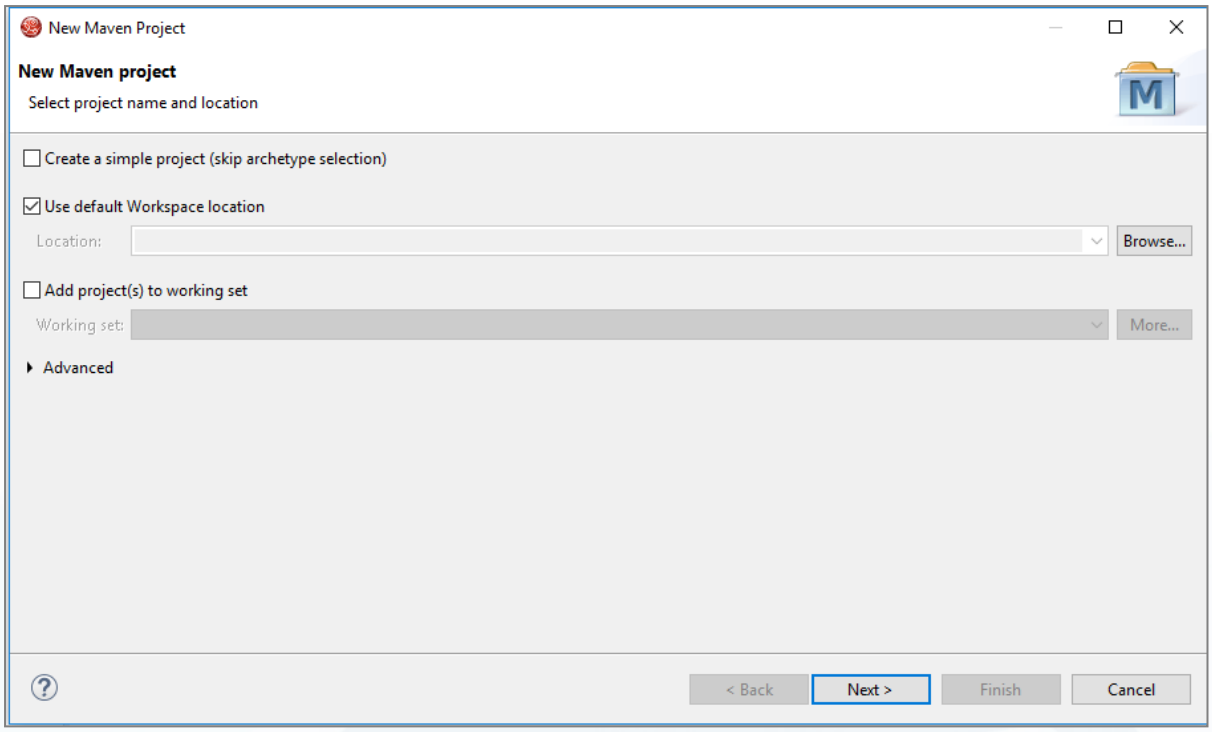
-DgroupId=com.temenos.irf.training -Dpackage=com.temenos.irf.training -Dversion=SNAPSHOT-0.0.1

Using Eclipse

From the File menu, select **File** > **New** > **Maven Project**

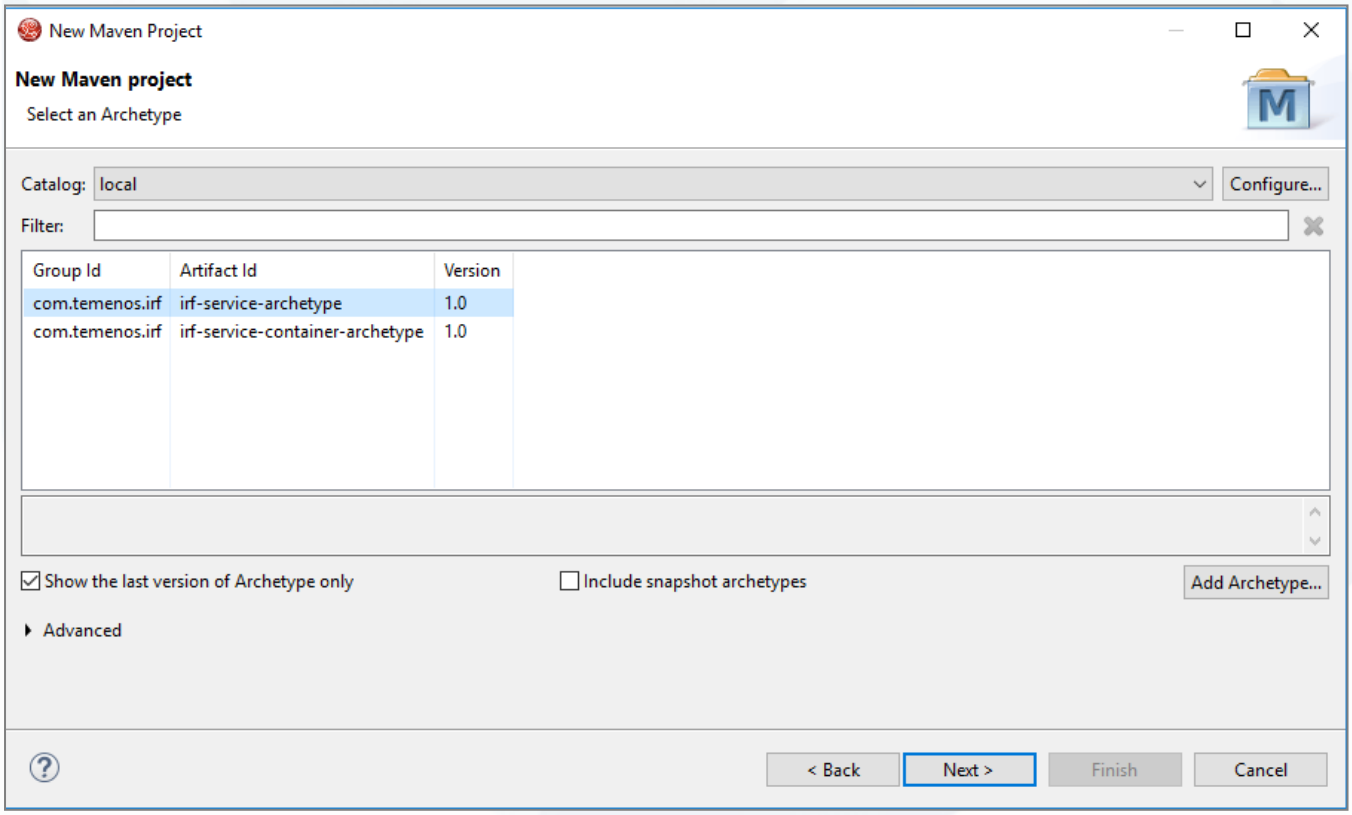


Click the **Next** button.



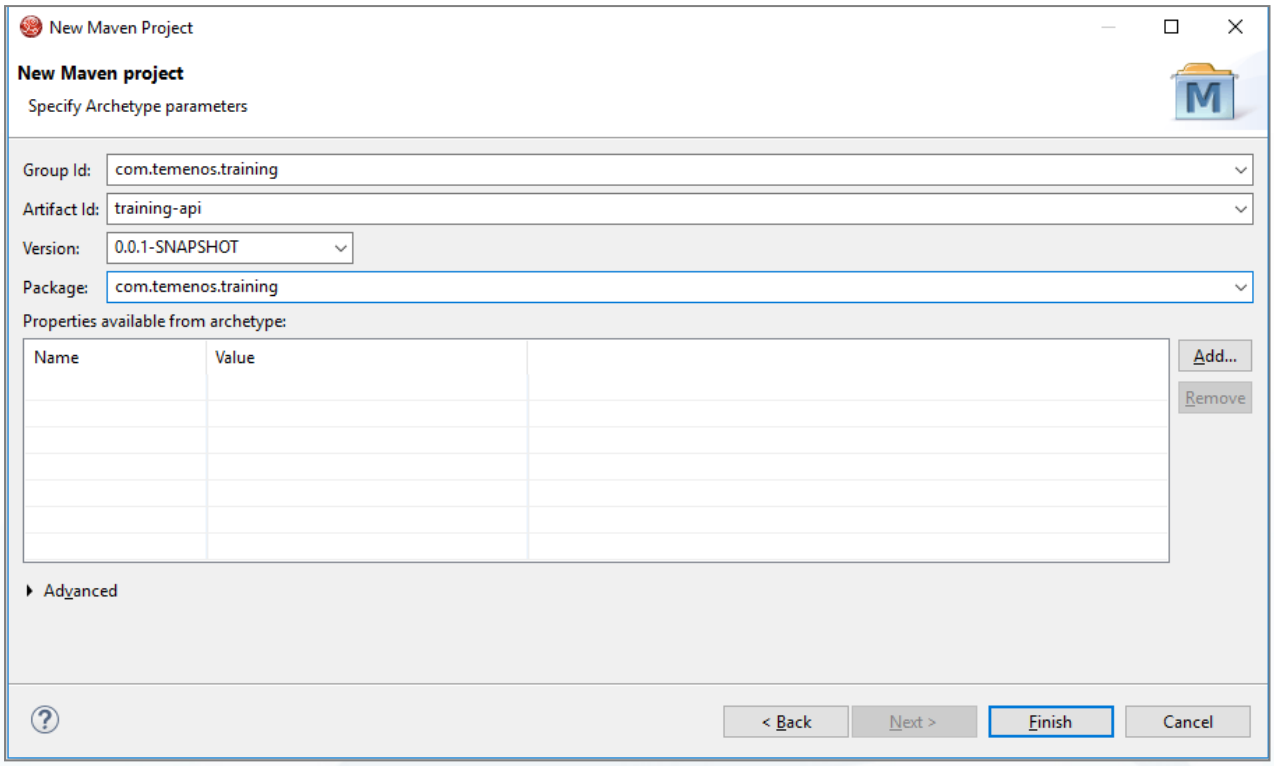
Select archetype

Select the **irf-service-archetype** option from Artifact Id column and click the **Next** button.



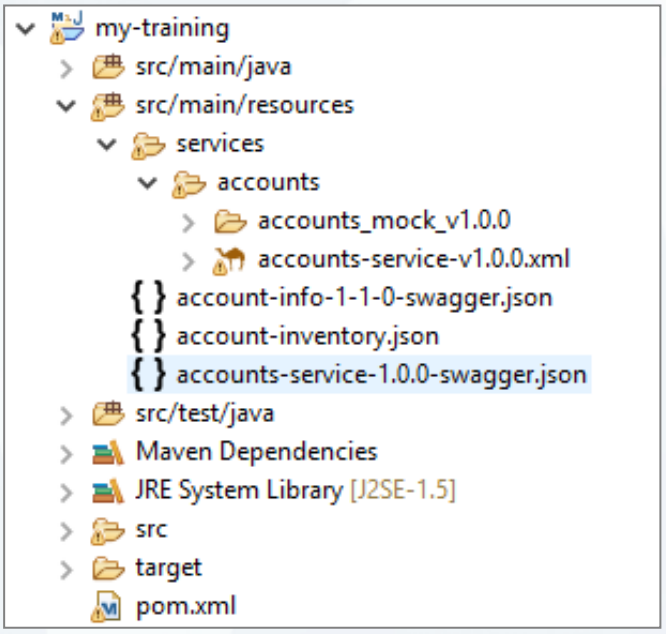
Provide Project details

Fill in the **Group Id**, **Artifact Id** and **Version** fields, and click the **Finish** button.



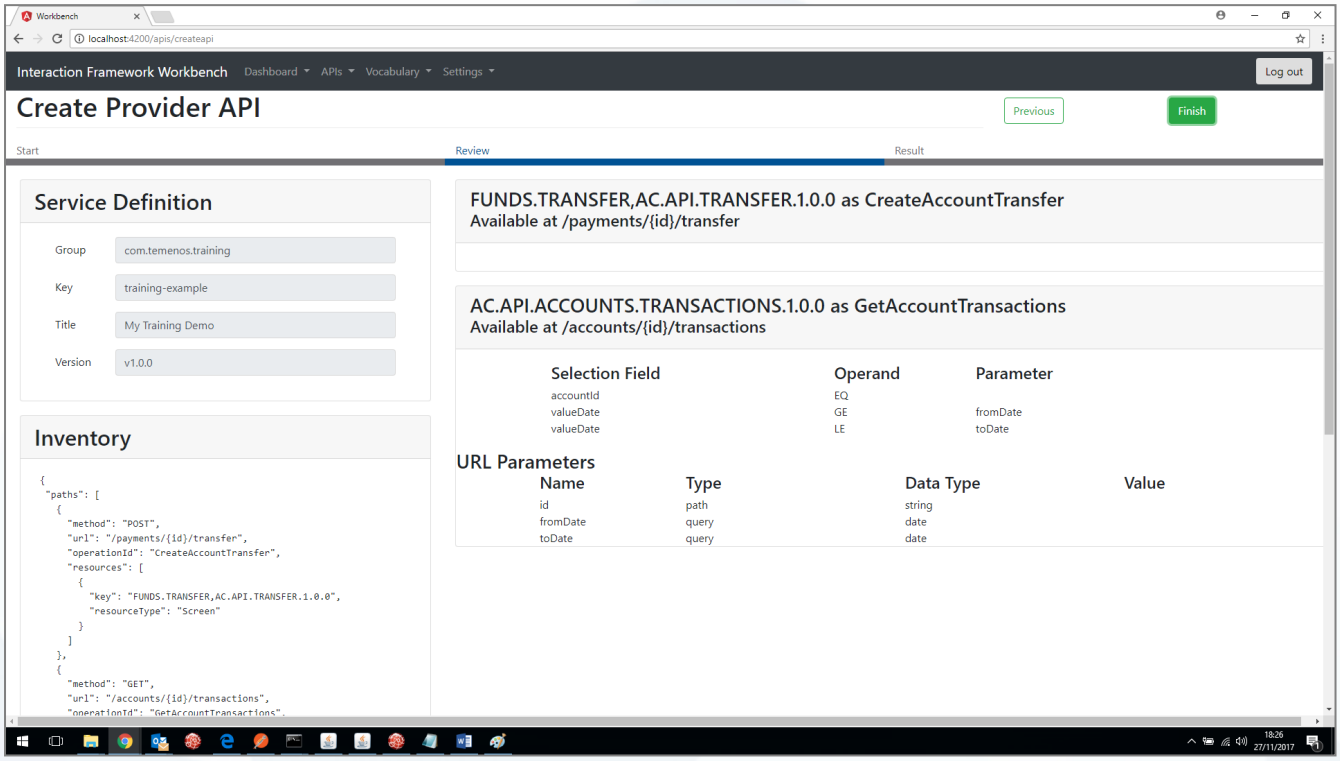
Project Structure

You should see a project created with the following structure:

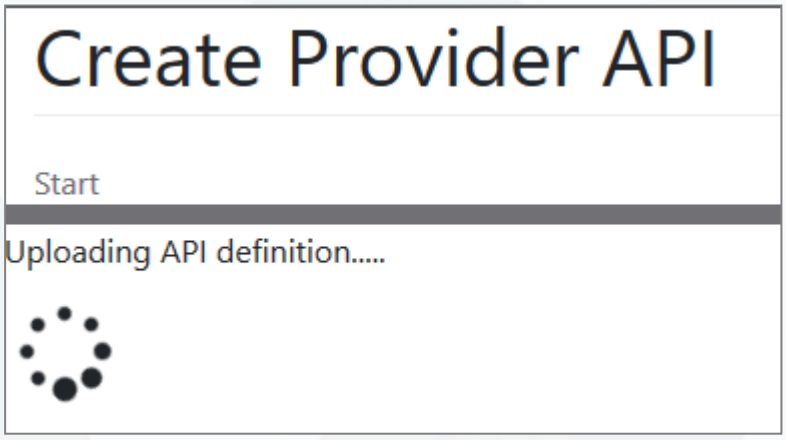


Submit the Provider API

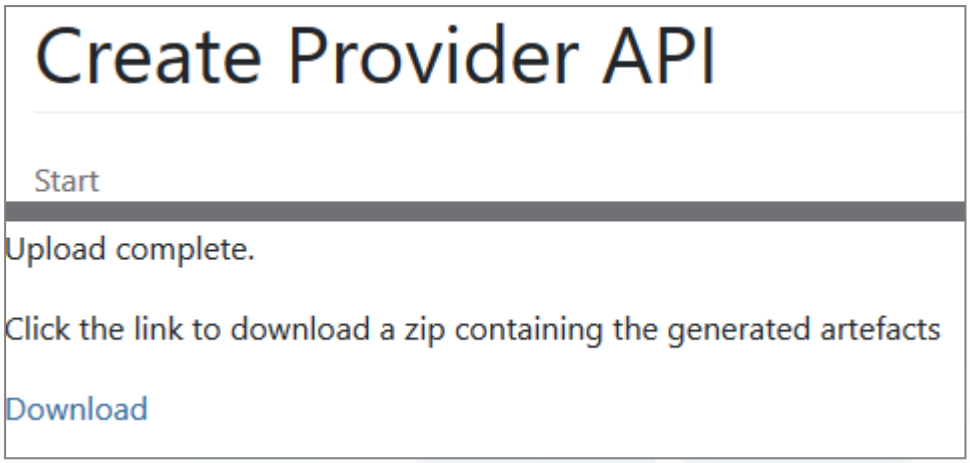
After creation of inventory file as shown in Step 4 of the Creating an Inventory using Interaction Framework Workbench.



Click Finish in the top right of the screen to submit the provider API.



The result provides a link to download a zip file containing the swagger spec, Camel routes, mappings and mock responses.



Build the Service Project

Run the project as Maven > Install to build the project

Click here to know about **T24 Connectivity Configuration TAFJ and TAFC**

Deploy and Test Provider API

Add to the Service Container Project

In your service container project, amend the pom.xml to include your API as a dependency:

NB There are several dependencies tags in the pom.xml file - this dependency needs to be added to the MAIN dependency, below the comment:

<!-- Add any project dependencies here-->

```
<dependency>

<groupId>com.temenos.training</groupId>

    <artifactId>provider-accounts</artifactId>

<version>0.0.1-SNAPSHOT</version>

</dependency>
```

The service container should restart automatically, if not stop the server and restart manually.

Check Deployment

Use Postman to make a GET request to the management service

```
localhost:8080/api/v1.0.0/meta/apis
```

You should see the servicer returned in the response:

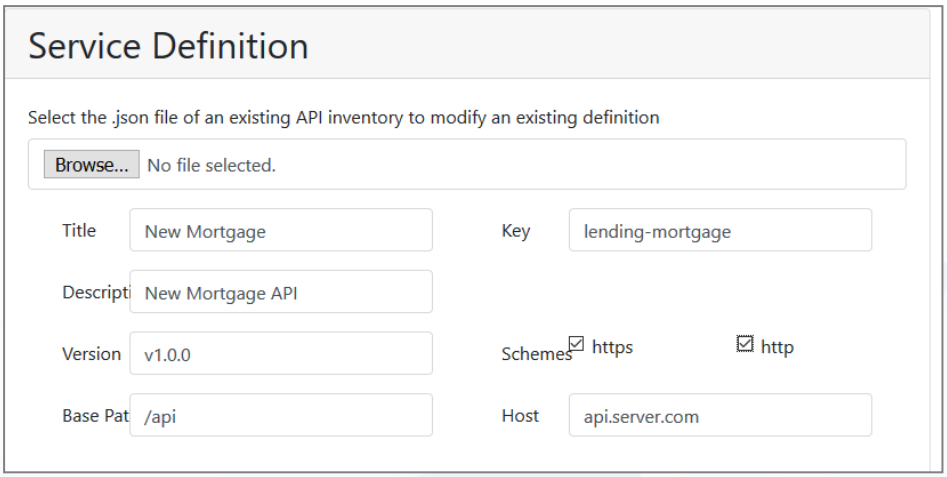
```
{

    "header": {

        "serviceCount": 9,
```

```
        "audit": {
            "processTime": 9
        }
    },
    "body": [
        {
            "service": "provider-accounts.service.v1.0.0",
            "endPoints": [
                {
                   "method": "POST",
                    "uri": "/v1.0.0/provider-accounts/payments"
                },
                {
                    "method": "GET",
                    "uri": "/v1.0.0/provider-accounts/{accountId}/transactions"
                }
            ]
        },
        {
            "service": "accounts.service.v1.0.0",
            "endPoints": [
                {
                    "method": "GET",
                    "uri": "/v1.0.0/accounts/{AccountId}/transactions"
                }
            ]
        },
```

Exposing AA Activities as Provider APIs

Service configuration

Define the Service Definition parameters for "New Arrangement creation" for a Mortgage loan product.



Select the "LENDING" option under the "Available artifact" section and add it.

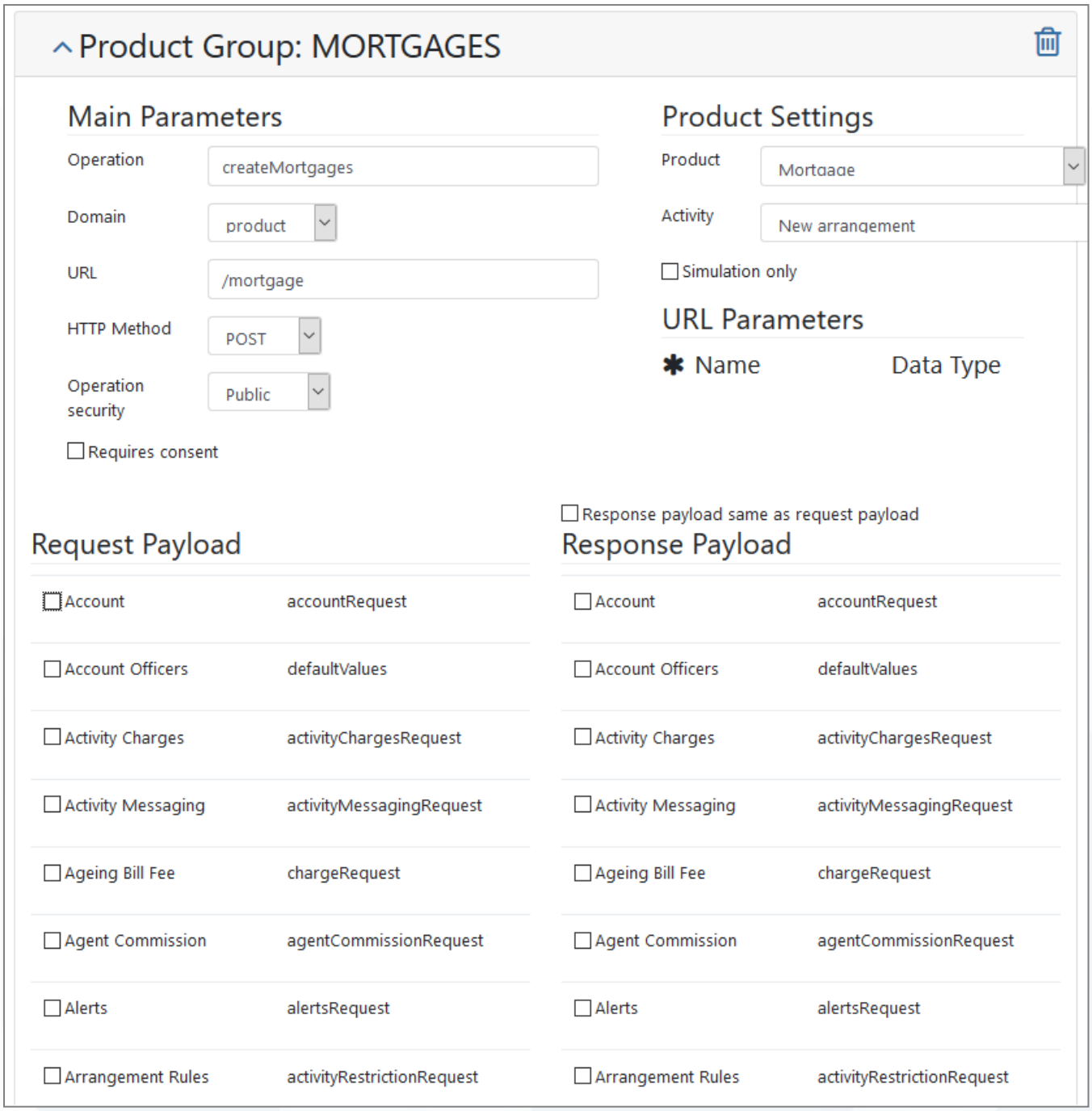
Enter Activity & Product Details

Select the product group **MORTGAGES** from the list and click Product: **Mortgage** and Activity: **New Arrangement**.

Select-able payload properties should appear on the screen.

Click the request and response payload tick boxes to specify the input (**Request Payload**) and output (**Response Payload**). In case both of them are the same, click "**Response payload same as request payload**".

Default behavior, i.e., in case no boxes are selected, will result in all the applicable properties to be used. Response payload version can be configured by creating **AA.ARR.{property class},AA.API.RESPONSE.1.0.0** For example: **AA.ARR.ACCOUNT,AA.API.RESPONSE.1.0.0**



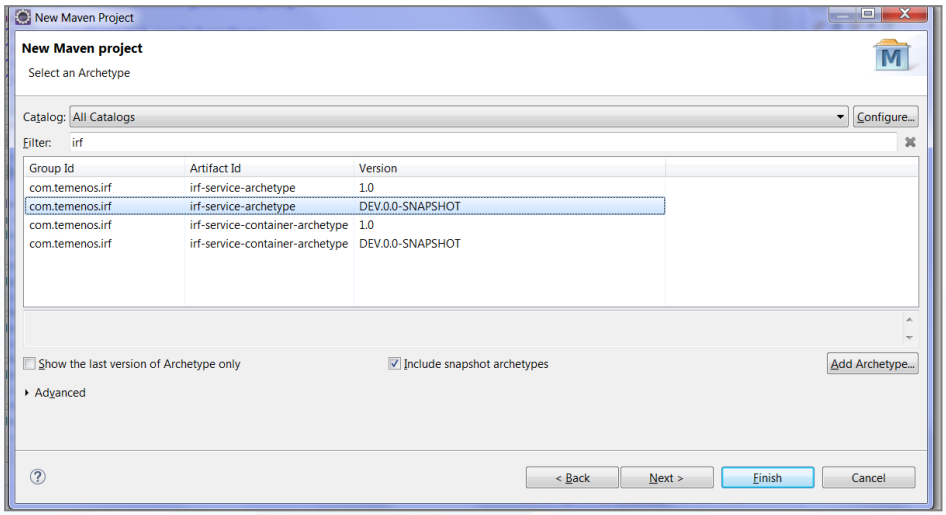
Download the Generated Artifacts

Click Next and click Finish.

Workbench displays a URL link to download the generated artifacts.

Create Lending API project using eclipse Maven Archetype

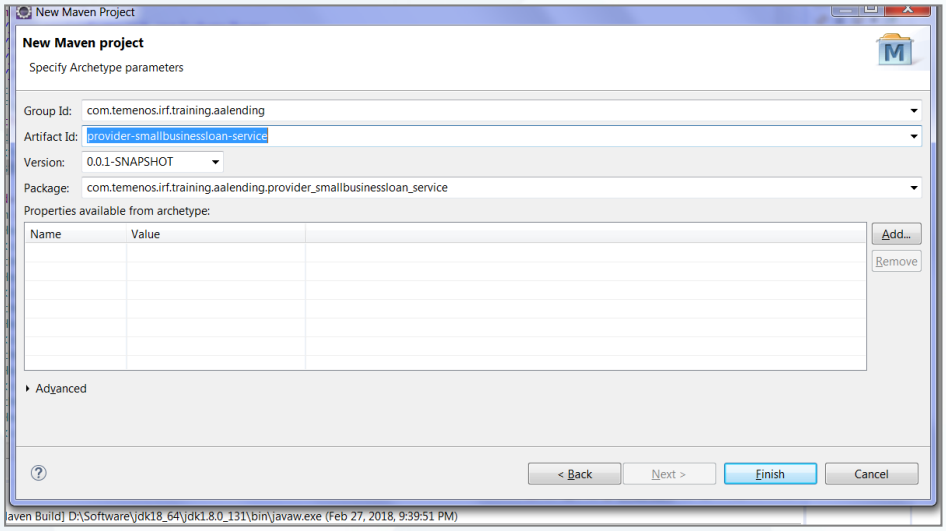
1. Create provider api project for Lending Product using the Maven Arch type.
    a. Select "**irf-service-archetype**" of Dev.0.0.0-Snapshot Version



    b. Enter Service API project details

    Group Id : com.temenos.irf.training.aalending

    Artifact id : provider-smallbusinessloan-service



    Click Finish.

Import and Build the Generated Artifacts

1. Extract the generated artifacts into the resources/services folder of the Lending API project created from the Service API maven archetype.
2. Buildthe Service API project using maven command
   mvn clean install

3. Add dependencies of API maven project in the Container maven pom.xml under "dependencies" section.

```
<dependency>
<groupId>com.temenos.irf.training.aalending</groupId>
<artifactId>provider-smallbusinessloan-service</artifactId>
<version>0.0.1-SNAPSHOT</version>
</dependency>
```

4. Build and Restart the Container project

Test Mortgage Loan Service

1. Test the Service from Postman using the below Url
    a. URL :http://localhost:8080/api/v1.0.0/product/mortgage
    b. data :

```
{
"header": {


},
"body": {
"parties": [
{
"partyId": "100100",
"partyRole": "USD"
}
],
"currencyId": "USD",
"commitment": {
"amount": "7000",
"term": "1Y"
}
}
}
```

    and Post the request
2. Get success response with the Arrangement details

Replace Indicator

By default, all PUT request messages are mapped with OFS replace Indicator.

Messages (update messages) with replace indicator tells T24 to update the record by regenerating the entire record. Hence, the mandatory fields required by the record generation process has to be passed to T24 for a successful output. To learn more about OFS replace indicator, please refer to OFS User Guides.

To disable replace indicator functionality, add the following in your service XML.

```
...
<setProperty propertyName="ignoreReplace">
<constant>true</constant>
</setProperty>
...
```

The feature Replace Field Indicator is added. When using this feature, only the fields/field-set (in case, of multivalue/subvalue) has to be passed, instead of the entire record.

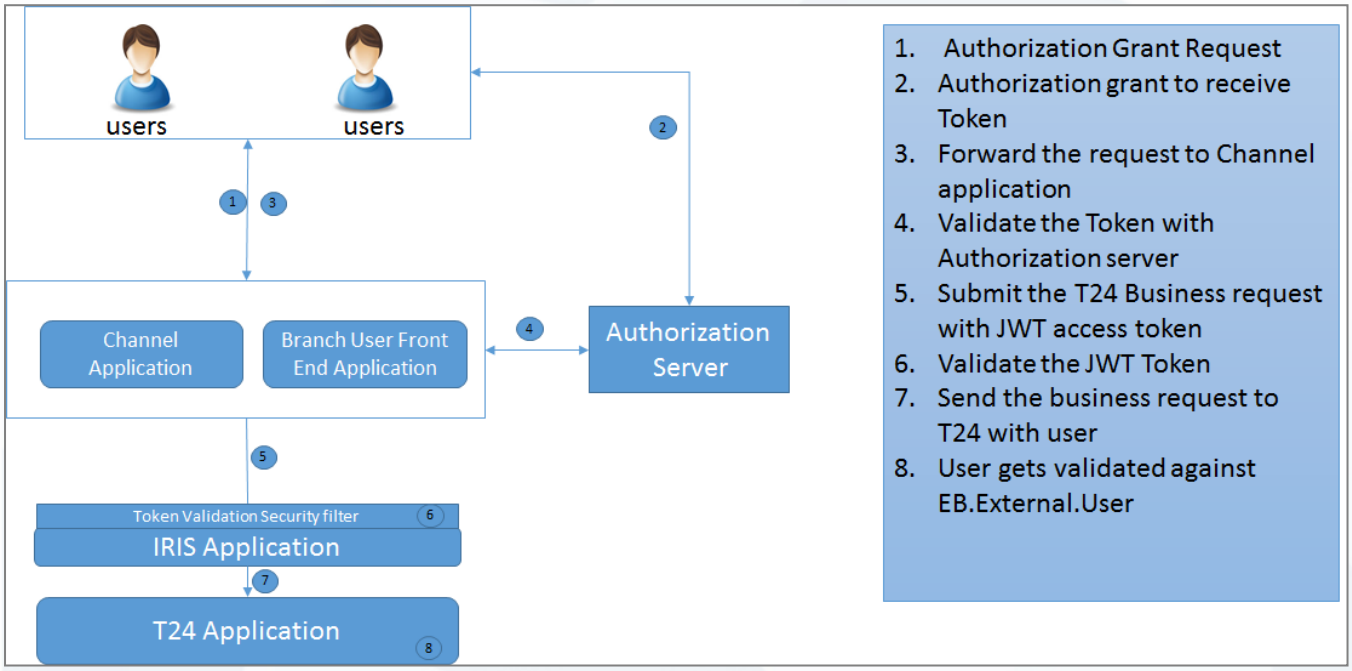
To enable it and remove previous replace indicator functionality, edit the bean **t24replaceFieldIndicator** in container **applicationContext.xml** with the value **true**.

```
...
<bean id="t24replaceFieldIndicator" class="java.lang.Boolean">
<constructor-arg value="true" />
</bean>
...
```

Security

Token Based Security Filter Implementation

IRIS Data Services can be supported as an intermediary in an authenticated request with the JWT token as a mechanism for authentication and identification with the help of this functionality. The handling of the JWT token is expected to be fully configurable to support various deployment choices from naive authenticator to the most secure one.



The JWT Token based authentication solution is designed as pluggable and extendable through configuration.

Click here to go to the **Configurations** page.

Configurations

Spring Security Filter Enablement

IRIS web application gets enabled with Spring security filter chains by configuring respective tags in the web.xml of IRIS application. **springSecurityFilterChain** Filter should be added to enable spring security.

```
<filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>
```

Spring Security Configurations

spring-jwt-iris-authenticator.xml – is used to define spring beans and should be loaded as a part of configuration. Add spring configuration xml in the web configuration settings.

```
 <display-name>Service Container</display-name>
<context-param>
     <param-name>contextConfigLocation</param-name>
     <param-value>
     classpath:applicationContext.xml
     classpath:spring-jwt-iris-authenticator.xml
     </param-value>
 </context-param>
<!-- Camel servlet -->
```

spring-jwt-iris-authenticator.xml Configuration

This authenticator xml allows the user to extend the IRIS infrastructure based on their token validation requirements.

Below section helps to configure the Spring Security Authentication Provider and the Token Validation filter based on the user requirement.

```
<context:component-scan base-package="com.temenos.security.oidc" />
<context:spring-configured />
<security:http entry-point-ref="authenticationEntryPoint">
    <security:custom-filter after="BASIC_AUTH_FILTER" ref="jwtTokenValidationFilter" />
    <security:session-management>
        <security:concurrency-control max-sessions="1" />
    </security:session-management>
</security:http>
<beans:bean id="jwtAuthenticationProvider" class="com.temenos.irf.web.security.jwt.filter.JWTAuthenticationProvider">



 <security:authentication-manager alias="irfam">
    <!-- <security:authentication-provider ref="oidcAuthenticationProvider" /> -->
      <security:authentication-provider ref="jwtAuthenticationProvider" />
</security:authentication-manager>



<beans:bean id="jwtTokenValidationFilter" class="com.temenos.irf.web.security.jwt.filter.JWTTokenValidationFilter">
    <beans:property name="authenticationManager" ref="irfam" />
    <beans:property name="authenticationSuccessHandler">
        <beans:bean class="com.temenos.security.oidc.filter.OidcAuthenticationSuccessHandler" />
    </beans:property>
</beans:bean>
```

Out of the box, Temenos provides Validation filter and Authentication provider to validate the JWT token. The Validation filter uses this configuration XML to perform the algorithm, claim, and Signature validations.

I. **AlgorithmValidator**

   The ‘alg’ value should be the default of RS256 or any one of the values given under RelyingPartyClient property “idTokenSignedAlg”.

   The ‘kid’ optional parameter, if present, is used to identify the key for decrypting the token applying the valid algorithm in the ‘alg’ claim.

II. **SignatureValidator**

   A signed token needs to be validated for the signature to verify the integrity of the token.

If the client is configured to support an unsigned token and there is no signature then there is no signature validation required.

III. **ClaimValidator**

ClaimValidator is a generic validator for various claims in the token. Each instance of the ClaimValidator would validate one specific claim. The claim that needs to be validated is injected into the instance.

The following are the claims default validators supported for:

1. exp
2. iss
3. aud
4. azp
5. iat

IV. **CustomValidator**

A generic custom validator can be defined through which any number of custom validations could be written and plugged into the validator composite through simple spring beans configuration.

For example, if the user wants to implement a custom validator to perform custom validation with authentication provider, user can implement additional validator by implementing base interface of the validation which provides token and the http request objects to the custom validator.

Custom validator needs to extend "**com.temenos.security.oidc.token.validator. TokenValidator**" and should implement "**verify**" method

```
@Override
public ValidationResult verify(String token) {
```

Upon successful authentication by obtaining and validating the token, the validation filter builds and propagates the authenticated principal and invokes the remaining filter change as normal.

Spring Security uses **org.springframework.security.core.userdetails.UserDetails** to manage user information across the application. Once the requests get authenticated, Spring security made userDetails available as a part of Spring Authentication object.

IRISAuthenticatedUser is the iris authenticated user object used to carry user data as a part of Spring Authentication object.

On successful authentication of the token, the successfulAuthentication callback sets Spring Authentication object references in the Spring Security Context. The Security Context holds the user information (IRISAuthenticatedUser) which is available across the IRIS Version and Enquiry processors.

T24 Security Filter Configuration

IRIS T24 Processors uses T24 Security Context to construct OFS message with the respective user credentials. T24SecurityFilter derives the User principal from the Spring Security Context and constructs user's T24 security context.

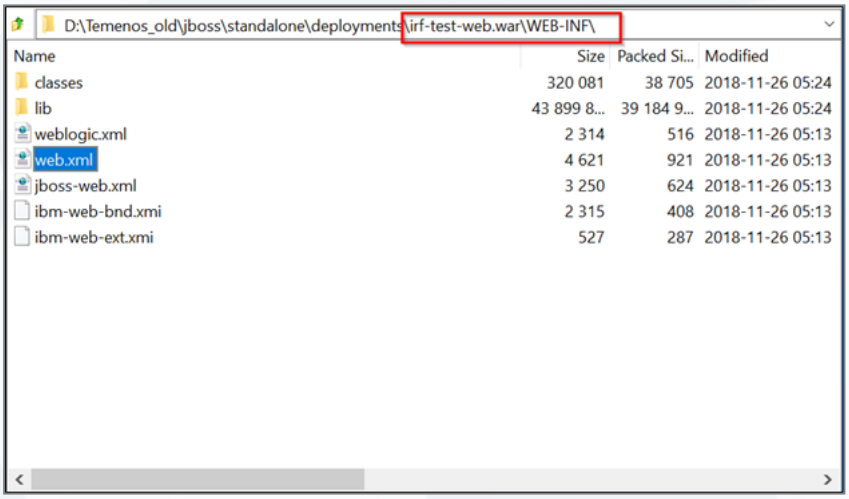
Out of the box, Temenos provides a T24 Security filter "com.temenos.irf.comms.security.AuthImpl.T24SpringSecurityContextFilter". This T24 Security filter retrieves the user principal from spring security context and constructs T24 Security context which is used for constructing the OFS message.

```
<bean id="t24SecurityFilter" class="com.temenos.irf.comms.security.AuthImpl.T24SpringSecurityContextFilter"> </bean>
```

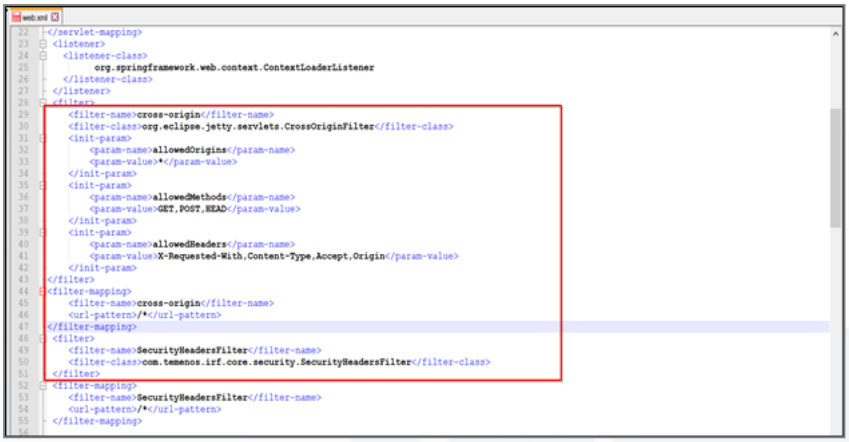
```
SecurityContext context =   SecurityContextHolder.getContext();
Authentication authResult = context.getAuthentication();
IRISAuthenticatedUser authUser =(IRISAuthenticatedUser) authResult.getPrincipal();
```

Security Headers Missing & Cross-Origin Resource Sharing: Arbitrary Origin Trusted

The Security Headers are enabled by configuring the web.xml as shown in the screenshot below:



Uncomment the cross-origin (CORS) filter and **SecurityHeadersFilter** as shown in the screenshot below.



To configure a strong **Content Security Policy**, follow these best practices:

- Specify a default policy. It is recommended to set it to 'none'.
- Avoid setting default-src and script-src to 'unsafe-inline' or data: The 'data:' URIs can be used to inject and execute JavaScript.
- Avoid setting default-src and script-src to 'unsafe-eval'. The 'unsafe-eval' policy allows the use of eval() and similar methods which can be abused for code injection.
- Specify a whitelist of sources for content source directives (child-src, connect-src, default-src, font-src, frame-src, img-src, media-src, object-src, script-src, and style-src) and do not set these directives to *.
- Avoid setting script-src or default-src to 'self' for hosts containing Angular applications, JSONP endpoints, or files uploaded from users, as attackers can manipulate such applications to execute attacker's code uploaded to the application's host.
- Avoid setting the frame-ancestors and form-action directives to *, as it may allow click-jacking and form data hijacking attacks.
- Only use third-party URIs that start with https: as content loaded over HTTP can be modified by an attacker.

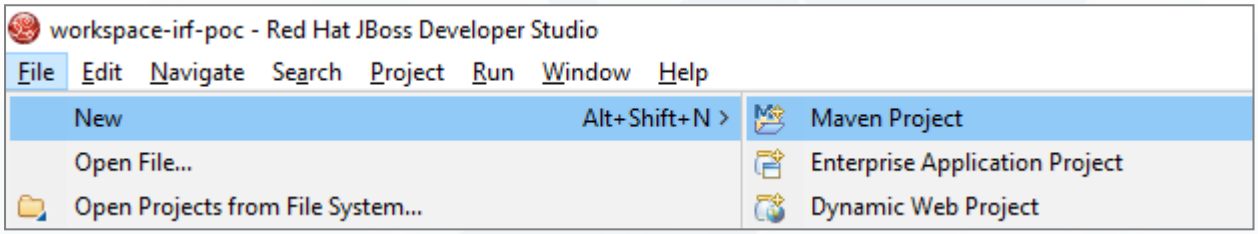
Creation of Published API

Overview

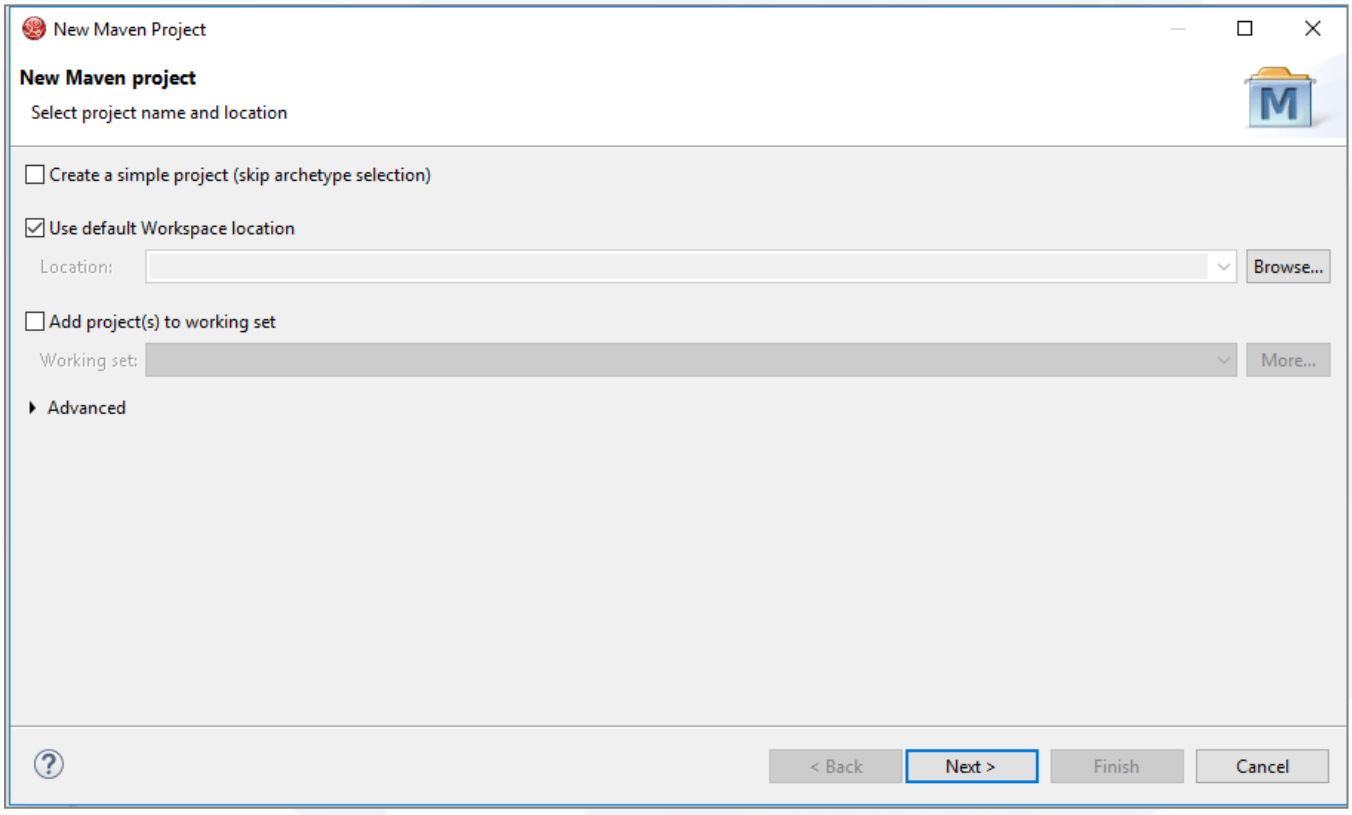
An API defined in Swagger 2.0 may be imported into an API project. The import process will create the Camel routes, mock responses and wire the routes to the mock responses. This is done using a Maven plugin, either from your IDE or from the command line.

Creation of Provider Service Project

From the File menu, select **File** > **New** > **Maven Project**

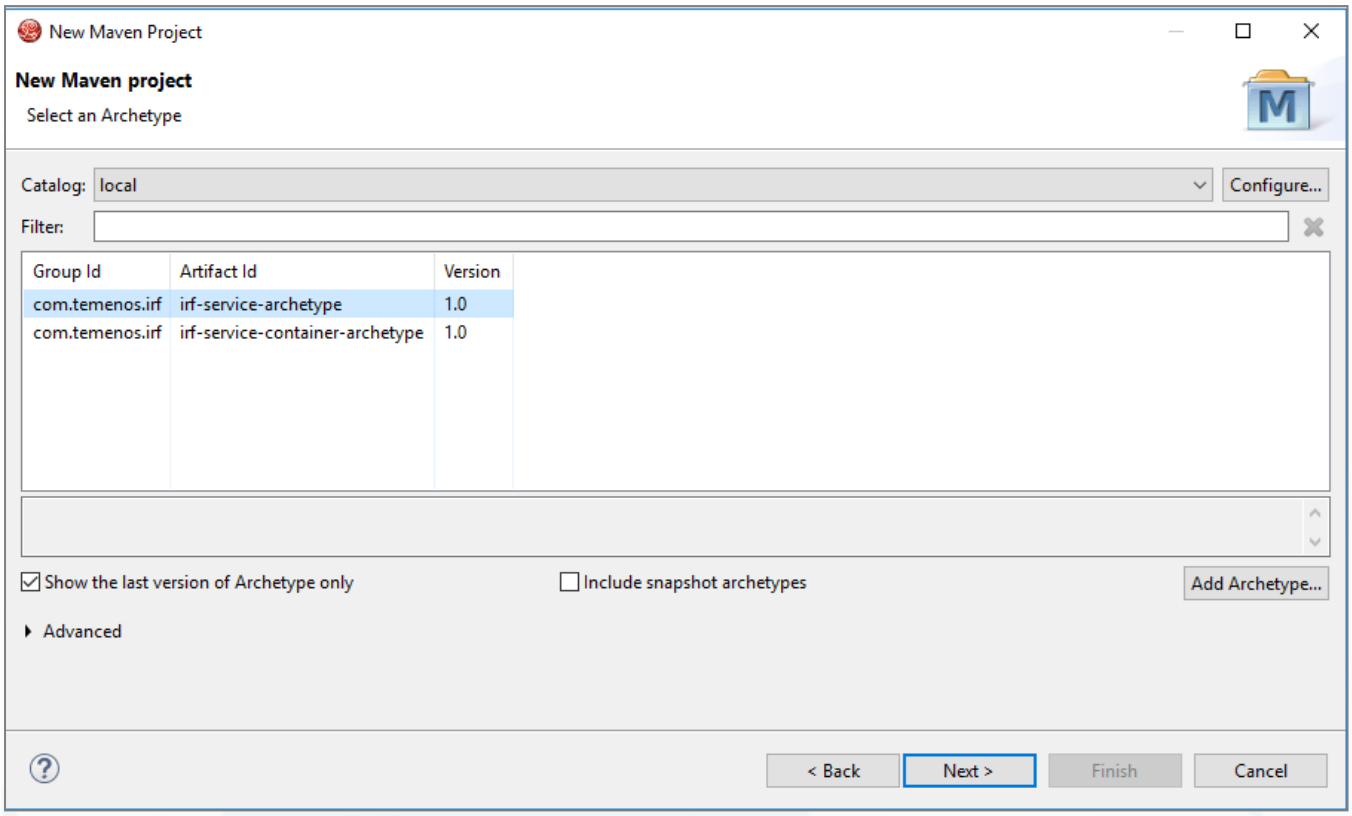


Click **Next**



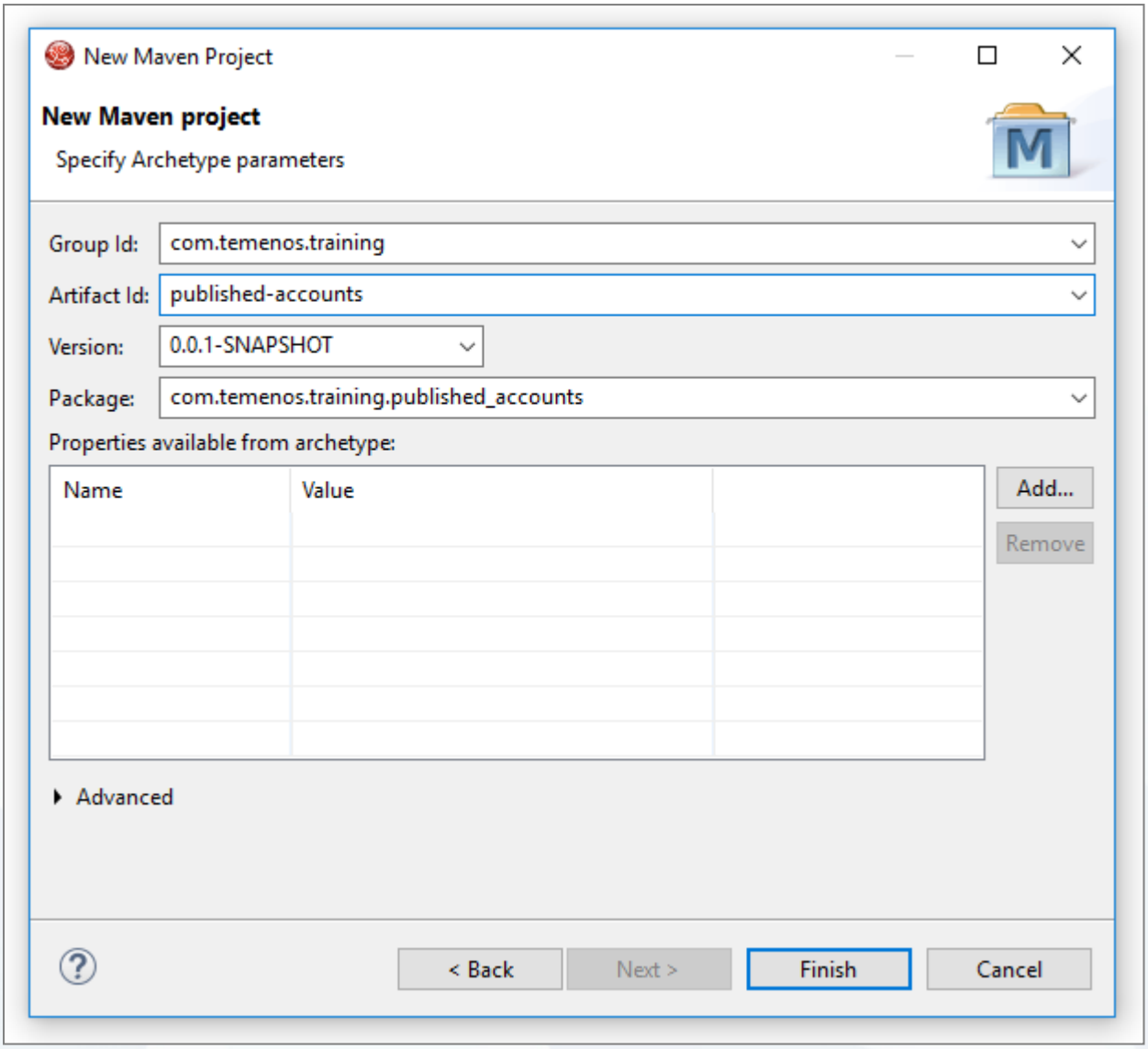
Select archetype

Select the **irf-service-archetype** and click **next**



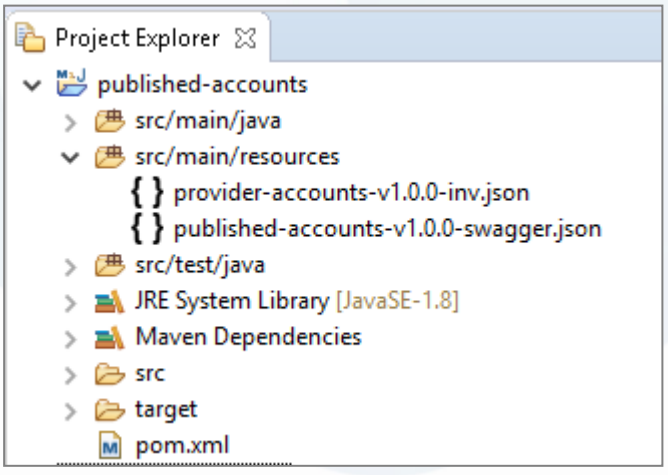
Provide project details

Fill in the **Group Id**, **Artifact Id** and **Version** fields, and click **Finish**.



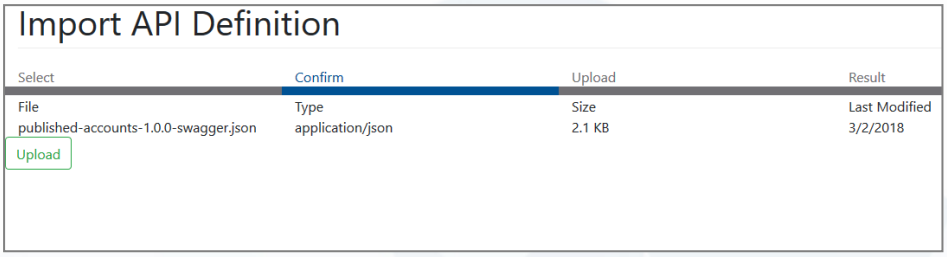
Project Structure

You should see a project created with the following structure:

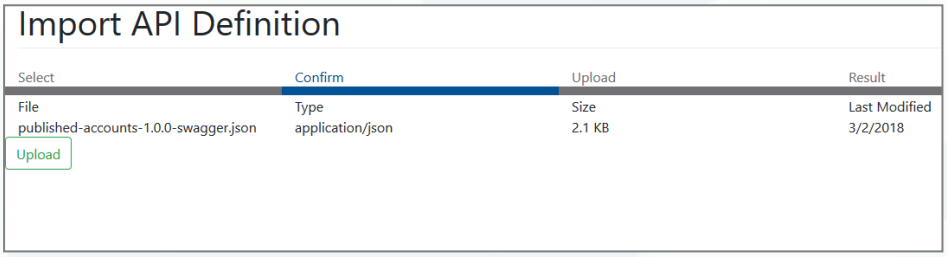


Import API Spec

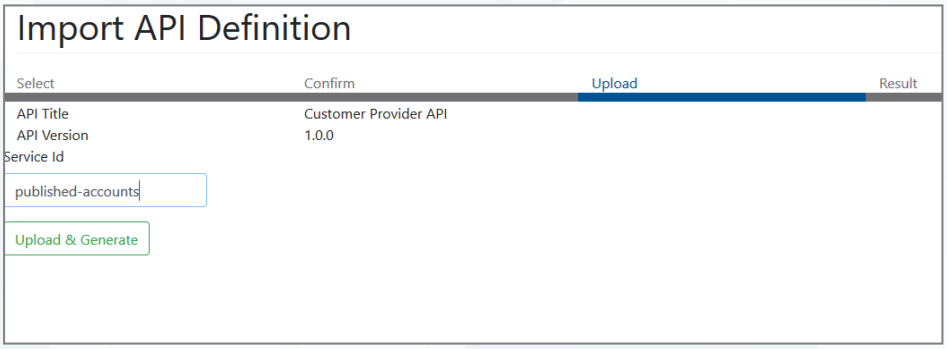
The import process is done using the Interaction Framework Workbench. Go to APIs > Import API Definition. You should be able to see the following screen.



Select the publisher swagger definition (swagger json file) that you want to import and then click Upload



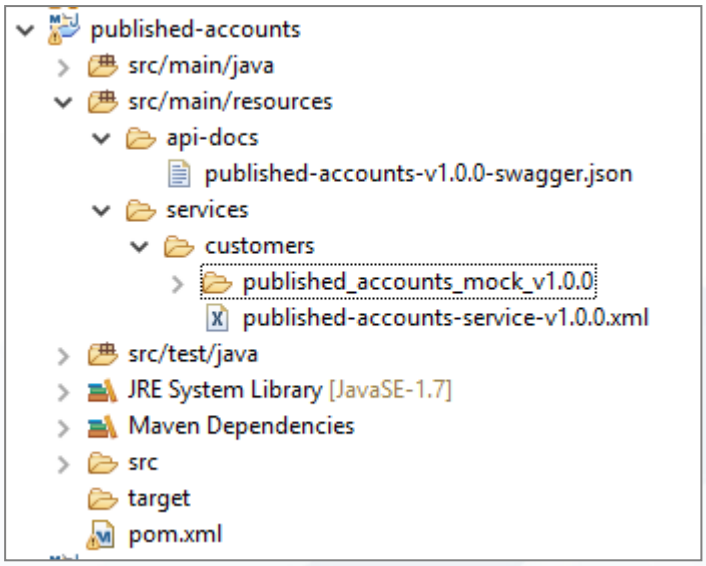
Give your service a name in the next screen and click Upload & Generate



Download and extract the zip and place the generated files in the service project

Project Structure

Refresh the project view and verify the project structure:



Provider API to Published API Mapping

Map path params from publisher to provider

Route for published Funds Transfer Status Report

```
...
<rest path="/v1.0.0/published" produces="application/json" id="published.service.restlet">
<get uri="/payments/{accountid}/statusreport" id="PublishedPaymentStatusReport">
<param name="accountid" type="path" required="true"/>
<to uri="direct:published.PublishedPaymentStatusReport"/>
</get>
</rest>
...
<route id="direct.published.PublishedPaymentStatusReport">
<from uri="direct:published.PublishedPaymentStatusReport"/>
<to uri="direct-vm:order-PSD2.getPaymentTransferList" />
<process ref="FTStatusReportResponseMapper"/>
</route>
```

> Note: Use "direct-vm" component to map all the publisher path params to provider path params. This will enable it to look through all the CamelContexts present in the same JVM and route to the underlying provider API.

Save all the files, and build the publisher api project as Maven > Install.

Sample Publisher route for the reference

Additional Beans before CamelContext tag

```
...

<bean id="FTInitiationRequestMapper" class="com.temenos.irf.published_FT_api.FTIni-
tiationRequestProcessor"/>

<bean id="FTInitiationResponseMapper" class="com.temenos.irf.published_FT_api.FTIni-
tiationResponseProcessor"/>

<bean id="FTStatusReportResponseMapper" class="com.temenos.irf.published_FT_api.FTStatusRe-
portResponseProcessor"/>

<bean id="FTSubmissionResponseMapper" class="com.temenos.irf.published_FT_api.FTSub-
missionResponseProcessor"/>

...
```

Published FT Status Report Route

```
...

<get uri="/payments/{accountid}/statusreport" id="PublishedPaymentStatusReport">

<param name="accountid" type="path" required="true"/>

<to uri="direct:published.PublishedPaymentStatusReport"/>

</get>

...

<route id="direct.published.PublishedPaymentStatusReport">

<from uri="direct:published.PublishedPaymentStatusReport"/>

<to uri="direct-vm:order-PSD2.getPaymentTransferList" />

<process ref="FTStatusReportResponseMapper"/>

</route>
```

Published FT Initiation Route

```
...

<post uri="/payments/transfer/initiation" id="PublishedPaymentTransferInitiation">

<to uri="direct:published.PublishedPaymentTransferInitiation"/>

</post>

...

<route id="direct.published.PublishedPaymentTransferInitiation">

<from uri="direct:published.PublishedPaymentTransferInitiation"/>
```

```
<process ref="FTInitiationRequestMapper"/>

<to uri="direct-vm:order-PSD2.getPaymentTransferList" />

<process ref="FTInitiationResponseMapper"/>

</route>
```

Published FT Submission Route

```
...

<post uri="/payments/transfer/{id}/submission" id="PublishedPaymentTransferSubmission">

<param name="id" type="path" required="true"/>

<to uri="direct:published.PublishedPaymentTransferSubmission"/>

</post>

...

<route id="direct.published.PublishedPaymentTransferSubmission">

<from uri="direct:published.PublishedPaymentTransferSubmission"/>

<to uri="direct-vm:order-PSD2.getPaymentTransferList" />

<process ref="FTSubmissionResponseMapper"/>

</route>
```

The request and response mappers configured will have java code to transform from publisher json format to provider json and vice versa.

Deployment and Configuration

This section covers the following topics.

Deployment options

Once the service project is build, the user can deploy it using either of the following application servers:

- Jboss
- Weblogic
- Websphere

You can configure the IRIS R18 solution using TAFJ (JMS Connector) or TAFC (TCS and TOCFEE Connectors).

T24 Connectivity Configuration TAFJ and TAFC

The following connectors are used in connectivity configuration of TAFJ and TAFC:

JMS Connector - TAFJ & TAFC TOCFEE

From T24 R09, the Temenos Open Connectivity Framework has been updated to allow deployment in Java Enterprise Edition application servers – TOCF (EE). TOCF (EE) is a family of JEE compliant components providing connectivity to T24. However, in addition to utilising industry standard connectivity mechanisms for security, high availability and connection pooling, TOCF (EE) also facilitates connectivity to external systems from T24.

JMS (Java Message Service) is an API that provides the facility to create, send and read messages. It provides loosely coupled, reliable and asynchronous communication.

JMS Connector provides JMS based connector implementation for Interaction Framework X to interface with T24 using OFS Message-based interface.

Deployment Configuration

1. JMS Resource Configuration required in the Deployment Descriptor

   The following resources references are expected to get configured to use the respective JNDI resources available in the application server:

   - queue/t24OFSQueue – need to get mapped to **OFS Request Queue 's JNDI resource**.
   - queue/t24OFSReplyQueue – need to get mapped to **OFS Response Queue's JNDI Resource**.
   - jms/jmsConnectionFactory – need to get mapped to **JMS Connection factory Resource**.
2. Application Context - Spring bean configuration

Below configuration is required in the Application Context.xml to load JMS Based connector factory.

```
<bean id="t24JmsConnectionProperties" class-
s="com.temenos.irf.config.StandardPropertyReader">
<property name="path" value="irf-config/jms.properties"/>
</bean>

<bean id="t24JMSConnectionFactory" class-
s="com.temenos.irf.comms.jms.JMSConnectorFactory">
<property name="propertyReader" ref="t24JmsConnectionProperties"/>
</bean>
```

irf-config/jms.properties — Contains the following set of configurations to handle retry in case of JMS Queue or Connection related failures:

- RetryWait=60
- RetryCount=3
- ConnectionTimeout=30

3. Remote Connectivity

   This section is applicable only when Interaction Framework components and TAFJ application are deployed in different Servers (specifically JBOSS).

   - Application context configuration

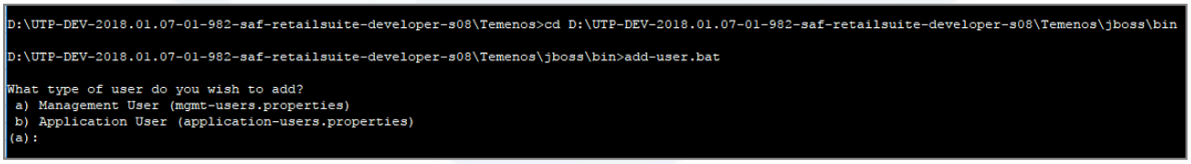
     Application Context configuration in applicationContext.xml will be

     ```
     <bean id="t24ConnectionProperties" class-
     s="com.temenos.irf.config.StandardPropertyReader">

     <property name="path" value="../irf-config/standalone-comms remote.properties"/>

     </bean>

     <bean id="serviceLocatorProperties" class-
     s="com.temenos.irf.config.StandardPropertyReader">

     <property name="path" value="../irf-config/service-locator.properties" />

     </bean>
     ```

   - JBoss Management User Creation

     To access TAFJ or TOCFEE JMS Queues, a management user needs to be created in JBOSS ManagementRealm.

     RUN add-user.bat/add-user.sh available in <JBOSS_HOME>/bin directory.

The user id and password set using add-user.bat/add-user.sh in the JBoss environment needs to be configured in standalone-comms remote.properties.

- standalone-comms remote.properties

  Sample content of standalone-comms remote.properties is as shown below

```
java.naming.factory.initial=org.jboss.naming.remote.client.InitialContextFactory

java.naming.provider.url=http-remoting://<remote host name>:9089

t24.security.context=INPUTT/xxxxxxx

java.naming.security.principal=mguser1
java.naming.security.credentials=xxxxxxx
```

  java.naming.provider.url — Remote URL of JBoss instance where TAFJ/TOCFEE is deployed

  java.naming.security.principal — Management User created in Remote JBOSS instance

  t24.security.context=INPUTT/xxxxxxx – T24 User to be used to connect T24

  java.naming.security.credentials - Credentials of the Management User (to be saved as plain text)

  internal.reqque.jndiname=jms/queue/t24OFSQueue

  internal.respque.jndiname=jms/queue/t24OFSReplyQueue

  external.reqque.jndiname=jms/queue/t24TCIBQueue

  external.respque.jndiname=jms/queue/t24TCIBReplyQueue

Usage

JMS Connector will be used as a T24 interface connector in Interaction Framework X where

T24 Deployment mode is:

- TAFJ or
- TAFC - where TOCFEE component is enabled.

TCS Connector - TAFC TCS

In earlier releases of T24, Temenos supported T24 Server Services (TSS) which provided a programmable interface for external application to interact with T24 over TCP/IP. The TCS client distribution has necessary libraries to connect to TSS and exchange messages for business transactions. In case of T24 deployment where TSS services are running, TCS connector needs to be enabled for T24 integration.

TCS Connector provides TCS based connector implementation for Interaction Framework X to interface with T24 using OFS Message-based interface.

Deployment Configuration

1. TCS Resource Configuration is required in the Deployment Descriptor

   The following resources references are expected to get configured to use the respective resources available in the Application Server:

   - tcs/tcsConnectionFactory – should get mapped to **TCS Connection factory Resource**.
   - Need to start listeners by invoking the TCServer.bat.

2. Application Context – Spring bean Configuration

   - Below configuration is required in the ApplicationContext.xml to load TCS Based connector factory.

   ```
   <bean id="t24ConnectionProperties" class="com.temenos.irf.config.StandardPropertyReader">
   <property name="path" value="../irf-config/tafc-connection.properties"/>
   </bean>

   <bean id="t24TafcConnectionFactory" class="com.temenos.irf.tcs.TCSConnectionFactory">
   <property name="propertyReader" ref="t24ConnectionProperties"/>
   </bean>
   ```

Usage

TCS Connector will be used as a T24 interface connector in Interaction Framework X where T24 Deployment mode is TAFC.

SSL Configuration for IRIS R18 on Jboss 7

This explains you how to setup SSL/HTTPS support for your Jboss 7 server using the pure Java implementation supplied by JSSE.

Firstly, you have to configure keys and (self-signed) certificates for your web server. This guide will briefly explain how to accomplish that.

- **Pure Java SSL-Setup using keytool**

  Generate a secret key/certificate and store it in a file called a "key store". The certificate is valid for 1 year = 365 days. The password use for encryption is "temenos".

  1. **Step 1: Generate key**

     $ keytool -genkey -alias temenos -keyalg RSA -keystore temenos.keystore -validity 365 (***open a cmd prompt pointing to the bin directory eg: ..\jdk8\bin & execute the given command to generate the key***)

     Command Line

```
Enter keystore password: temenos

Re-enter new password: temenos

What is your first and last name?

[Unknown]: localhost

What is the name of your organizational unit?

[Unknown]: IT

What is the name of your organization?

[Unknown]: temenos

What is the name of your City or Locality?

[Unknown]: Bangalore

What is the name of your State or Province?

[Unknown]: Karnataka

What is the two-letter country code for this unit?

[Unknown]: IN

Is CN=localhost, OU=IT, O=temenos, L=Bangalore, ST=Karnataka, C=IN correct?

[no]: yes
```

```
Enter key password for <deva> temenos

(RETURN if same as keystore password):

Re-enter new password: temenos
```

You will find a key generated "temenos.keystore" in the same jdk bin directory.

2. **Step 2: Configure JBoss 7**

   Once the key is generated, move the key to the configuration directory of your application server(in this case \jboss\standalone\configuration\).Inthestandalone.xmlconfigfile,underthesubsystem<subsystem xmlns="urn:jboss:domain:undertow:3.1">, include a http-listener for "https" as follows : (include the highlighted line)

Standalone.xml

```
<subsystem xmlns="urn:jboss:domain:undertow:3.1">
<buffer-cache name="default"/>
<server name="default-server">

<https-listener name="https" socket-binding="https" security-realm="ApplicationRealm"/>

<http-listener name="default" max-parameters="10000" socket-binding="http" />

<host name="default-host" alias="localhost">
<location name="/" handler="welcome-content"/>
<filter-ref name="server-header"/>
<filter-ref name="x-powered-by-header"/>
</host>
</server>
<servlet-container name="default">
<jsp-config/>
<websockets/>
</servlet-container>
<handlers>
<file name="welcome-content" path="${jboss.home.dir}/welcome-content"/>
</handlers>
<filters>
<response-header name="server-header" header-name="Server" header-value="JBoss-EAP/7"/>
<response-header name="x-powered-by-header" header-name="X-Powered-By" header-value="Undertow/1"/>
</filters>
</subsystem>
```

3. **<server-identities> in ApplicationRealm realm Definition**

   The server identities section of a realm definition, which is used to define how a server appears to the outside world, currently this element can be used to configure a password to be used when establishing a remote outbound connection.

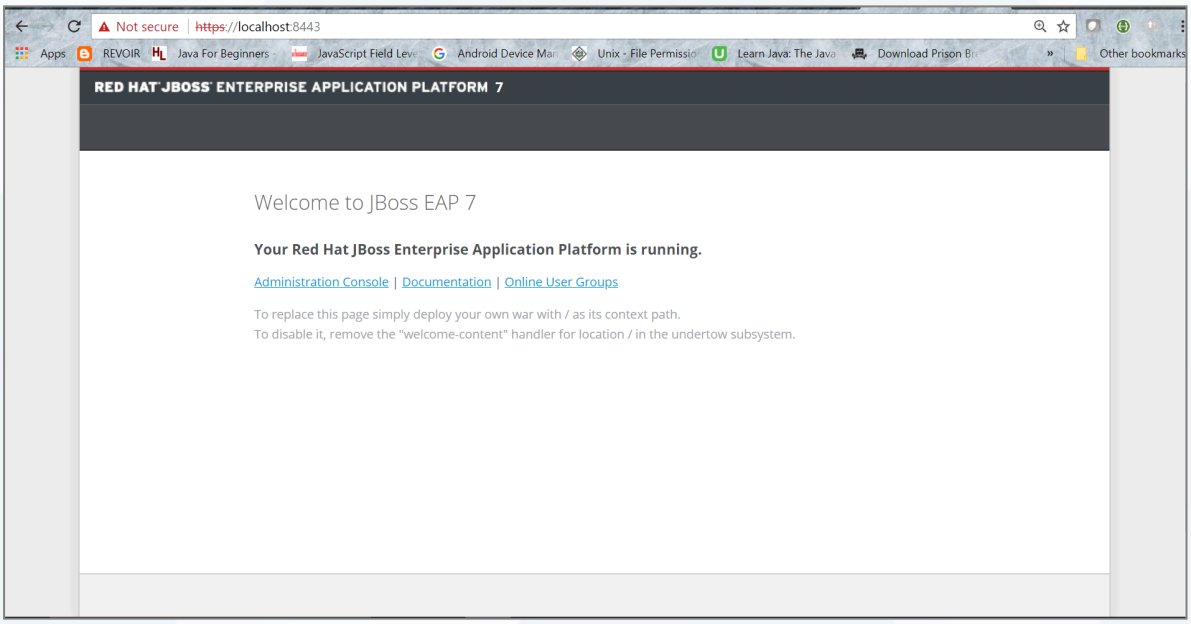
   Add the Highlighted lines:

Standalone.xml

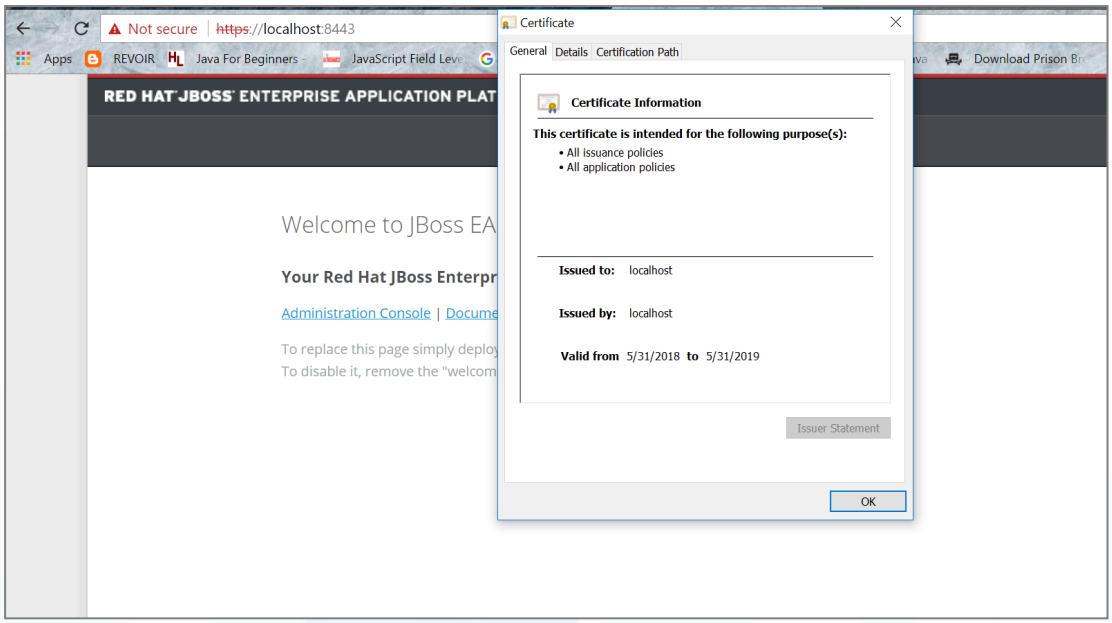
```
<security-realm name="ApplicationRealm">
<server-identities>
<ssl>
<keystore path="temenos.keystore" relative-to="jboss.server.config.dir" keystore-password="temenos" alias="temenos"
key-password="temenos" />
</ssl>
</server-identities>
<authentication>
<local default-user="$local" allowed-users="*" skip-group-loading="true"/>
<properties path="application-users.properties" relative-to="jboss.server.config.dir"/>
</authentication>
<authorization>
<properties path="application-roles.properties" relative-to="jboss.server.config.dir"/>
</authorization>
</security-realm>
```

Once this is done, startup you jboss, by default jboss https runs on port 8443, you can configure as per your wish.

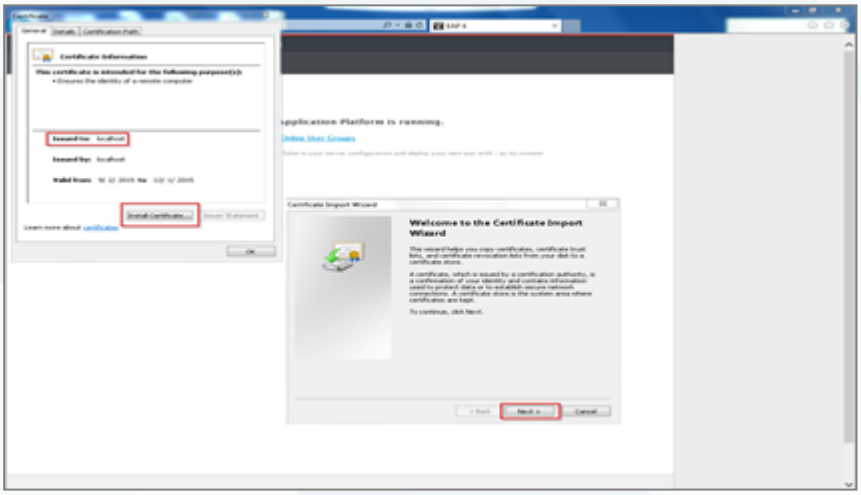
Ensure your jboss is up and running.Try to access https://localhost:8443/ , if you see the below page, its configured properly and listening on configured port 8443.



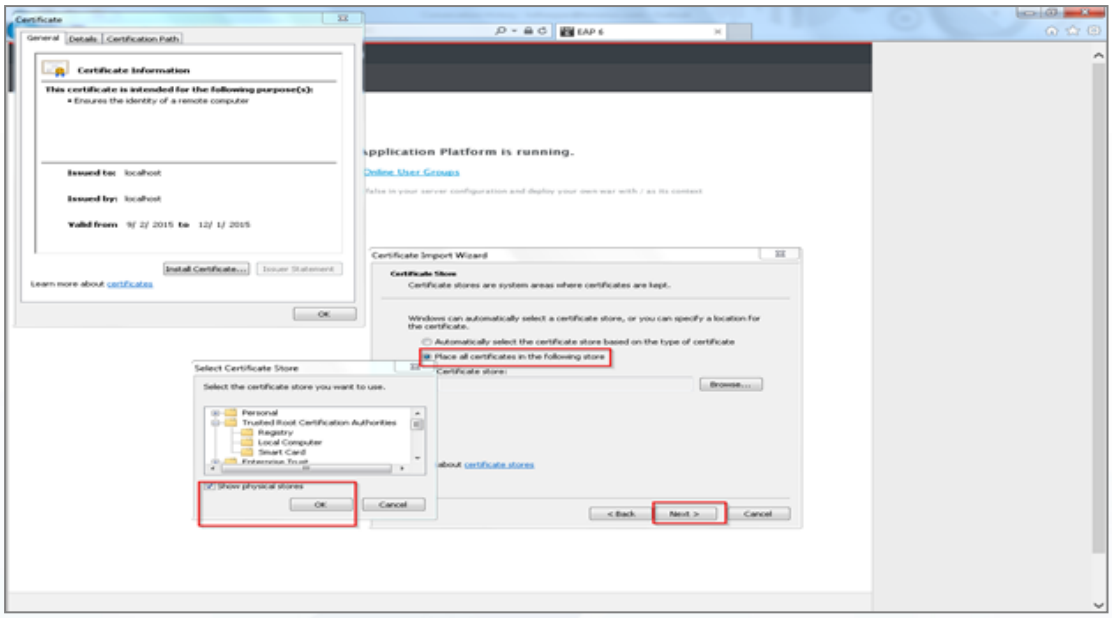
4. **Step 4: Certificate Installation in Browser**
    - In this step, you need to install the self-signed certificate as per the screen shots below.



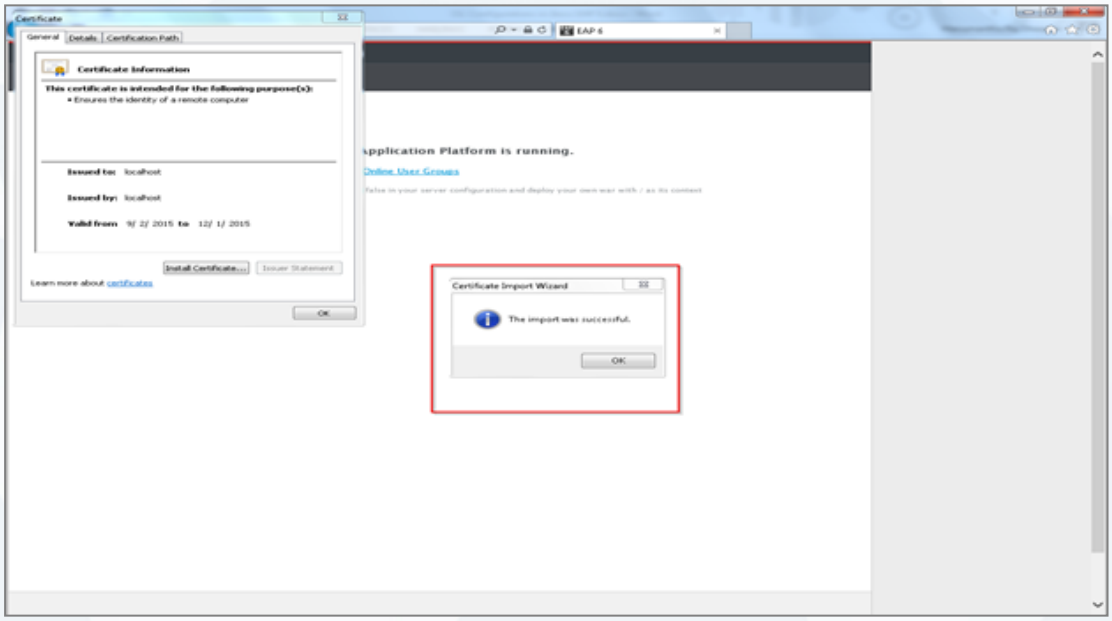
- Certificate Installation Wizard



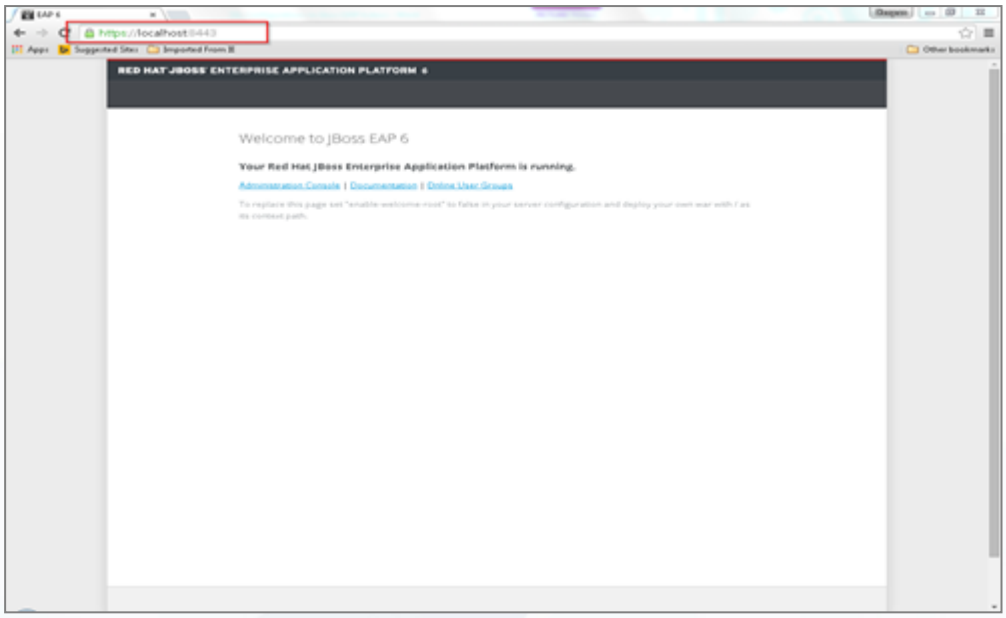
- Provide the path to import the self-signed certificate



- Certificate imported successfully



- Now the SSL works fine with HTTPS



Note: The name ‘localhost’ should be same with the Common Name (CN) provided while creating a keystore and truststore.

IRIS R18 Dynamic Mocking

Overview

IRIS Mock Services support dynamic mock response generation based on the fields present as a part of API metadata and dynamically generated data based on the local mock dictionary maintained. IRIS Mock Services are responsible to generate random data for the metadata fields based on the domain mock files maintained for the same. If the fields are not present in the specified domain (specific mock file), it generates random data based on the corresponding field datatype.

Design

MockDataMgmtProcessor & MockResponder are the classes responsible for handling and generating random data for a request dynamically at run time.

The **MockDataMgmtProcessor** provides implementation to insert, update, view and delete entries into domain specific mock file maintained at basepath specified as a part of **mockDataMgmtProcessor** bean property.

The **MockResponder** provides implementation to generate mock responses based on metadata at run time , it gets corresponding random data by performing a lookup on the domain specific mock file maintained at basepath specified as a part of **mockResponder** bean property, and generating response schema out of it. If the field is not present in the specified mock domain dictionary file, based on the mentioned datatype, it is expected to generate some random data for it.

Spring Configurations

The **mockDataMgmtProcessor** bean should be properly configured with basePath property set to the base Directory where mock Files folder holding all the domain specific mock files are placed.

```
<bean id="mockDataMgmtProcessor" class="com.temenos.irf.mock.MockDataMgmtProcessor">
   <property name="basePath" value="classpath:mock/"></property>
</bean>
```

The **mockResponder** bean should be properly configured with basePath property set to the base Directory where mockFiles folder holding all the domain specific mock files are placed.

```
<bean id="mockResponder" class="com.temenos.irf.core.MockResponder">
   <property name="basePath" value="classpath:mock"></property>
</bean>
```

Services for Domain Specific Mock Dictionary

There is a provision to perform various operations such as to insert, update, view and delete new entries into the maintained mock dictionary via various services. Corresponding Swagger schema is also published in reference to that.

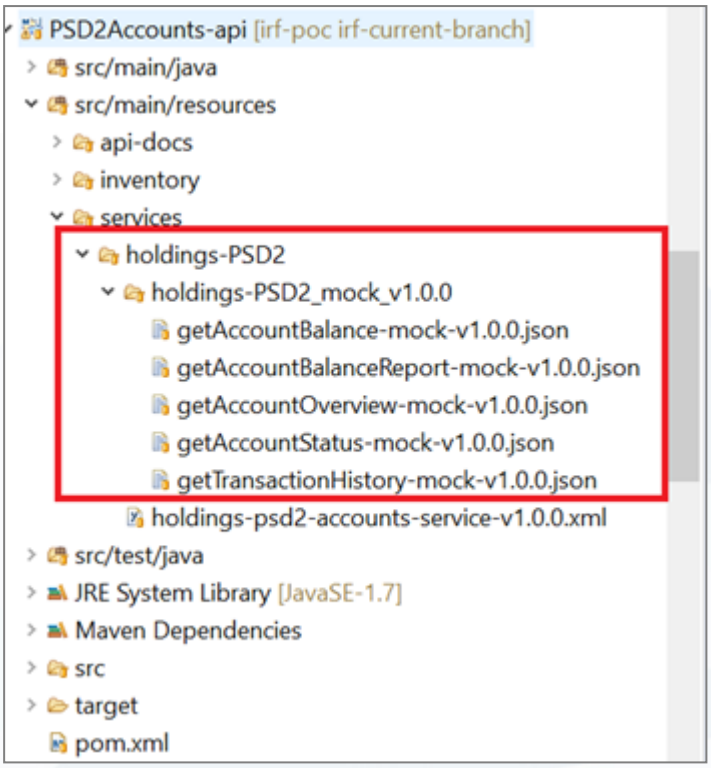
Note: If you are referring to the mock dictionary from the classpath, then you will not be allowed to add, update or delete any of the mock dictionary files from these services, only view access are permitted.

```
<rest path="/v1.0.0/meta/mock/" produces="application/json">
    <post uri="/{domainId}/entries/" id="createEntry">
        <param dataType="string" name="domainId" type="path"/>
        <to uri="direct:mock.create.entry"/>
    </post>
    <put uri="/{domainId}/entries/{entryId}" id="updateEntry">
        <param dataType="string" name="domainId" type="path"/>
        <to uri="direct:mock.update.entry"/>
    </put>
    <get uri="/{domainId}/entries/{entryId}" id="getEntry">
        <param dataType="string" name="domainId" type="path"/>
        <to uri="direct:mock.get.entry"/>
    </get>
    <delete uri="/{domainId}/entries/{entryId}" id="deleteEntry">
        <param dataType="string" name="domainId" type="path"/>
        <to uri="direct:mock.delete.entry"/>
    </delete>
</rest>
```

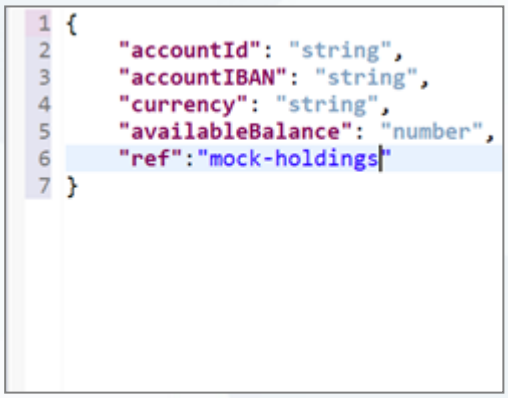
Understanding MockResponder Service

The **MockResponder** provides implementation to generate mock responses based on metadata at run time, it gets corresponding random data by performing a lookup on the domain specific mock file maintained at basepath specified as a part of **mockResponder** bean property, and generating response schema out of it. If the field is not present in the specified mock domain dictionary file, based on the mentioned datatype, it is expected to generate some random data for it.

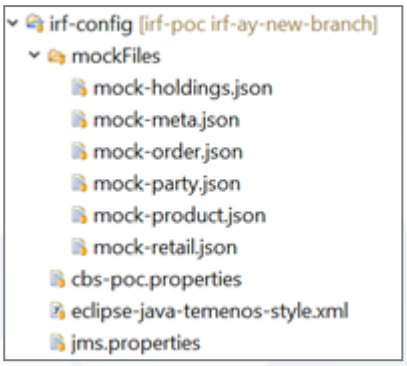
As a part of APIs created from the Workbench ,you will have a mock json file corresponding to the service generated in a mock directory inside the service directory as shown below:



1. Each of the generated mock json contains all the metadata fields along with corresponding data-types.



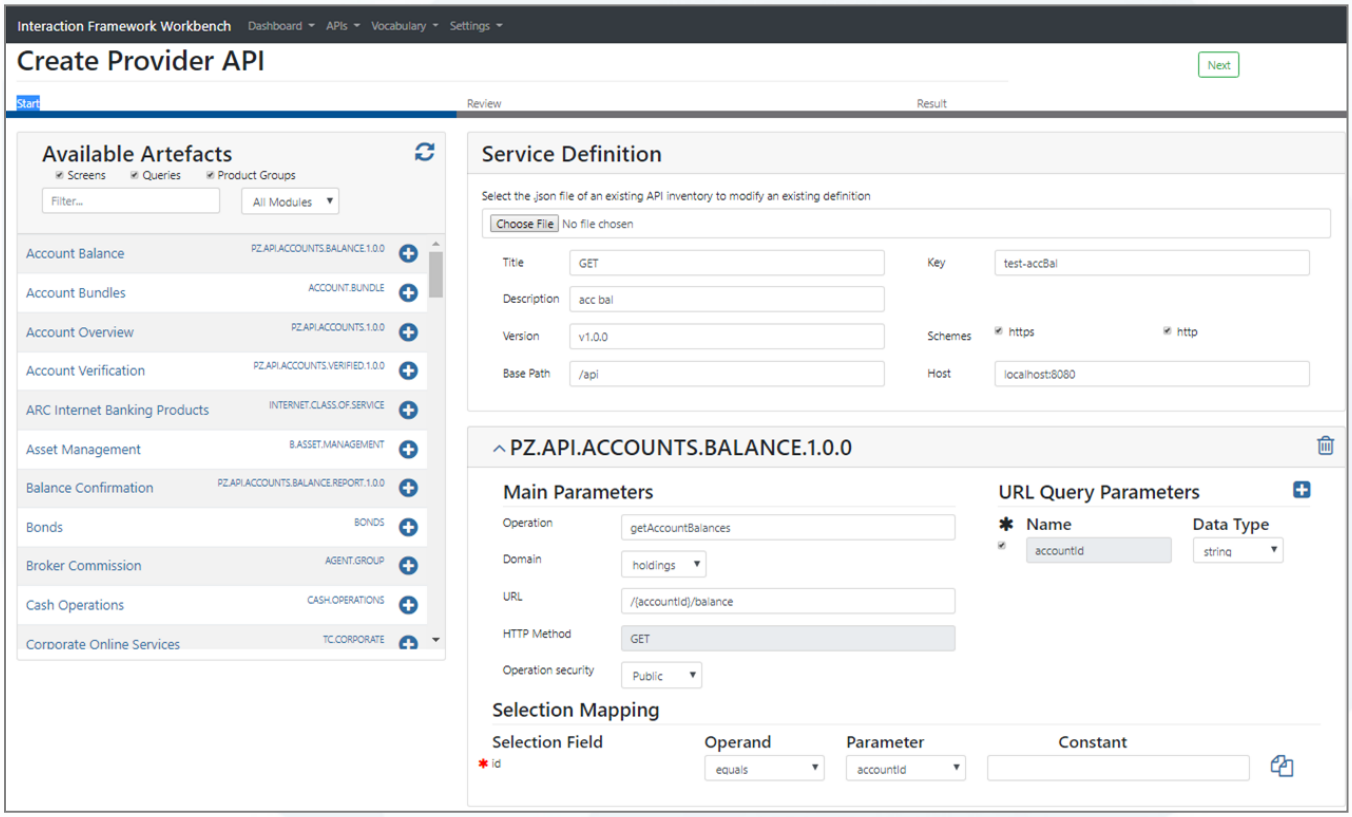
2. Along with this, there is a unique field “ref” along with the mapped domain specific mock file, created during generation itself as shown in above figure.
3. During Mocking the Request, IRIS R18 looks into the above file mentioned in the “ref” of the mock json to get random data from already maintained domain specific mock file.
4. If “**ref**” is not present in the json, it just generates the response same as in the generated mock json.



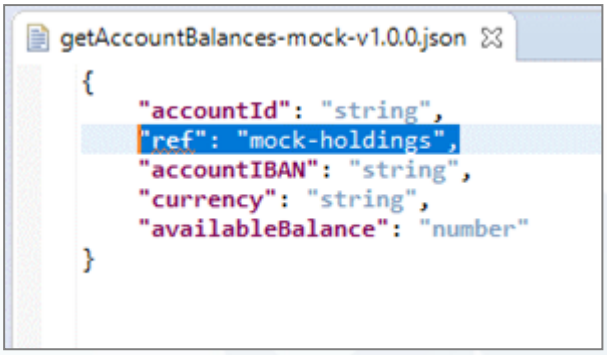
5. It perfoms a lookup in the mock dictionary files based on the domain in the generated mock json and get random values for all the fields in the mock json.
6. If not present, it randomly generates values based on the datatypes of the fields mentioned in the generated mock json.

Steps you need to Follow to Mock your Service

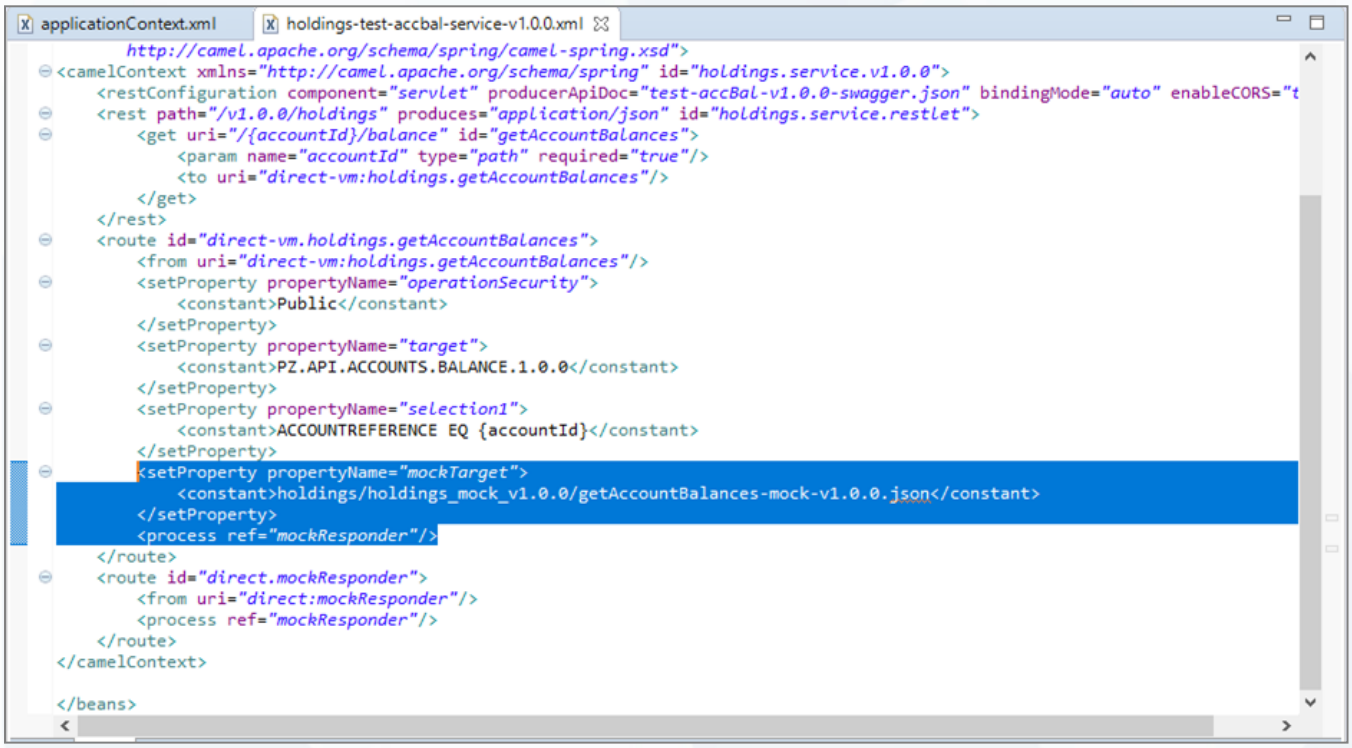
Step 1 : Create a new service



Step 2 : Once Service is generated, Check the generated mock file for "ref" in the json is proper or not.



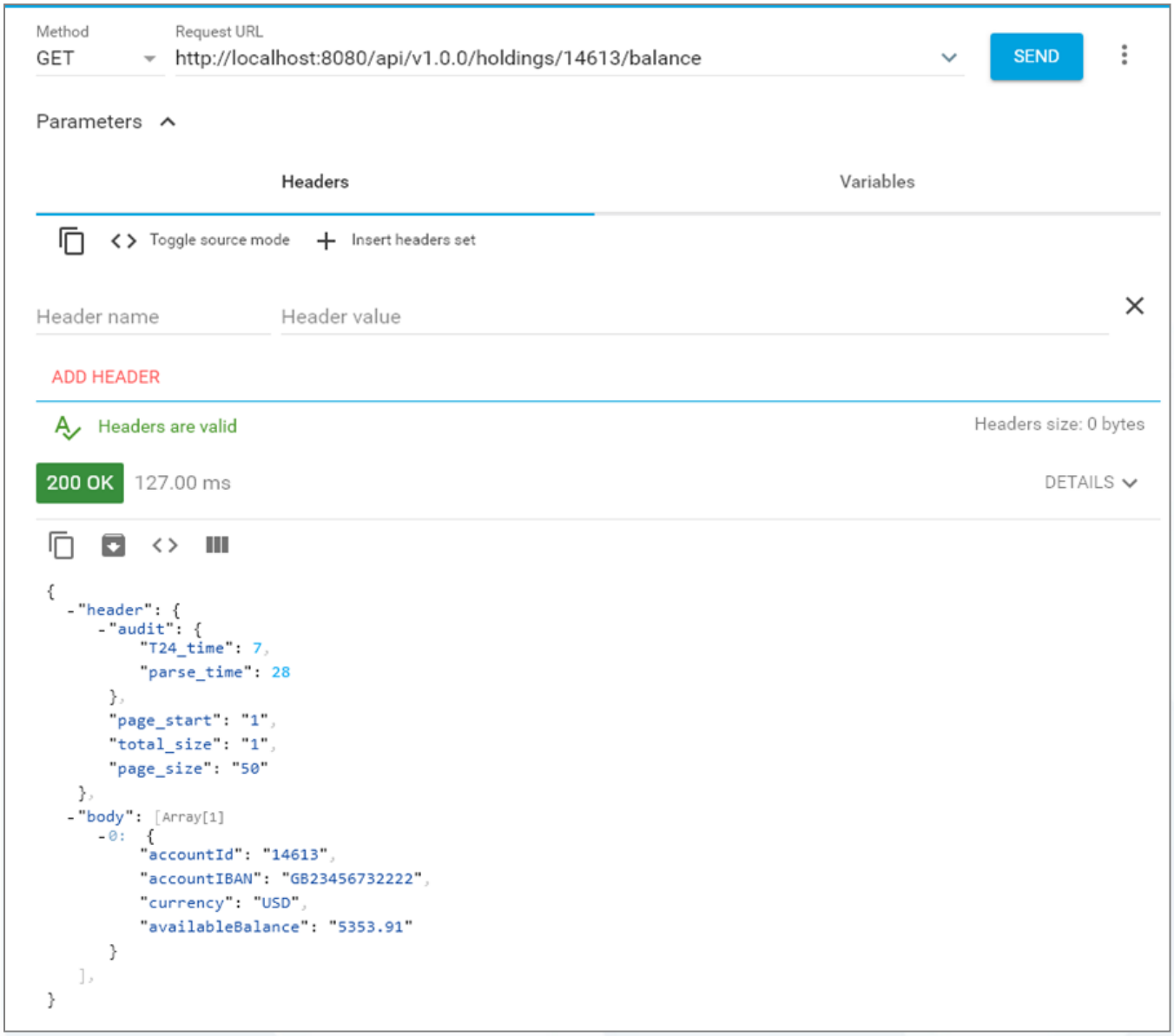
Step 3 : Modify the service xml, configure the property to set "mockTarget" pointing it to the generated mock file for reference to fetch metadata and "ref" for dictionary reference at run-time.



Step 4 : Ensure that the mockResponder and mockDataMgmtProcessor beans are configured properly in ApplicationContext.xml so that the mock dictionary can be reference to generate mock Data.

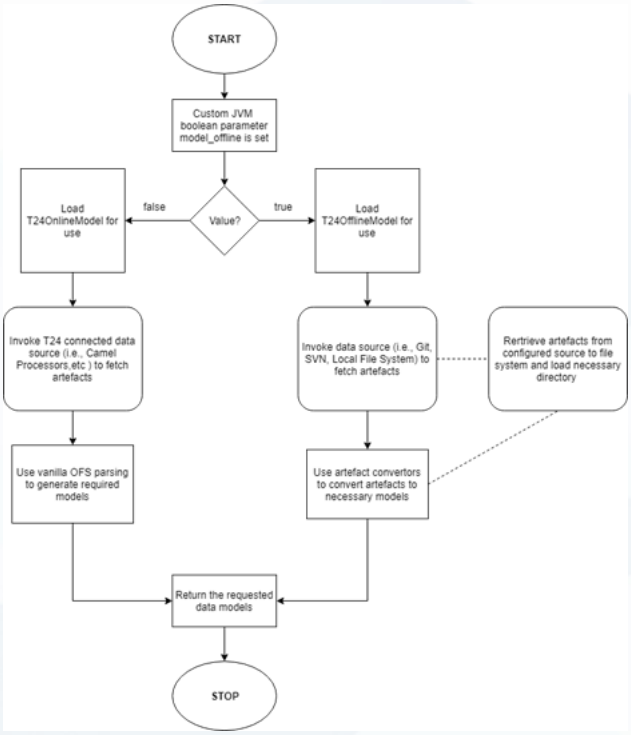
```
21
22      <!-- Comment the above bean with id t24SecurityFilter and uncomment below bean for Security Filter  -->
23      <!--<bean id="t24SecurityFilter" class="com.temenos.irf.comms.security.defaultimpl.T24Security">
24      </bean>-->
25
26       <bean id="serviceLocatorProperties" class="com.temenos.irf.config.StandardPropertyReader">
27          <property name="path" value="classpath:/irf-config/service-locator.properties" />
28      </bean>
29
30      <bean id="mockDataMgmtProcessor" class="com.temenos.irf.mock.MockDataMgmtProcessor" >
31      <!-- basePath should be set to the base Directory where mockFiles folder is placed. -->
32          <property name="basePath" value="C:\docs\irf-config"></property>
33      </bean>
34
35      <bean id="mockResponder" class="com.temenos.irf.core.MockResponder">
36      <!-- basePath should be set to the base Directory where mockFiles folder is placed.  -->
37           <property name="basePath" value="C:\docs\irf-config"></property>
38      </bean>
39
40      <bean id="vocabProcessor" class="com.temenos.irf.vocab.VocabProcessor" >
41          <property name="basePath" value="classpath:/irf-config/"></property>
42          <!-- Comment the above property if you need to point to some external folder and uncomment below one
43          <!-- <property name="basePath" value="../irf-config"></property> -->
44      </bean>
45
46
47      <import resource="classpath*:/services/**/*-service-v*.*.*.xml" />
48      </beans>
49
```

Step 5 : Add the service dependency in the container project, start the jetty and you are good to go. Test your Service.



T24 Model Generation

Flow of Model Generation (for VERSIONS and ENQUIRIES)



By default, if the **JVM parameter** *model_offline* is not present in the system, then online models are loaded, i.e., the system tries to connect to T24 to fetch the model metadata.

For the T24 offline mode, a bean called **sourceAdapter** needs to be present which can accept any custom adapter inheriting **ISourceAdapter** interface.

How to use?

By default the system goes to the online metadata generation flow if the JVM parameter is not present.

Any class implementing the interface **ISourceAdapter** can be set as the **sourceAdapter** bean.

For example, in the default **sourceAdapter** implementation we use the **GitSourceAdapter**.

```
...
<bean id="sourceAdapter" class="com.temenos.irf.git.source.adapter.GitSourceAdapter" />


<bean id="gitConfiguration" class="com.temenos.irf.config.StandardPropertyReader">
<property name="path" value="classpath:/irf-config/git-dev-config.properties" />
</bean>
```

. . .

The **gitConfiguration** bean is a bean dependency for **GitSourceAdapter**.

The following folder structure **should** be followed:

loaded file directory

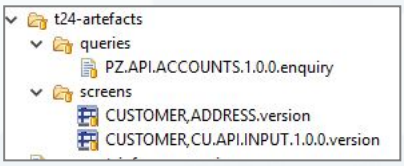
|---------->t24-artefacts

|----------> queries

|----------> AC.API.ACCOUNT.1.0.0.enquiry

|----------> screens

|----------> CUSTOMER,ST.API.CREATE.1.0.0.version



Current git implementation

The current example for Git repository support for offline metadata generation uses a configuration file called *git-dev-config.properties* present in the container irf-config folder.

```
remote=https://github.com/temenosdummyremote/test-git.git
branch=master
userName=temenosdummyremote
password=TemenosIndia123
directory=../../tmp/git/
```

The current Git source service clones the repository and loads the directory path for the IRIS ModelFactory to use the .version/.enquiry files to generate the service API. It also supports 3 git URLs for use

- /v1.0.0/git/clone - Clones the git repository
- /v1.0.0/git/pull - Pulls the configured branch of the repository
- /v1.0.0/git/load - Loads the configured local git repositoryfor use
- /v1.0.0/git/dir - Gives back the directory path for the configured git repository

Use code for other source implementations

Any class which implements the ISourceAdapter should implement the getDir() method. This method returns the path to the files in the system. This class is also responsible for loading the remote files on the local disk so as to use it to find the files.

Infrastructure Services

This section covers the following topics.

Directory Services

Directory based service locator is one of the three, lookup implementation for dynamically listing the available APIs and displaying the swagger spec. This service locator implementation is available in irf-service-locator and configurable in Application Context xml file as properties of processor ServiceDirectoryProcessor. IRIS R18 APIs adopts Open API Specification and defines schema for all services supported.

Listing the services

Users can discover the services supported using the discovery url. Below url helps users to get the swagger documentation url for all the services supported in T24 deployment.

http:<base url>/api/v1.0.0/meta/apidocs?

Where base url will be used to access IRIS R18 application.

Example: http://mybank.com/IRIS-Web/api/v1.0.0/meta/apidocs ?

Sample Response:


```
{
  "header":{},
  "body": [
    {
      "title": "Payments Provider API's'",
      "key": "payments-1.0.0",
      "url": "http://localhost:9089/irf-test-web/api/v1.0.0/meta/apidocs/
payments-1.0.0"
    },
    {
      "title": "Accounts Provider API's'",
      "key": "accounts-1.0.0",
      "url": "http://localhost:9089/irf-test-web/api/v1.0.0/meta/apidocs/
accounts-1.0.0"
    }
 ]
}
```


Swagger api document url for Account APIs

Swagger api document url for Service Catalogue APIs

Additionally users can also filter the services by using below query parameters:

- resource- used to filter based on the resource name.

  Example: resource=”account-info”

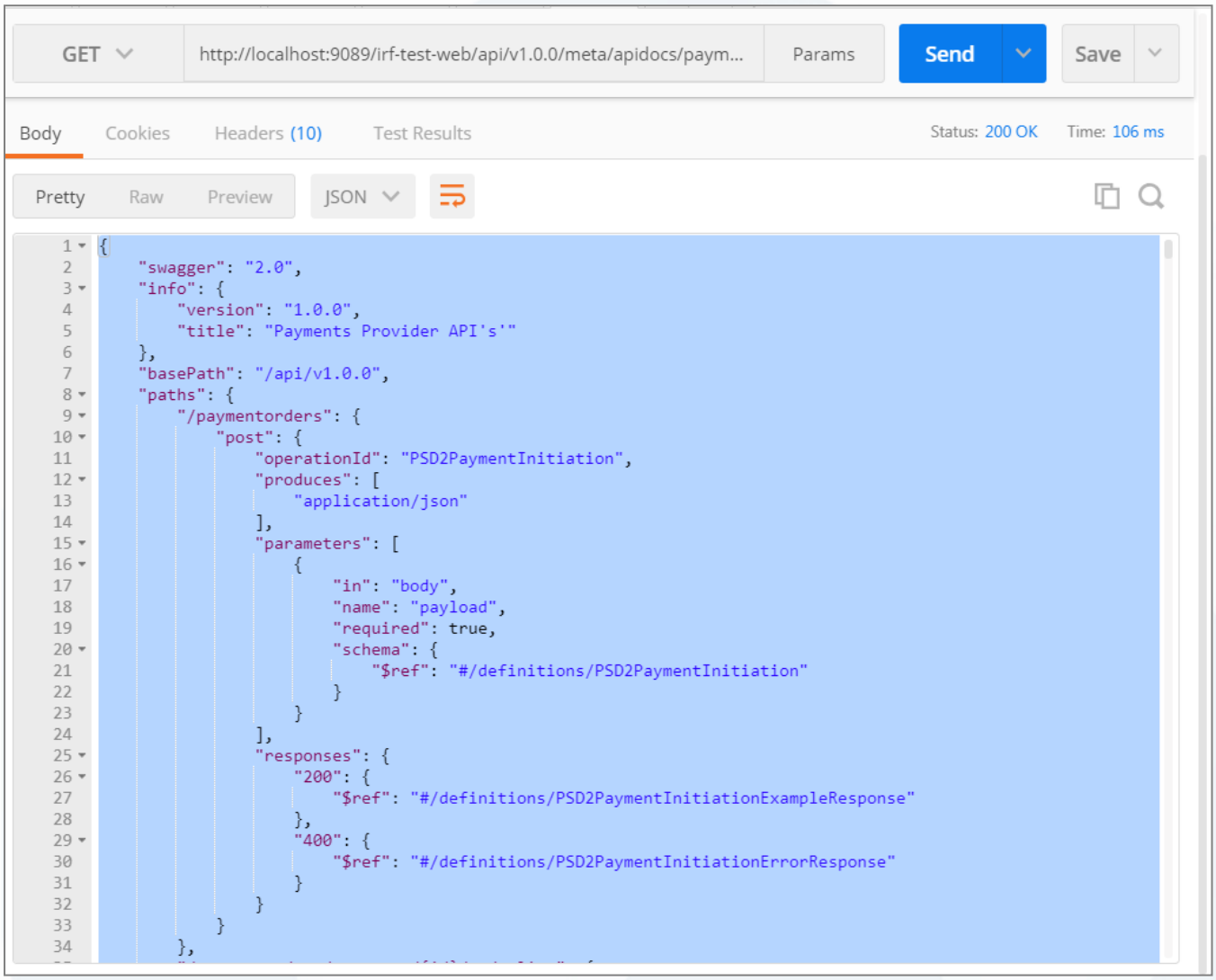
- tag- used to filter based on the tags specified in the swagger API specification.

  Example: tag = “accounts”

Swagger Specification

The url available in the response of **API Docs** request can be used to get the swagger api documentation of the provider apis.

"http://localhost:9089/irf-test-web/api/v1.0.0/meta/apidocs/payments-1.0.0"



API Usage Metrics

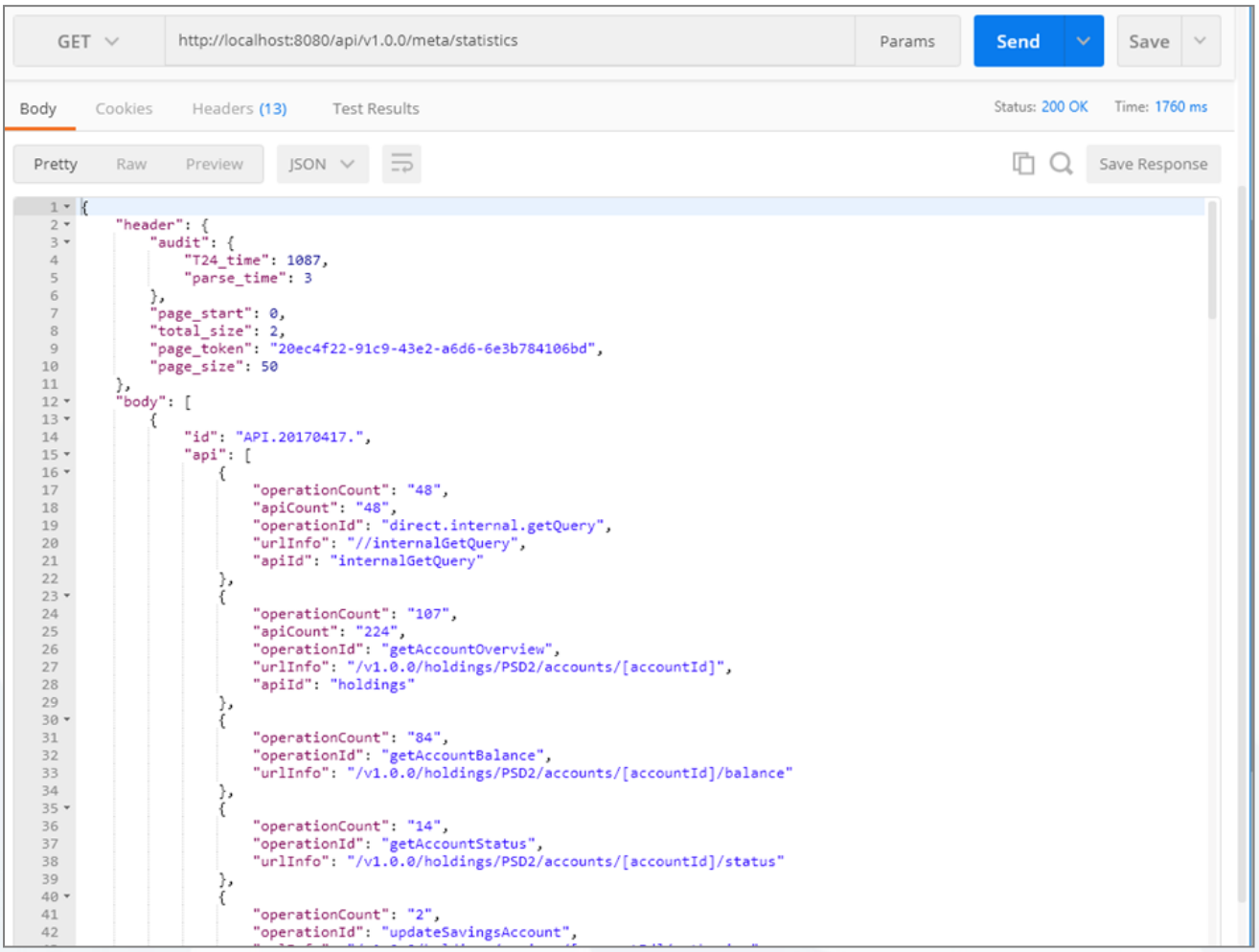
IRIS API Metrics provides an infrastructure to audit the IRIS Request related metrics in Temenos core application. The pre-identified IRIS Service request metrics information along with service information gets captured. Subsequently, those collected metrics gets recorded in T24 Application. IRIS Metrics Recorder component and Metrics Flusher component of IRIS helps to capture and record the iris metrics.

The current API Metrics report can be generated using the below IRIS API.

http:<**base url**>/api/v1.0.0/meta/metrics? Where base URL is used to access the IRIS R18 application.

Example: http://mybank.com/IRIS-Web/api/v1.0.0/meta/metrics ?

A sample response of Metrics is depicted below.



You can get the statistics using **apiId** (Sample url: http://-localhost:8080/api/v1.0.0/meta/statistics?apiId=holdings).

You can also get the statistics using **operationId** (Sample url: http://-localhost:8080/api/v1.0.0/meta/statistics?operationId=getAccountOverview).

Advanced Configurations

SimpleCountMetricFlushStrategy is the default strategy configured to flush the API metrics to T24. This strategy is a simple count based metric flush strategy, flushes the metrics based on the maximum limit configured.

Maximum limit parameter(**metricsFlushCounter.limit**) is used to flush metrics which is made configurable as a JVM parameter. The default count is 1000.

Custom Data Type Support

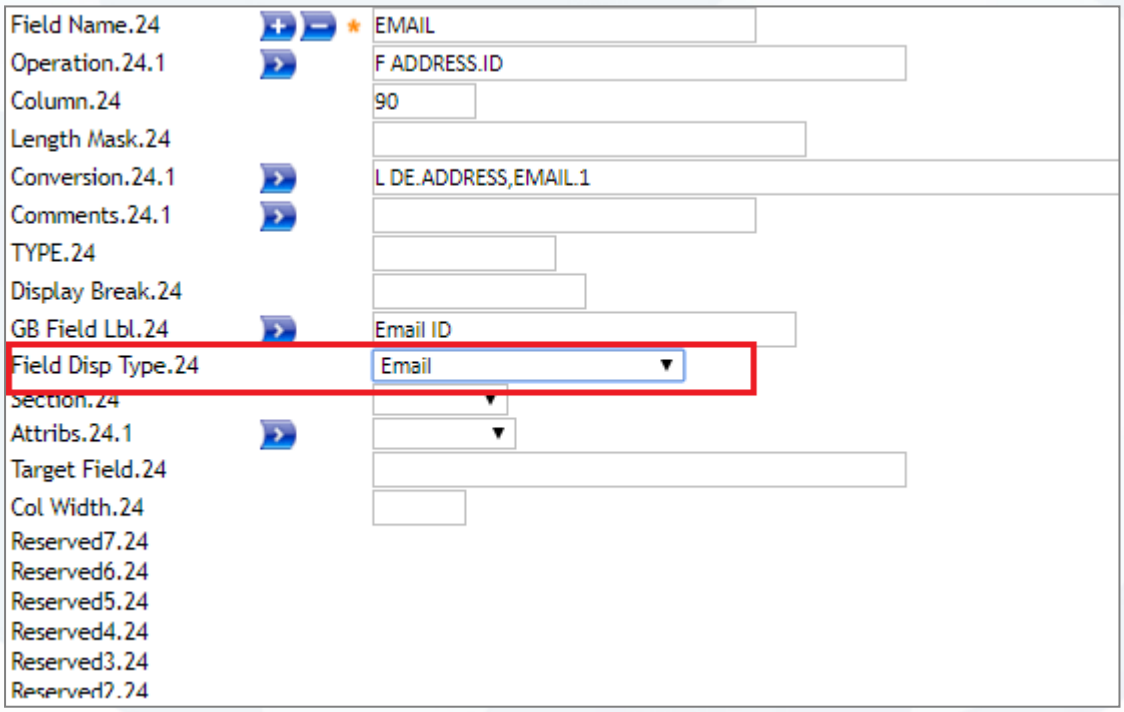
Open API Specification - Support for Format & Pattern

OpenAPI Specification allows you to define following special data types to provide a hint to the API consumer about the data format of the data fields.

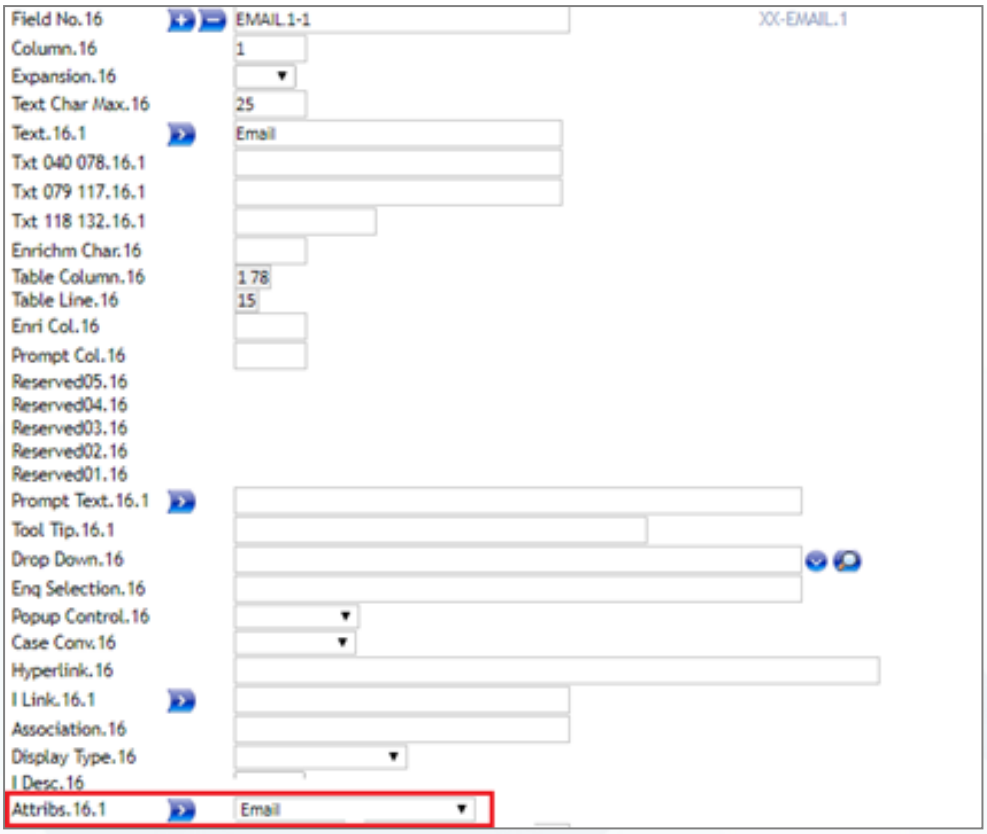
- email
- uuid
- uri
- hostname
- ipv4
- ipv6
- and others

IRIS allows Provider API documentation to support above data types and specify as "format" field in the Open API specification documentation published for IRIS Provider APIs. To support that, Respective field definition in T24 Artifacts needs to be modified to map to required custom attribute data type.

For example, below image "email" attribute data type is mapped to the EMAIL data field in an Enquiry and Version.



in **Version**



To extend support for custom data types, "**datatypes.properties**" in IRF-CONFIG folder needs to be configured as mentioned below,

```
#Attribs/FieldDispType field value = format in swagger
email = email
amount = ^\\d+(\\.\\d{1,2})?
```

The above sample entries in **datatypes.properties** contain the mapping between Attribute data type supported in the respective Core System(example **T24**) and the Swagger data format associated with a specific field in the request or response of IRIS API.

The swagger documentation for supporting format are as follows:

Similarly, to support regex pattern for existing T24 Attribute data type defined, regex pattern can be mapped in **datatypes.properties**

For Patterns,

Understanding Pagination

Pagination in APIs helps client application to handle responses in a easier way.

Pagination is controlled via three parameters **page_size** , **page_start** and **page_token**

- **page-size** - limits the number of records in the api response.
- **page-start** - helps to retrieve next set of records.
- **page-token** - unique number extracted to get as part of response from server on every successful request, which is use to get the consecutive pages.

All three parameters are optional and if not passed, default value is considered. Default value for page-start is 1 i.e. by default response displays 1st page, default value for page-size is 99 which is configurable. On each successful request, server returns an unique token number. Using the same token number, user can get consecutive pages. In this case, no need to send any page size.

```
localhost:8080/api/v1.0.0/provider-accounts/10944/transactions?fromDate=2017-01-01&page_
size=5
```


```
{

"header": {

"data": {

"accountId": "10944",

"currency": "USD"

},

"audit": {

"T24_time": 1554,

"parse_time": 4

},

"page_start": 1,

"page_token": "123456789.02",

"total_size": 39,

"page_size": 5

},

"body": [

{
```

```
"reference": "FT170803R9LY",
"amount": 200000,
"bookingDate": "2017-03-21",
"valueDate": "2017-03-21"
},
{
"reference": "FT17080GCFJJ",
"amount": 150000,
"bookingDate": "2017-03-21",
"valueDate": "2017-03-21"
},
{
"reference": "SCTRSC17080KMQW4",
"amount": -2805,
"bookingDate": "2017-03-21",
"valueDate": "2017-03-21"
},
{
"reference": "SCTRSC1708064H4D",
"amount": -70434.35,
"bookingDate": "2017-03-21",
"valueDate": "2017-03-21"
},
{
"reference": "SCTRSC17086RZV02",
"amount": -3263.27,
"bookingDate": "2017-03-27",
"valueDate": "2017-03-27"
}
]
```

```
}
```

```
localhost:8080/api/v1.0.0/provider-accounts/10944/transactions?fromDate=2017-01-01&page_start-
t=2&page_token=123456789.02
```


```
{

"header": {

"data": {

"accountId": "10944",

"currency": "USD"

},

"audit": {

"T24_time": 448,

"parse_time": 1

},

"page_start": 2,

"page_token": "123456789.02",

"total_size": 39,

"page_size": 5

},

"body": [

{

"reference": "FT17086WV7YX",

"amount": 200000,

"bookingDate": "2017-03-27",

"valueDate": "2017-03-27"

},

{

"reference": "SCTRSC17088S0BZW",

"amount": -22792.77,
```

```
"bookingDate": "2017-03-29",
"valueDate": "2017-03-29"
},
{
"reference": "FT1708899Y3D",
"amount": 75000,
"bookingDate": "2017-03-29",
"valueDate": "2017-03-29"
},
{
"reference": "SCTRSC17088RFNT2",
"amount": -6928.71,
"bookingDate": "2017-03-29",
"valueDate": "2017-03-29"
},
{
"reference": "SCTRSC17088ZP11W",
"amount": -49171.54,
"bookingDate": "2017-03-29",
"valueDate": "2017-03-29"
}
]
}
```

Additional Features and Support

This section covers the following topic.

Multiple Version Response Support

This is a special handling of Version based APIs where a version is expected to return more than one records which is usually the case for a single request sent. Such Scenarios might happen because of some routines attached at the T24 level to the target versions on which the APIs has been developed, this internally creates another record onto another or same table.

To handle such scenarios, IRIS R18 provides support to get all those responses into JSON structure and display them as a json array.

> Note: IRIS R18 Design Time doesn't support this out of the box , the generated service is expected to be manually updated for this feature. Also, the swagger needs to be modified to align with the changes.

Necessary Configuration to Enable this Feature

The following changes are made to support the Parsing of Multiple OFS responses from IRIS R18:

For IRIS R18 , to support multiple responses and proper parsing of the OFS responses, there are some manual changes expected in the **service.xml** file generated for the corresponding services.

> Note: The Swagger documentation and service definition needs to be manually updated to support the multiple version response via single request mechanism. It cannot be expected to be generated automatically from the workbench.

Expected Service XML config needs to be changed manually

```
<setProperty propertyName="target">

<constant>PAYMENT.ORDER,GSWW.API.INSTANT.SEPA.1.0.0</constant>

</setProperty>

<setProperty propertyName="responseScreens1">

<constant>PAYMENT.ORDER,PX.API.INITIATE.PAYMENT.1.0.0</constant>

</setProperty>

<setProperty propertyName="responseScreens2">

<constant>CUSTOMER,ST.API.CUSTOMER.INPUT.1.0.0</constant>

</setProperty>
```

And so on properties such as “responseScreens3”,” responseScreens4”, etc. (Note the first OFS response will be the target response itself) need to set in exchange based on how many responses we are expecting,

- Note that the first responseScreen in the Arraylist will be that actual “target” screen set in the Exchange property followed by the “responseScreens” like , “responseScreens1”, ”responseScreens2”, “responseScreens3”, ”responseScreens4”etc & so on. is set.
- Assumption is that all the responseScreens set in the Exchange property have properly numbered suffix and are in the same order in which the corresponding OFS responses for the corresponding screens in the responses, it will be parsed in the same order.
- Only the services where responseScreens other than “target” is set in the exchange property will automatically be eligible for Multiple response parsing.

Expected Sample Responses

- Response in case of success when 4 responses are expected.


```
{
  "header": {
    "transactionStatus": "Live",
    "audit": {
      "T24_time": 373,
      "responseParse_time": 2466,
      "requestParse_time": 1
    },
    "id": "190226",
    "status": "success"
  },
  "body": [
    {
      "Sector": "1001",
      "mobileBanking?": "NULL",
      "internetBanking?": "NULL",
      "fullName": "IRISX",
      "branchName": "GB0010001",
      "Residence": "US",
      "Gender": "MALE",
      "title": "MR",
      "amlCheck": "NULL",
      "Nationality": "US",
      "Industry": "1000",
      "customerStatus": "1",
      "accountOfficer": "1",
      "Target": "999",
      "Language": "1",
      "Mnemonic": "MNEM4",
      "familyName": "IRISXFAMILY",
      "id": "190226",
      "shortName": "IRIS",
      "amlResult": "NULL"
    },
    {
      "dateTime": "1809141754",
      "currNo": "22",
      "coCode": "GB0010001",
      "description": "***Individuals**",
      "riskExpoType": "410",
      "id": "1000",
      "shortName": "NewIndividuals",
      "Authoriser": "82126_INPUTTER_OFS_GCS_190226",
      "Inputter": "82126_INPUTTER__OFS_GCS_190226",
      "deptCode": "1"
    },
    {
      "dateTime": "1809141754",
      "currNo": "23",
      "coCode": "GB0010001",
      "description": "***Individuals**",
      "riskExpoType": "410",
      "id": "1000",
      "shortName": "Individuals",
      "Authoriser": "82126_INPUTTER_OFS_GCS_190226",
      "Inputter": "82126_INPUTTER__OFS_GCS_190226",
      "deptCode": "1"
    }
  ]
}
```


- Response in case of success when any of the responseScreen fails.

```
{
  "header": {
    "audit": {
      "T24_time": 294,
      "responseParse_time": 1825,
      "requestParse_time": 1
    },
    "status": "failed"
  },
  "error": {
    "type": "BUSINESS",
    "errorDetails": [
      {
        "code": "T24-000",
        "message": "INPUT NOT NUMERIC"
      }
    ]
  }
}
```


- Response in case when OFS response received from screens more than configured in service xml.


```
{
  "header": {
    "audit": {
      "T24_time": 364,
      "requestParse_time": 1
    }
  },
  "error": {
    "type": "BUSINESS",
    "errorDetails": [
      {
        "code": "TGVCP-016",
        "message": "Not Sufficients Targets Set for Received Responses"
      }
    ]
  }
}
```


Important Points

- If the responseScreens configured in the service xml is 2 and the ofs response returned from t24 has more than 2 responses , it will throw an error stating “Not enough targets set”.
- If the responseScreens configured in the service xml is more than 2 and the ofs response returned from t24 has is only 2 responses , it will parse and display those 2 records.
- Ordering mentioned in the service for response will be strictly followed, i.e. target followed by “responseScreens1”,” responseScreens2”, “responseScreens3”,etc , .Please ensure OFS response received also is in the same order as configured.
- If any error occurs in the response where multiple response are expected, it will throw an error from only the first encountered error in the OFS, meaning if 3 responses are expected, first from the actual target and other two from respective screens configured and if second response fails, in that case , it will display error response with details as to what is the issue because of which second response received as failure.

- Response Header will display only the id of target screen response, for rest of the child response, id will be embedded in its body itself.

Support for Advanced T24 functions

IRIS Framework needs to support the following advanced T24 Functions:

- Delete
- Authorize
- Reverse
- Validate

In order to set the functions, property needs to be set in the service.xml file. The Property name should be "function" and constant should be the T24 function.

For example,

```
<route id="direct-vm.order-PSD2.deletePayment">
    <from uri="direct-vm:order-PSD2.deletePayment"/>
    <setProperty propertyName="operationSecurity">
        <constant>Public</constant>
    </setProperty>
    <setProperty propertyName="target">
        <constant>FUNDS.TRANSFER,FT.API.INITIATE.PAYMENT.1.0.0</cons
    </setProperty>
    <setProperty propertyName="recordid">
        <header>paymentId</header>
    </setProperty>
    <setProperty propertyName="function">
        <constant>delete</constant>
    </setProperty>
    <process ref="t24VersionProcessor"/>
</route>
<route id="direct.mockResponder">
    <from uri="direct:mockResponder"/>
    <process ref="mockResponder"/>
</route>
/camelContext>
```

NOTE : By default, Input function is set.

If the httpMethod is **DELETE** and function property is not specified, **reverse** function is set by default.

If the httpMethod is **GET** and function is not specified in service xml, **See** function is set by default.

If the httpMethod is **PATCH**, it throws an error "Patch method not supported".

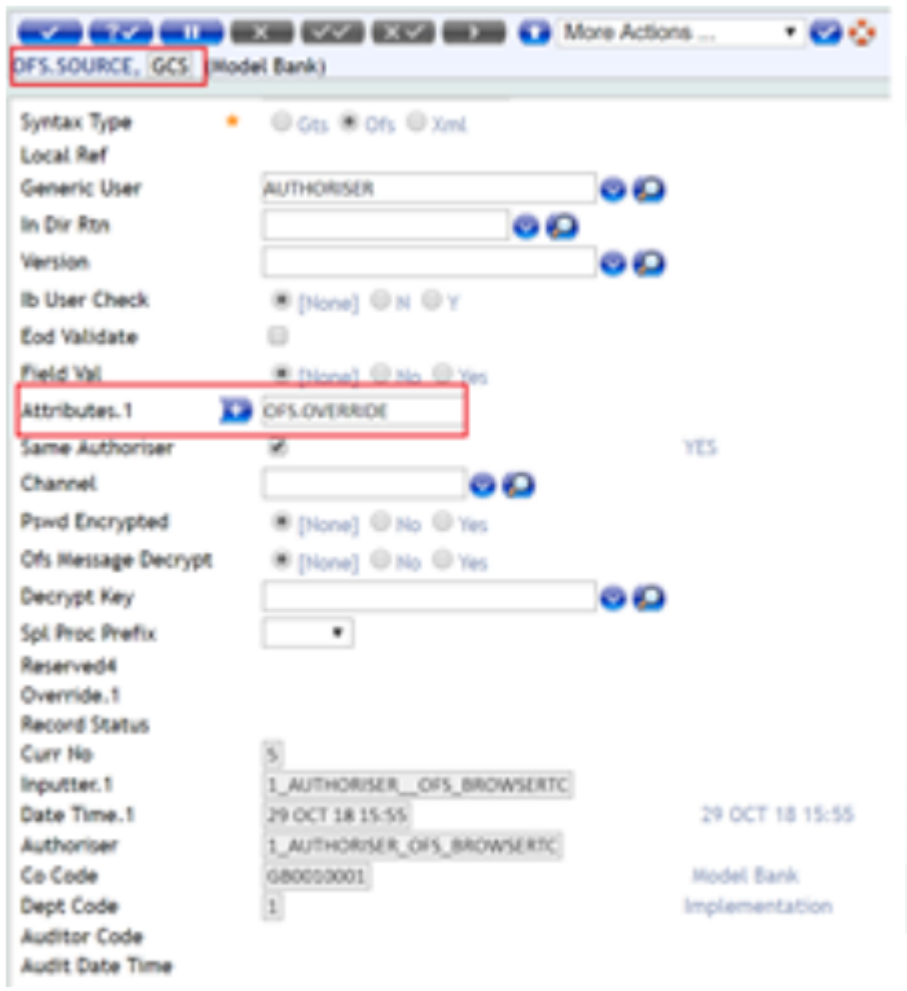
IRIS R18 Overrides/ Warnings Handling

This feature captures all the overrides/ warning thrown by T24 and give a provision for the end users to view those and accept those overrides/ warnings to complete the transaction.

An attribute “OFS.OVERRIDE” is added in the OFS.SOURCE. This enables the OFS response through that OFS Source to appropriately capture all the overrides and return override messages along with its ID so that end users can view and send them back in order to accept those overrides.

> Note: This attribute configuration is applicable only if you wish to use OFS.SOURCES other than IRIS R18 specific OFS.SOURCES (i.e IRISINTERNAL,IRISEXTERNAL,IRISAAINTERNAL,IRISAAEXTERNAL), as this is already pre-configured for these sources as a default setting.

The below figure depicts the setting of OFS.OVERRIDE attribute in the OFS.SOURCE



On encountering any overrides/ warnings from T24, IRIS R18 captures those and create an "override" object and return back with a proper Response format.

The below figure depicts the JSON response in case of any override/ warning encountered.

Accepting Overrides/ Warnings in IRIS R18

IRIS R18 provides a simpler and straight-forward provision to accept the overrides/ warnings returned by T24.

In order to do that, you need to re-send the original JSON request along with the overrides thrown by T24 back to T24 as a part of JSON payload header so as to send it via OFS message request back to T24 to acknowledge and accept those.

You can set overrides/ warnings back in the original request header to accept them as shown below.

```
{
  "header": {
    "override": {
"overrideDetails": [
  {
"overrideId": "NO.WORKING.DAY",
"overrideDescription": "DEBIT VALUE NOT A WORKING DAY"
},
  {
"overrideId": "ACCT.UNAUTH.OD",
"overrideDescription": "Unauthorised overdraft of USD 1676795.96 on account 22098."
}
]
}
  },
  "body": {
    "transactionType": "AC",
    "debitAccount": "22098",
    "debitCurrency": "USD",
    "debitAmount": 100,
    "debitValueDate": "20180415",
    "creditAccount": "22411"
  }
}
```


IRIS R18 retrieves these overrides from the request payload and append it to the OFS Request at the 19th position after company code as T24 expects it back to accept those override and complete the transaction.

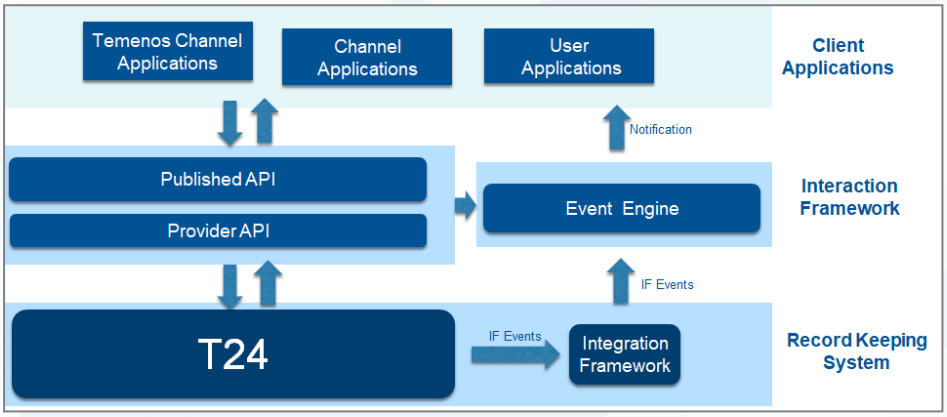
You can view the JSON response on accepting all the corresponding overrides/ warnings thrown, as shown below.


200 OK
3020.72 ms
DETAILS
{
"header": {
"transactionStatus": "Live",
"audit": {
"T24_time": 2958,
"parse_time": 5,
"responseParse_time": 6,
"requestParse_time": 21
},
"id": "FT181077P2XF",
"status": "success"
},
"body": {
"transactionType": "AC",
"debitValueDate": "20180415",
"debitAccount": "22098",
"creditAccount": "22411",
"debitCurrency": "USD",
"debitAmount": "100.00"

IRIS Event Processing

You can provide support for asynchronous API calls (for long running transactions) through an Event processing capability in Interaction Framework.

Event Engine component adds asynchronous requests handling capability in IRIS. The Event engine is an infrastructural component, which accepts the event subscription requests from client application and listens for events raised on completion of asynchronous services from record keeping systems like T24. Once it receives events from T24, Event engine maps your event subscription request and deliver the T24 Event to respective client application.



Prerequisites

1. JAVA Version 8 or above.
2. IRIS binaries should be 201901 or above.
3. Ensure the **irf-event-management-container-<version>.war** is deployed.
4. Ensure the table "**EventsRequests**" has been created in the database, if not, execute the below script in database.

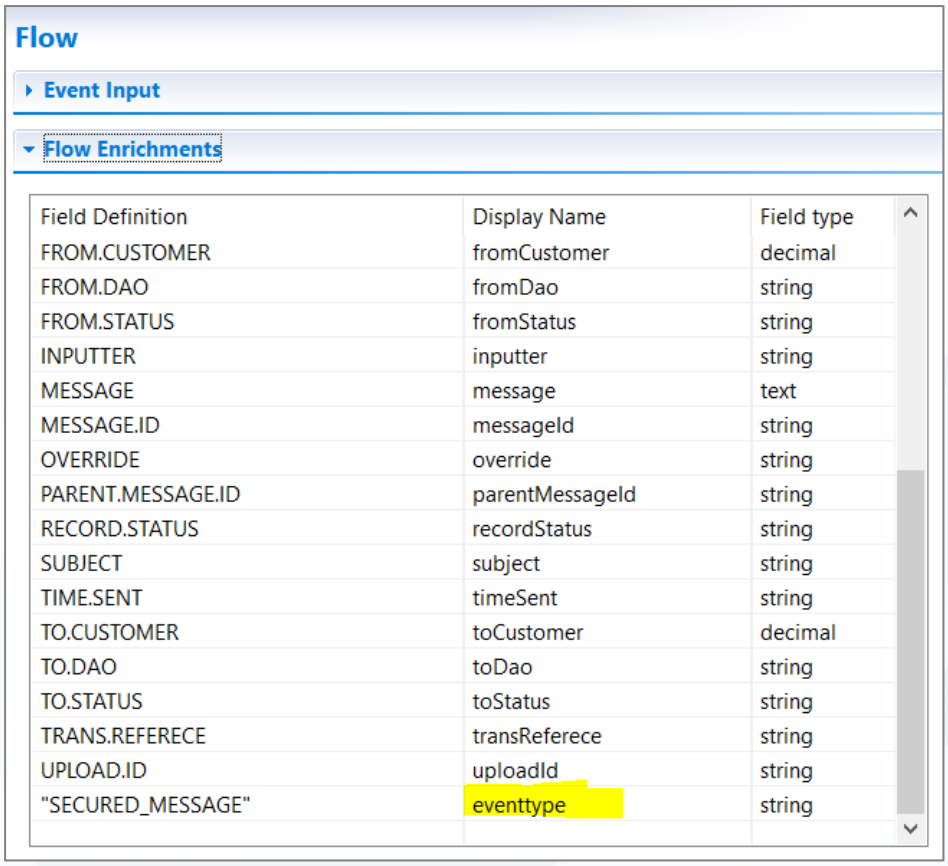
```
create table EventsRequests (requestId bigint not null auto_increment, callbackUrl
varchar(255), CREATION_TS TIMESTAMP DEFAULT CURRENT_TIMESTAMP, UPDATE_TS integer,
eventId varchar(255), eventType varchar(255), expiry integer not null, keyFieldsData
varchar(255), status varchar(255), subscriptionId varchar(255), primary key
(requestId));
```

API Designing /Design Time changes

Event configuration in Integration framework

For IRIS to process the events and send the events back to the consumer, the IF events should be raised with a mandatory property called "**eventtype**". IRIS processes the events, only if, the "eventtype" property is available in the event message, else it ignores all other messages that does not have the "**eventtype**" property.

The below screen print shows the IF event raised with the "**eventtype**" property.



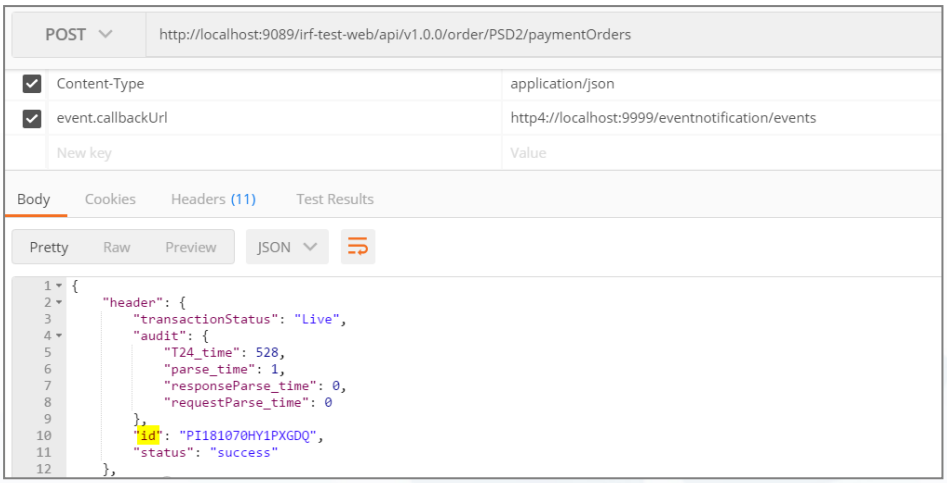
For Version services

For Version services, refer the below table and add the necessary properties in the service accordingly.

S.No	Request Fields	Purpose	Mandatory/Optional
1	event.type	The IF event type for which user interests to get subscribed	Mandatory
2	event.id	Event group id	Optional
3	event.request.keys	The id for which the event matches and needs to be sent to the endpoint.	Mandatory

Note:In the event.request.keys field,

Left side value= "id", this is the field where T24 generated id is displayed by IRIS response, refer the image below for reference.



Right side value = "CstmrPmtStsRpt.OrgnlGrpInfAndSts.OrgnlMsgId" is in the JMS message in which this ID is available.

Screen shot example of service.xml for version service:

For Enquiry services

For Enquiry services, refer the below table and add the necessary properties in the service accordingly.

S.No	Request Fields	Purpose	Mandatory/Optional
1	event.type	The IF event type for which user is interested to get subscribed.	Mandatory
2	event.id	Event group id	Optional

3	event.request.keys	The id for which the message must match	Mandatory
4	event.path.param	The path parameter in the URL for which the event is requested. Eg : uri="/{customerId}/secureMessages" , the {customerId} is the path param for which the event is requested.	Mandatory

Note:In the event.path.param field,

Let's assume the service URL is http://<host:port>/api/v1.0.0/party/customers/{customerId}/secureMessages, then the"{customerId}"is the path parameter in which you can pass the customer id, hence this should be your event.path.paramconstant field.

Screen shot example of service.xmlfor the enquiry service:

```
<setProperty propertyName="event.id">
        <constant>PO_PROCESSED_EVENT</constant>
    </setProperty>
    <setProperty propertyName="event.type">
        <constant>SECURED_MESSAGE</constant>
    </setProperty>
    <setProperty propertyName="event.request.keys">
        <constant>id=toCustomer</constant>
    </setProperty>
     <setProperty propertyName="event.path.param">
        <constant>{customerId}</constant>
    </setProperty>

<process ref="t24EnquiryProcessor"/>
```

Event Subscription / Run time changes

While invoking/consuming the service, IRIS expects you to provide the "**event.callbackUrl**" in the request header, with a valid call back url at which the user is listening for the event message.

Only by providing this header, IRIS considers this request for event processing and the corresponding request is logged in database for the event subscription.

The below table defines the available headers for event request logging.

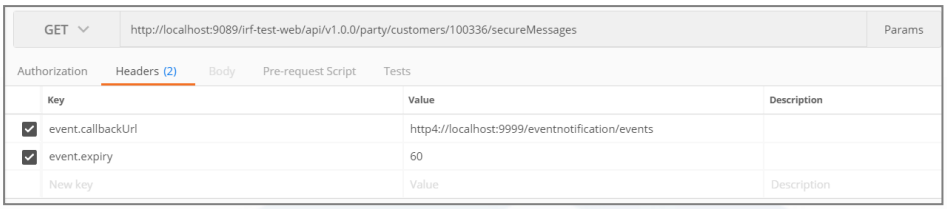
S.No	Request Fields	Purpose	Mandatory/Optional
1	event.callback.url	The endpoint at which the user is listening for the event messages.	Mandatory
2	event.expiry	The expiry period in **MINUTES** till which the event is subscribed.	Optional

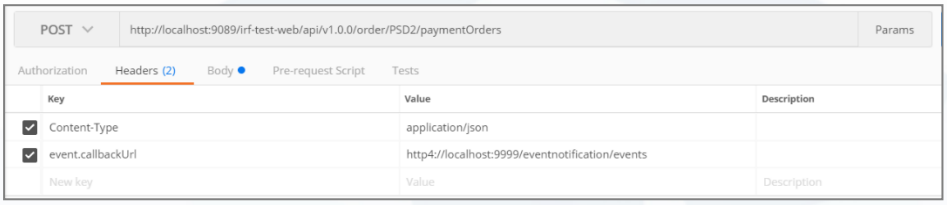
Example:

event.callbackUrl=http4://localhost:9999/eventnotification/events

event.expiry= 20

Screens shot example:





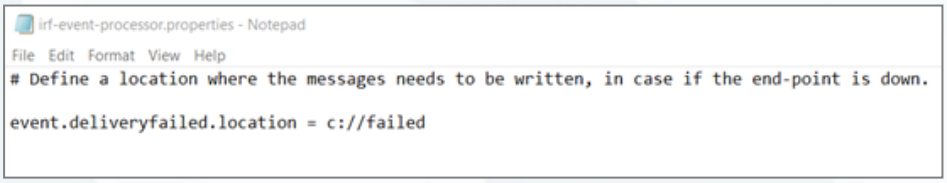
irf-event-management-container.war

This is the iris war file, which does the event processing from IRIS.

Delivery failed Messages

In case, if the client call back url end point is down, then the messages are written in the file, and this location can be configured in a property file.

Called “irf-event-processor.properties”



Data source definition

In case, if you have a different data source other than specified, then update the below,

For the database connectivity, update the data source information if required.

1. Open Spring-persistance-config.xml file in the \WEB-INF\classes.

2. Update the data source.

```
<bean id="dataSource" class="org.springframework.jndi.JndiObjectFactoryBean">
    <property name="jndiName" value="java:/jdbc/t24DS" />
</bean>
```

IRIS Auth token Generation (JWT Token)

The functionality required in IRIS is to provide capability in IRIS to validate the credentials passed in header(basic Authentication) against T24 and send back the JWT token for a the successful authentications.

Prerequisite

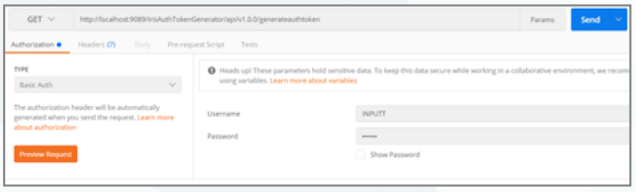
1. JAVA Version 8 or above.
2. IRIS binaries should be 201902 or above.
3. Make sure the **irf-auth-token-generation-container -<version>.war** is deployed.
4. Make sure the routine EB.IRFX.AUTH is available in your T24 environment.

How to Generate the Auth Token ?

Deploy the war file **irf-auth-token-generation-container -<version>.war**, and send a GET request using any rest client, with the basic authentication credential. This authentication credentials is validated in T24 , if the credentials are successful, a JWT token is received as a response.

URL : http://<HOST>:<PORT>/IrisAuthTokenGenerator/api/v1.0.0/generateauthtoken

Request



Response

```
{

"id_token":"eyJh-
bGciOiJSUzI1NiJ9.eyJp-
c3MiOiJURU1FTk9TIi-
wiYXVkI-
joiVDI0VXNl-
ciIsIm-
lhdCI6MTU0NzYyMDUyNCwiZXh-
wIjoxNTQ3NjIyMzI0LCJzdWIiOiJJTlBVVFQifQ.bMgdanLDnDyOE6jgzSHdAr29G765pHvfhJSb4RGHecD3g2E7-
K2W1rYzMmVXW8nIWRu-
J8quLV0P4f3c5yUn0w9un-
RuECNxZo0OKZQi8fo8q5GFrg4v6WPyIA9HhWwwcS7bso0Bc5fIwg2vIzvqNiKUELeCdBguWZqc-
jH3oMw72t41M2PUJHYs_f_W4gaKmpwVY4gl_46HSC1P11iUsZhZrS2k2jsvB1wVEF3Lhcx5kqgwMSSZ38w-
auL94zZeJ59ETg0zwE2dObkfThtIAJWIXHYwJTY9XW4u4waIaz-
jKv4BMmjdJIs82gI47buXQYvuxRmklpjTAP8Sgt6Vzvg",

"token_type":"Bearer"

}
```

How to use the client's own Private key to sign the Token ?

By default, the **irf-auth-token-generation-container -<version>.war** is delivered with built –in **key-storefile (temenos_iris.jks)**, using which the token is signed, in case if the user wanted to create their own key store file to get the private key and sign the token, this can be done in the following ways:

How to Generate the Keystore file ?

Use the following command to generate your own, keystore file. Change the values if required. This command generates the .jks (keystore) file. Make a note of the "alias","storepass",and "keypass" vaues, this needs to be used inside the war.

keytool -genkey -alias temenos -keyalg RSA -keysize 2048 -keystore temenos_iris.jks -dname "CN=localhost,OU=Home,O=Home,L=SL,S=WS,C=LK" -storepass temenos123 -keypass iris123 -validity 20000

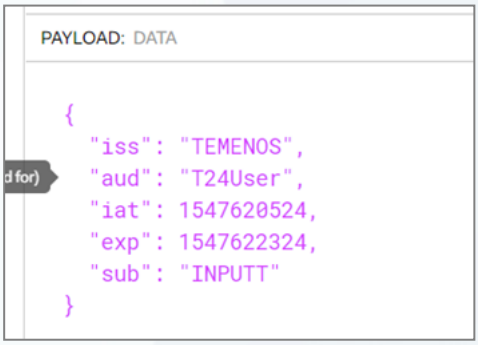
How to use the Keystore file to sign the JWT token?

Copy the above generated (or) your own .jks file in to the class path location of the war file. i.e.,

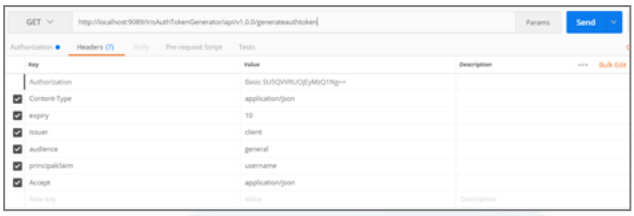
1. Open the war file and go to "**.war\WEB-INF\classes**" and paste the .jsk file here.
2. Open the "keystore.properties" in the same location and update the properties
    i. keystore.file.name = <your keystore file name>
    ii. store.password = <your keystore password>
    iii. key.password = <your key password>
    iv. alias.name = <your alias name>
    v. Save the changes and deploy the war file and test the same.

How to customize the JWT token properties

By default IRIS generates the Standard JWT token with default claim values, refer below image.



In case, if the customer need to change the default behaviour, this can be done by sending the corresponding header in request headers like below.



This results in JWT token response.

PAYLOAD: DATA


```
{
  "iss": "client",
  "aud": "general",
  "iat": 1547619578,
  "exp": 1547620178,
  "username": "INPUTT"
}
```


Additional Info : How to get the .cer file to validate this generated JWT token signature ?

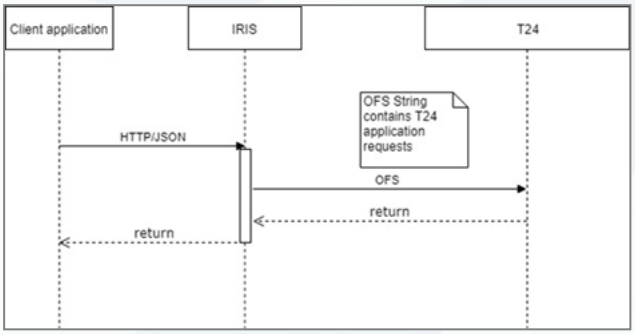
Using the .jks (keystore file), you need to generate the corresponding certificate (.cer) file to get the public key out of it and using this public key the token needs to be validated. Use the below command to generate the .cer file and share this with the user to validate the JWT token while consuming it. Change the names if required.

keytool -export -keystore temenos_iris.jks -alias temenos -file temenos_iris.cer

Supporting Bulk Capability

This feature allows user to use Provider API infrastructure to create or update T24 Application records in T24 under single transactions. This functionality guarantees that all records get committed under a single commitment boundary of T24 Transaction if there are no errors in the data.

Bulk Functionality allows client application to Create or Update date records belonging to a single application using a configured T24 VERSION under single T24 Transaction scope.



Provider Service properties Added

ISBULK Property

The following property is added as a part of providing services to handle bulk requests. For all Bulk request API, “isBulk” property needs to be set as “true”.

```
<setProperty propertyName="isBulk">

<constant>true</constant>

</setProperty>
```

RecordId Property

In case of update scenario, IRIS framework needs to identify the recordId of the individual application records for performing OFS Bulk update. The existing “recordId” property is used to specify the field name of IRIS request payload field.

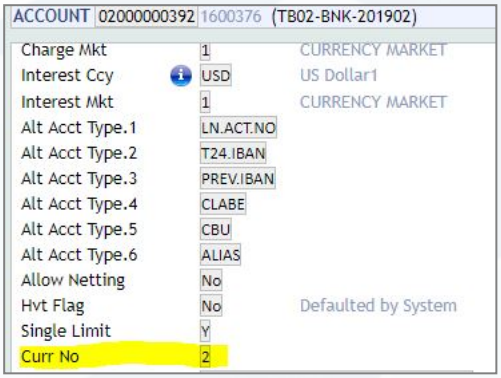
For example, above Bulk Customer update service recordId gets derived based on the ‘customerId' field in the input JSON payload of the request.

```
<setProperty propertyName="recordid">

<constant>customerId</constant>

</setProperty>
```

E-Tag Support

What is E-Tag/Version Number?

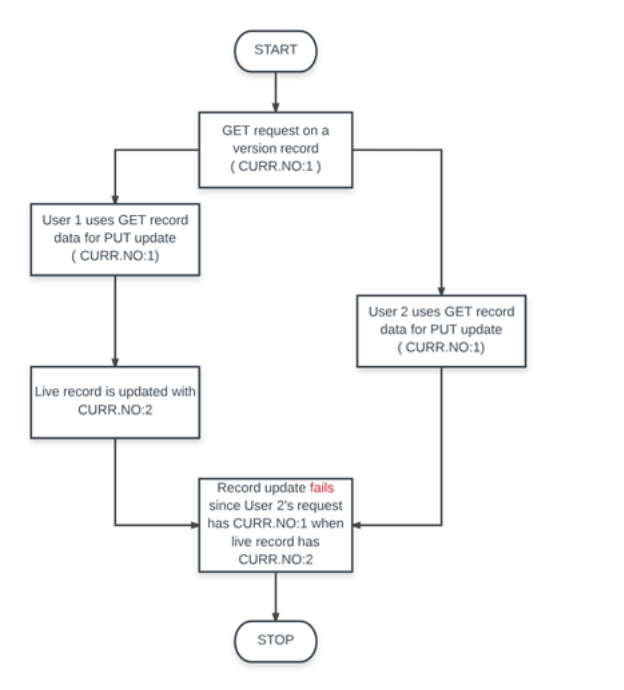
An E-Tag/Version Number is an indication of the the current T24 live record's CURR.NO i.e.; an indication of how many times the live record has been modified.



The above live record has been modified 2 times.

In T24 core, this feature can be used to support an optimistic record locking system while using update IRIS R18 APIs so as to prevent data loss in a single record being updated by multiple users scenarios.

API Flow for Update with E-Tag



How does IRIS R18 handle E-Tag in

Versions API

If an IRIS R18 version does not have the **CURR.NO** field defined in the main/ associated versions, then GET/PUT/POST requests supports a field called "**versionNumber**" present in the "**audit**" object of the "**header**" object in the **JSON**.

Example, for an account GET request:


```
{
"header": {
"transactionStatus": "Live",
"audit": {
"T24_time": 430,
"parse_time": 1,
"responseParse_time": 0,
"requestParse_time": 0,
"versionNumber": "5"
},
"id": "02000000389",
"status": "success"
},
"body": {
"productCode": "6001",
"customerId": "1600281",
"mnemonic": "TEST",
"currencyId": "USD",
"accountNames": [
{
"accountName": "TESTsuccess"
}
]
}
```

```
}
```

This **versionNumber** is directly mapped to the T24 field CURR.NO

To use this feature in PUT/POST requests, the **versionNumber** to be passed should be same as the one returned by the GET request (present in the LIVE record).

If an IRIS R18 version has the **CURR.NO** field defined in the main/ associated versions, the GET request then will have the CURR.NO field as defined in the version, instead of being present explicitly as the versionNumber.

However, the functioning of the PUT/POST request remains the same,i.e.,the versionNumber to be passed should be same as the one returned by the GET request (present in the LIVE record).

Enquiry API

The IRIS R18 enquiry should have the field CURR.NO defined in the enquiry record as a column field so as to display the CURR.NO in the API response.

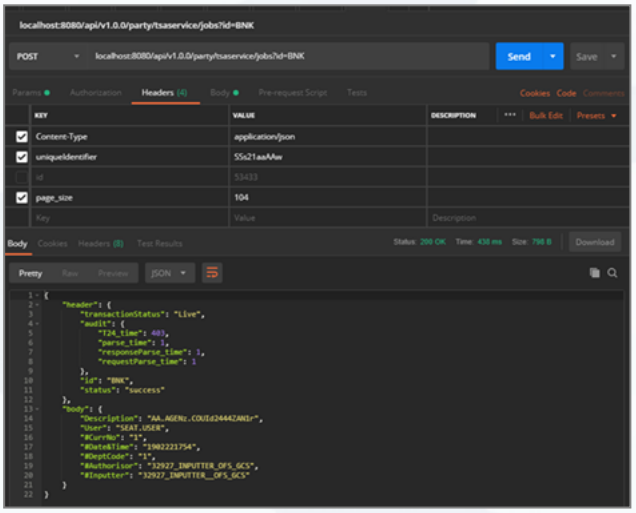
Unique Identifier

Purpose of the Guide

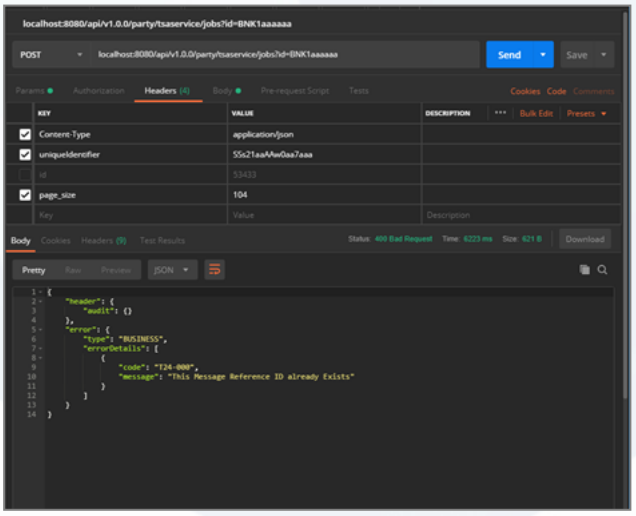
- Unique Identifier is used by T24 to detect any duplicate request sent via Version.
- If Unique Identifier value is same for two different requests (not applicable for GET Requests), that transaction is declared duplicate and is not carried out.
- In IRIS R18 we can add value to the variable “uniqueIdentifier” in the header. It can be alpha-numeric.

How to pass Unique Identifier in Postman

Value for unique Identifier can be passed in the variable "uniqueIdentifier" in the header or it can be passed as a query parameter.



- If Same Unique Identifier is used for multiple request we get this error message.



OFS Request containing Unique Identifier

- SECTOR,/I/PROCESS,INPUTT/123456/////,1008/I/NI922209/1,DESCRIPTION:1:1=CHECKDUPLICATE,SHORT.NAME:1:1=CHE2821CK

Successful OFS Response containing Unique Identifier

Tools

This section covers the following topics.

RIM Importer

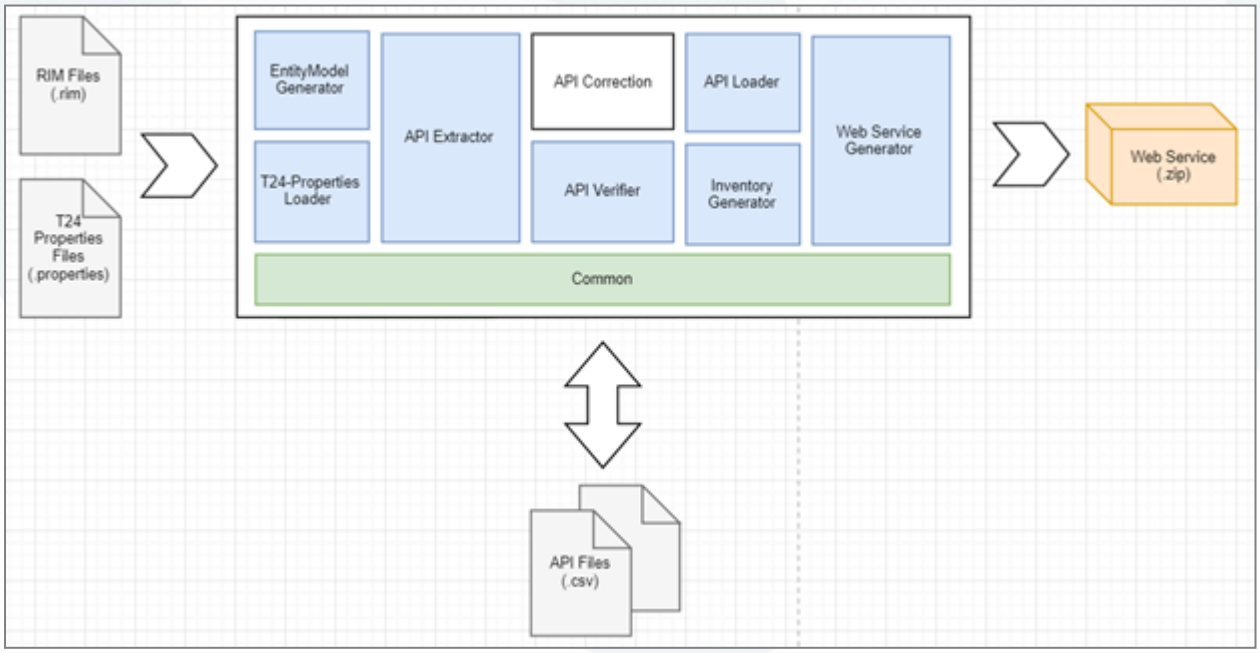
This section covers the following topics.

Overview

RIM Importer is a part of IRIS R18 tool chain, the tool intends to address the challenge of migration of legacy IRIS generated API's to current IRIS R18. The Tool also intends to migrate 80% of the existing API involving both manual and automated work process. The tool helps to "Load, Transform, Validate and Generate" API Services that is compatible to IRIS R18.

> Note: The generated API Services is still up to the user or an individual to test and verify for intent.

The IRIS R18 API migration is a commandline based tool that helps in reducing the effort of API migration. Below is the architectural diagram for RIM Importer. RIM Importer intends to take RIM and generated properties files to convert the legacy IRIS generated API to IRIS R18 compliant API as Service i.e CamelContext (.zip) files. The RIM files alone is sufficient if the artefact naming convention was followed which case the generated properties files are optional, though recommended for accurate and reliable extraction of API's and is must for field mapping for the ENQUIRY type of artifacts. The tool generates intermediate API (.csv) files for verification, nice REST compliance of the URL, other details to be compliant to IRIS R18 convention and rules as needed or applicable. The updated API (.csv) files is fed back to tool for API Service generation harnessing the existing IRIS R18 tooling system. The output of the system is Camel Service Context (.zip) files which are consumed in the IRIS R18 Service Component which in turn gets hosted by the IRIS R18 Container getting deployed and hosted on the Web Server as Container to host the T24 artefacts as L2 API Service aka Provider API's.



Workflow

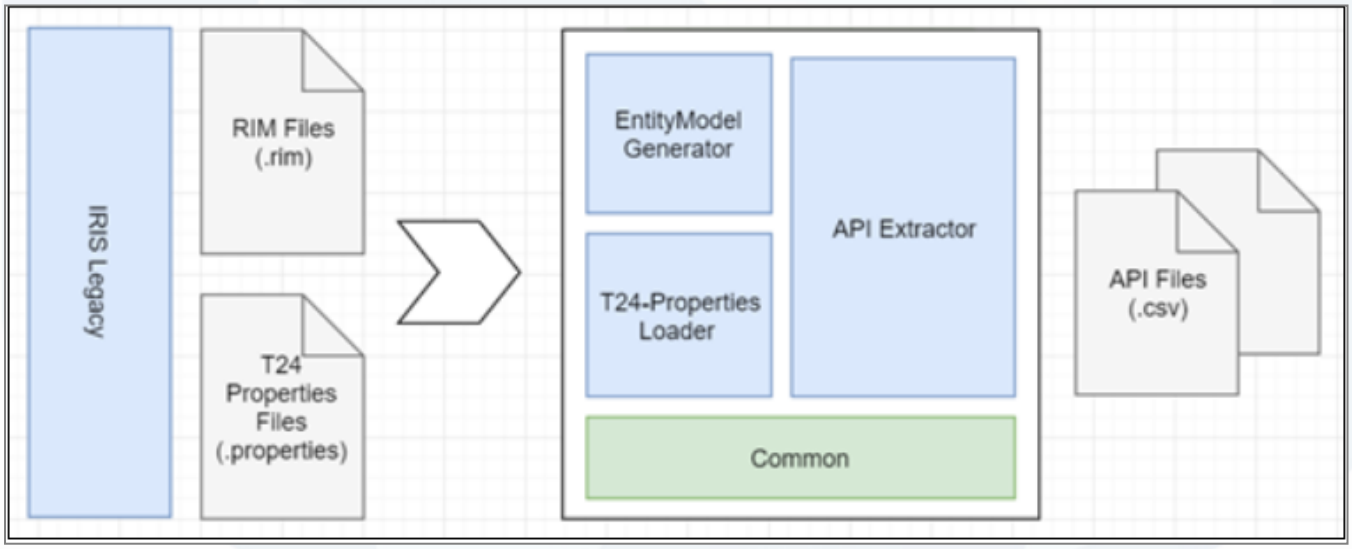
RIM Importer performs migration from Legacy IRIS System to IRIS R18 in three steps which involves both manual and automated work job towards correction and generation.

- API Extraction
- API Validation
- API Generation

API Extraction

The base RIM and T24 Properties Files from the delivered API's from the legacy IRIS is provided for API Extraction project wise. The artifacts are used to reverse engineer the API information and to compile a list of API for the migration. The extracted API list from each of the Project are compiled to respective API CSV files.

Once the API's are extracted from the provided RIMS, the tool can automatically create the relevant artefact in T24 Server by following the IRIS R18 convention making completely discoverable to the IRIS R18 designer to extend and validate the APIs using the designer. The auto creation of the artefact automatically updates resource and field description compatible to vocabulary. Vocabularies are maintained by the vertical domain experts and by the API governance team. Vocabulary standardizes the rest api path formation based on domain terms. The artefact creation step by default is not mandatory but if the naming convention is satisfactory to the expectation then auto creation of the artefact is recommended.



API Extraction

Usage : -e -f C:\IRIS-Importer\TCImages -o C:\IRIS-Importer\TCImages.csv

-e : (M) Generates the Entity Models from RIMS and optionally loads the properties files if input to the importer to extract the list of API's and details.

-f : (M) The input RIMS and Properties files are provided as files.

or

-d : (M) The input RIMS and Properties files are provided as directory.

-p : (O) Input importer properties file e.g C:\IRIS-Importer\importer.properties

-o : (O) Specifies the output file and location where the raw Extracted API is saved. This token is optional, when not specified, gets saved in a file **"IRIS-ExtractedAPIs.csv"** in the current working directory of the IRIS R18 Importer Console Application.

Artefact Creation

Usage : -c -f C:\IRIS-Importer\Release\output.csv -p C:\IRIS-Importer\Release\importer.properties -m C:\IRIS-Importer\Release\domains.properties

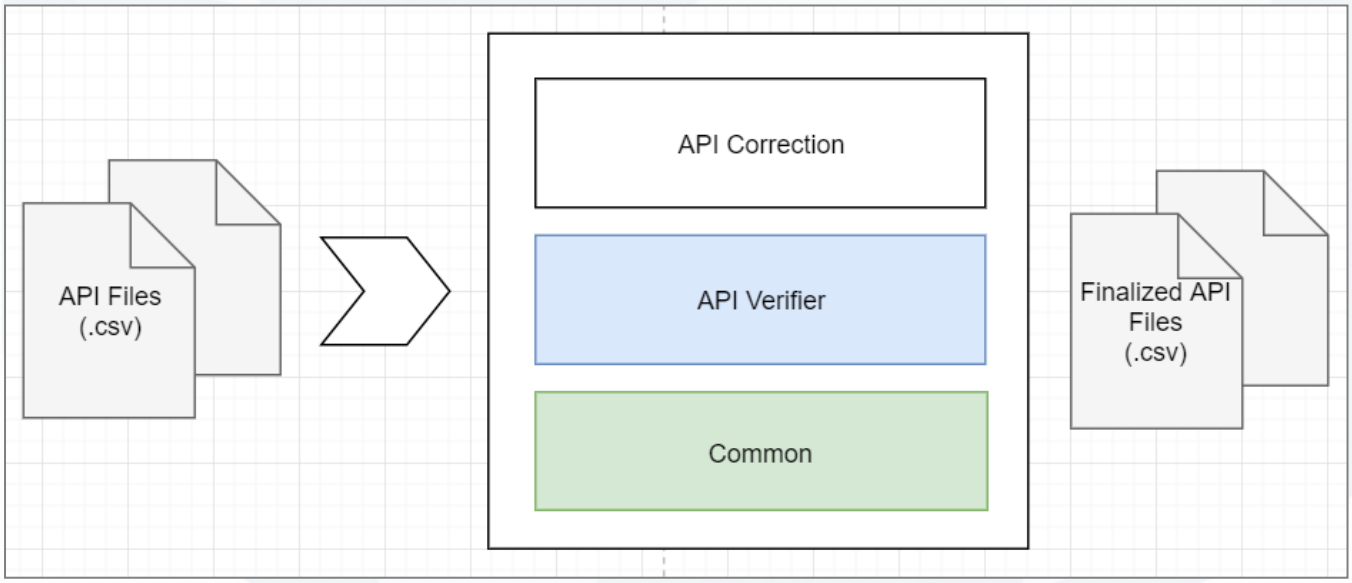
-c : (M) Sets the importer in automatic artefact creation mode, the extracted api's output file is loaded to generate artefact compatible with IRIS R18 API specification from old artefact.

-p : (M) Input importer properties file e.g C:\IRIS-Importer\importer.properties

-m : (M) Specifies the domain properties file that classifies or categorizes the artefact into domain (Maintained by each vertical or by governance team)

API Validation

The extracted API CSV is then audited manually, the URL and parameters for the enquiries are corrected manually as required to be compatible with IRIS R18 specification. The audited files are then sent to T24 for artifact verification and validation upon which the results are saved as finalized API CSV files.



Usage : -v -f C:\IRIS-Importer\TCImages.csv -o C:\IRIS-Importer\TCImagesEx.csv

-v : (M) Sets the importer in validation mode, the extracted list of API's and details are loaded for validation especially URL and Artefacts(VERSION and ENQUIRY).

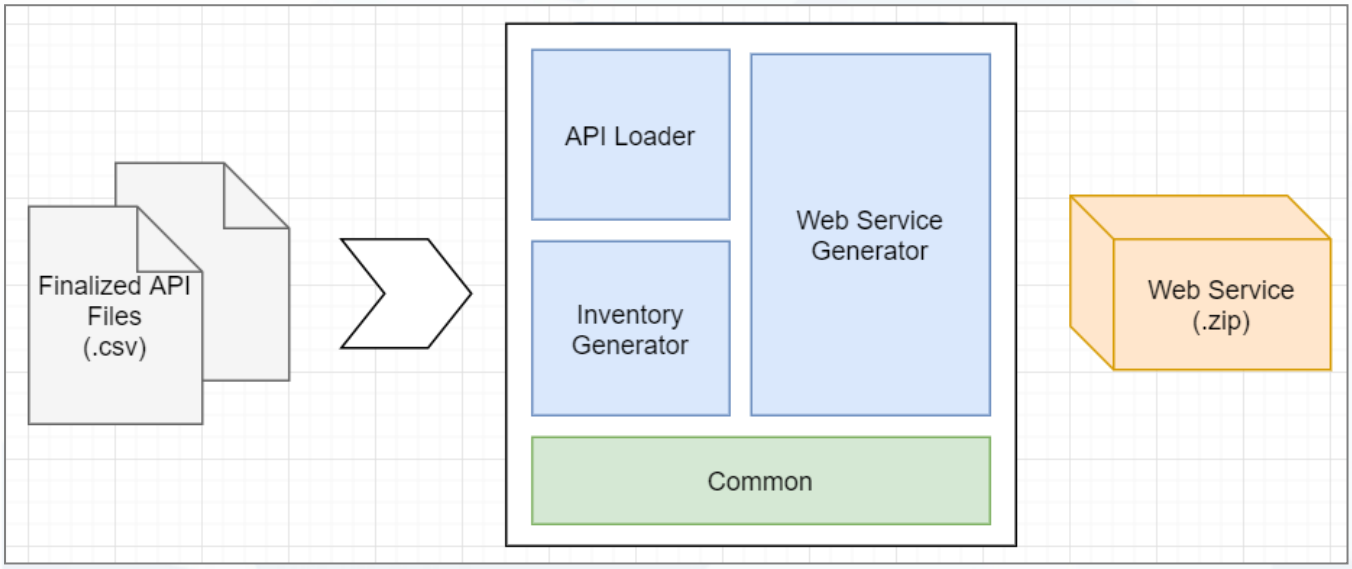
-f : (M) Input extracted api list file e.g C:\IRIS-Importer\TCImage.csv

-p : (O) Input importer properties file e.g C:\IRIS-Importer\importer.properties

-o : (O) Specifies the output file and location where the validated Extracted API is saved, token is optional.

API Generation

The finalized APIs are loaded for an intermediate inventory(json) file generation compatible to IRIS R18 API Web Service generation. The inventory file is then used to generate API Services(.zip) using existing IRIS R18 Tooling.



Inventory Generation

Usage : -i -f C:\IRIS-Importer\TCImagesEx.csv -o C:\IRIS-Importer\Importer-Provider-Inventory.json

-i : (M) Sets the importer in inventory generation mode, the extracted & validated list of API Details are loaded for inventory generation compatible with IRIS R18 API specification.

-f : (M) Input extracted & validated API file e.g C:\IRIS-Importer\TCImagesEx.csv

-o : (O) Specifies the output inventory file and location where the Inventory file is saved, token is optional

Camel Service Context Generation

Usage : -s -f C:\IRIS-Importer\Release\Importer-Provider-Inventory.json -o C:\IRIS-Importer\Release

-s : (M) Sets the importer in camel service context generation mode, the generated and validated inventory file is loaded for camel service context for the APIs compatible with IRIS R18 API specification.

-f : (M) Input generated API Inventory file e.g C:\IRIS-Importer\Release\Importer-Provider-Inventory.json

-o : (O) Specifies the output path or location where the Services needs to be generated and saved

Getting Started

Prerequisites

1. Java/JDK 1.8

   http://www.oracle.com/ , JAVA_HOME is set appropriately

2. Maven 3.3+

   https://maven.apache.org/ , MAVEN_HOME is set appropriately

3. Latest UTP with TAFJ

   http://utp.temenosgroup.com/ , TAFJ_HOME is set appropriately

4. Eclipse Oxygen or any preferred version

   http://www.eclipse.org/downloads/eclipse-packages/

5. REST Client or POSTMAN or Insomnia (*Optional*)

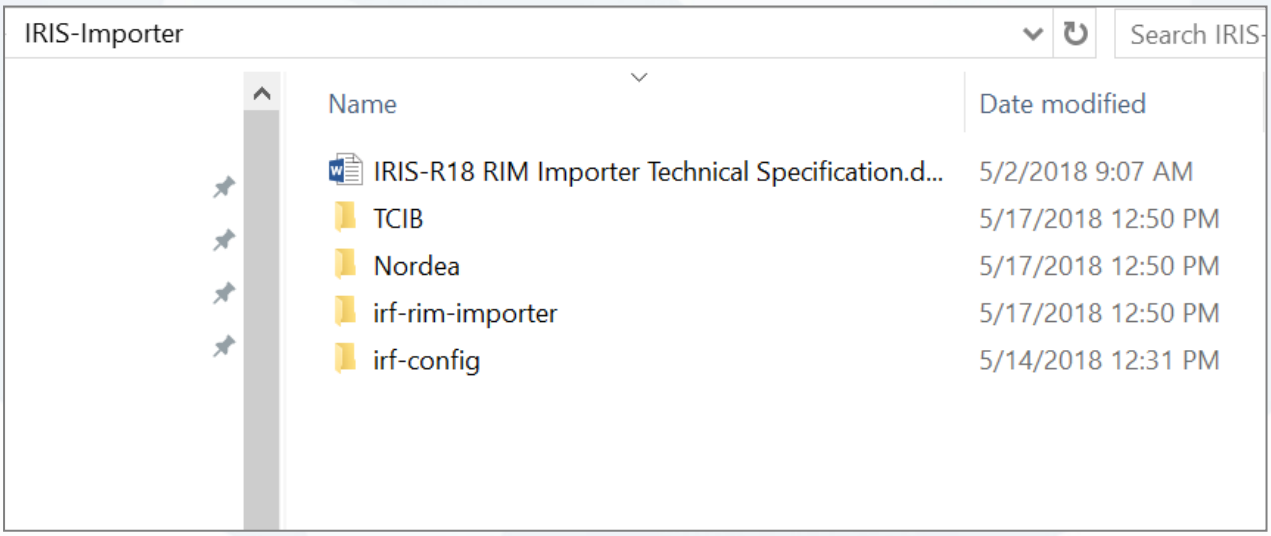
6. Browser Extensions : JSON Formatter, JSONView, etc (*Optional*)

Procedure

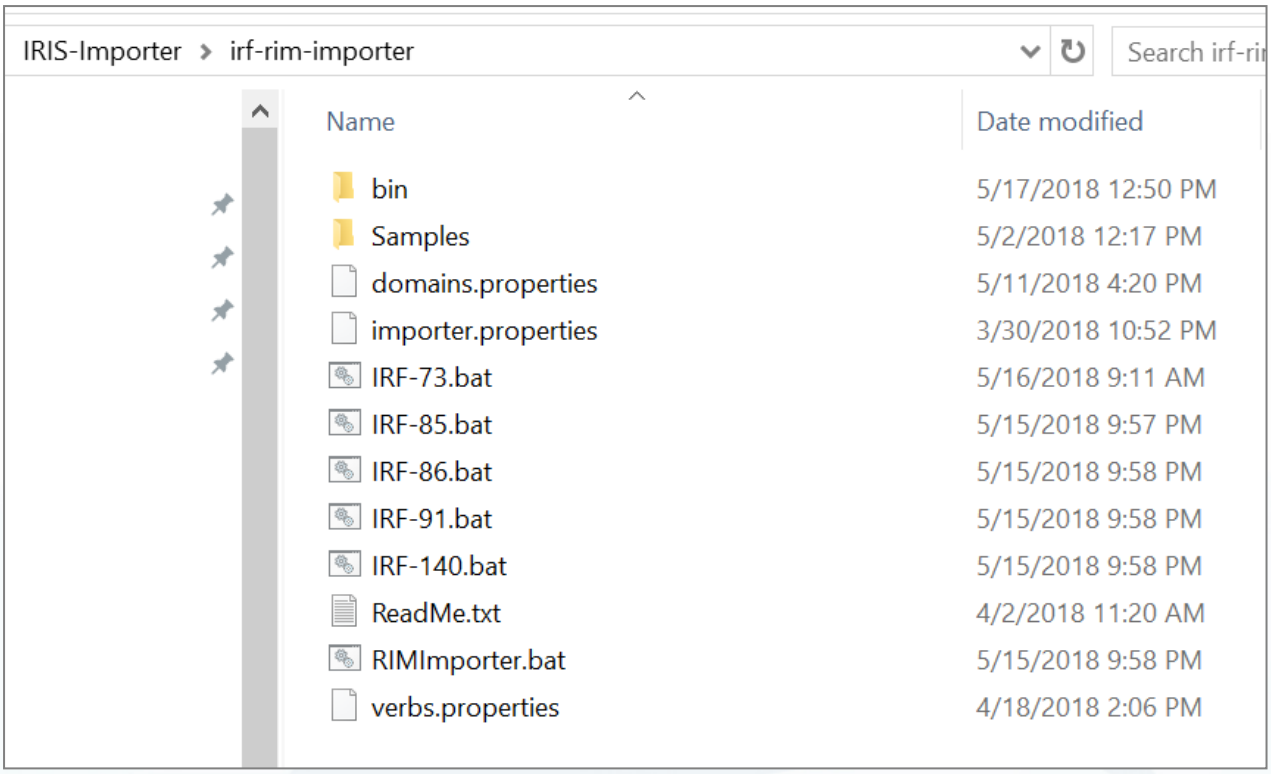
1. Download latest rimimporter released pack from Temenos Maven Repository Click here to Download.

   Link : http://maven.temenosgroup.com/content/groups/temenos/com/temenos/irf/irf-rim-importer-packager/DEV.0.0-SNAPSHOT/

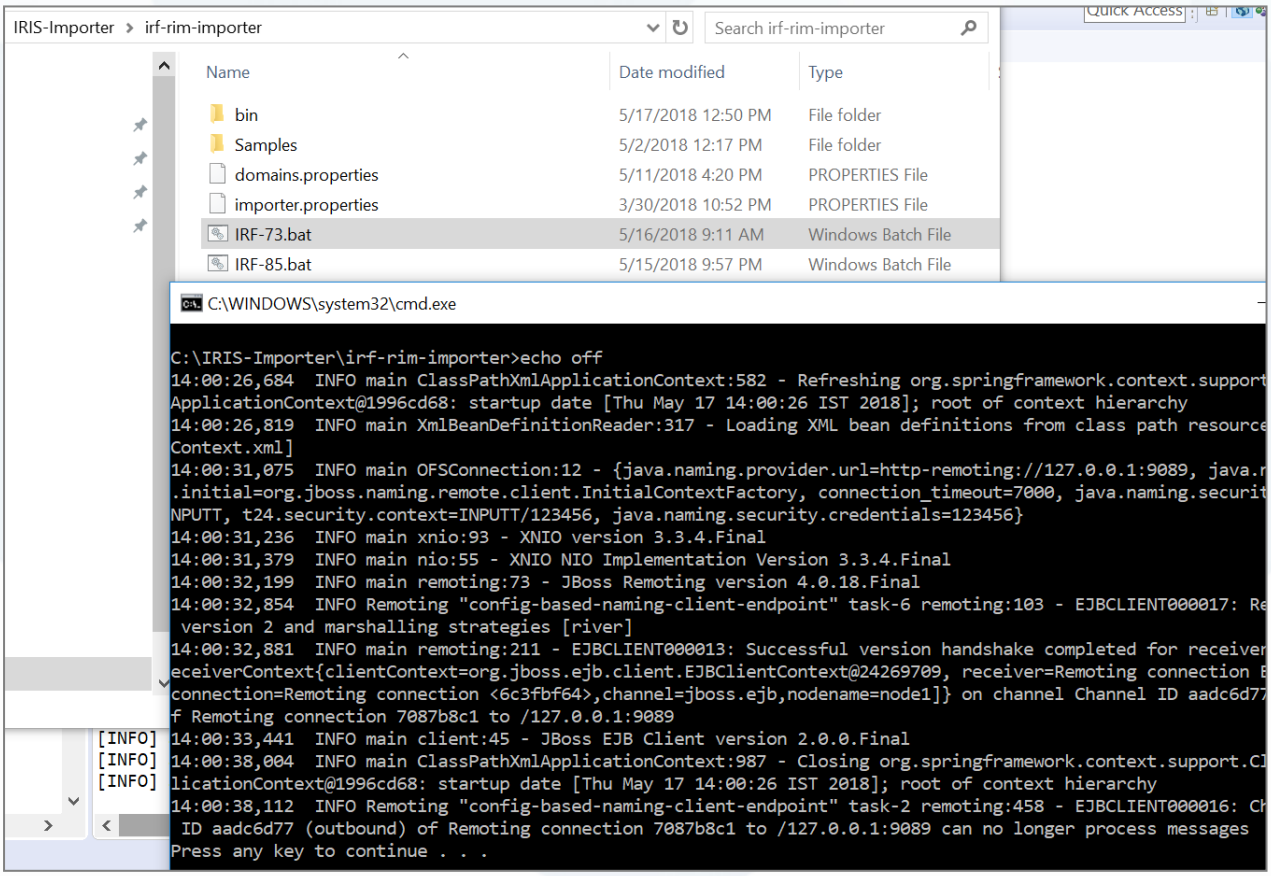
2. Extract the rimimporter pack.



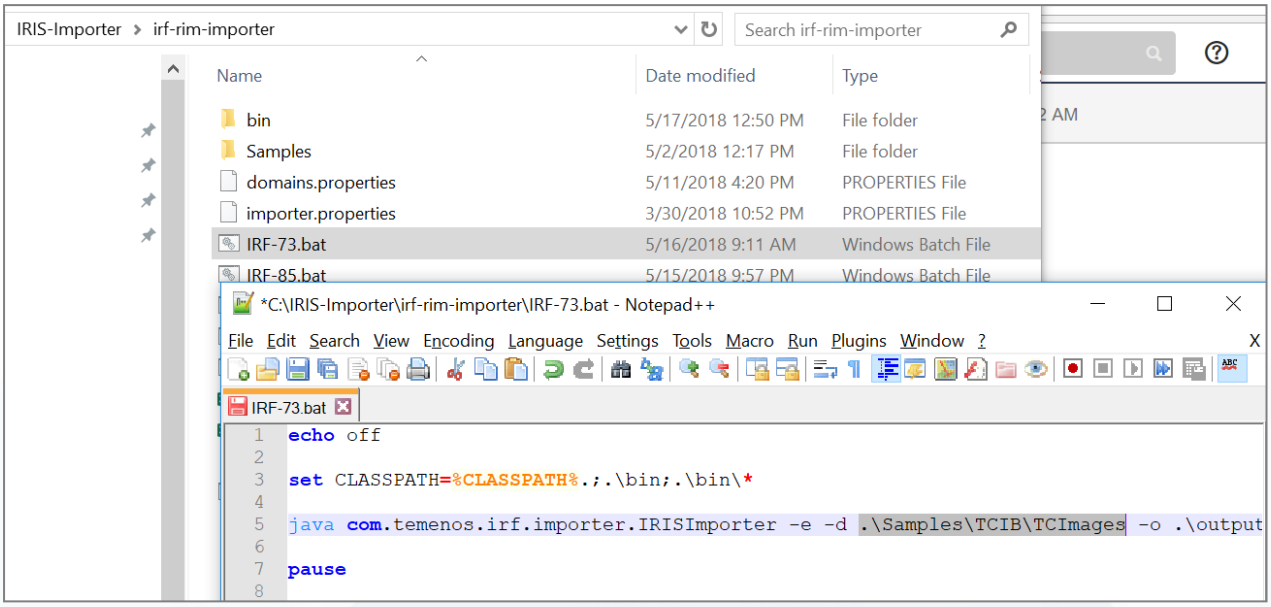
3. Navigate to location **"..\irf-rim-importer"** within the extract folder.



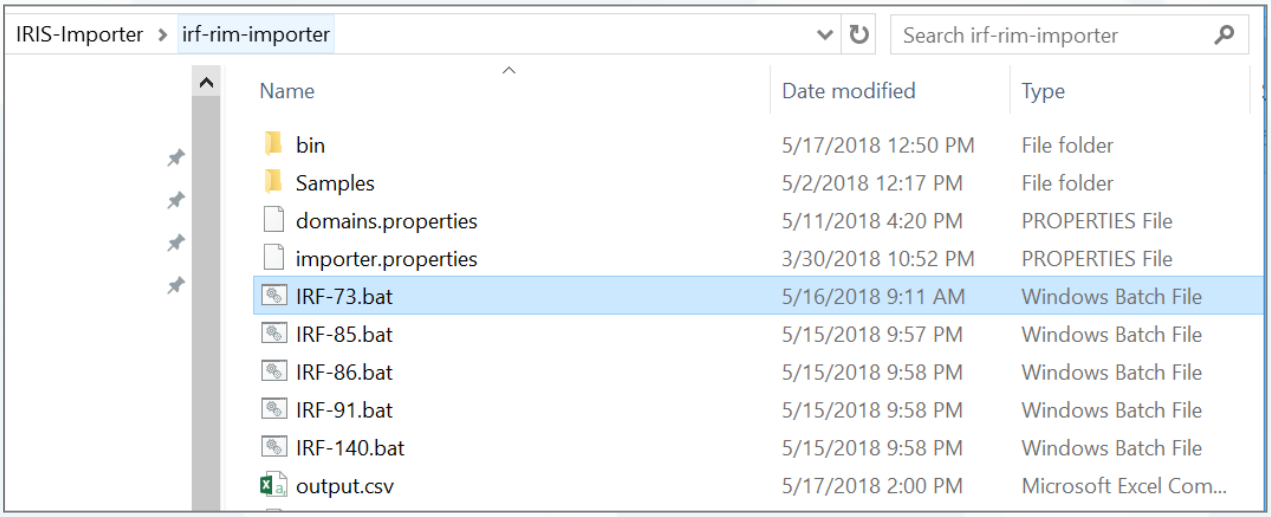
4. To Extract API's using sample RIMs (TCIB, Nordea), Run **"1.extract-apis.bat"**



Note: Open Run IRF-73.bat and Edit the rim Source Path



5. Extracted APIs will be written and found in file **"output.csv"**



6. Verify and Confirm the API extracted from RIMS and edit, if required

	A	B	C	D	E	F
1	Model	T24Artifact	T24Type	T24Module	Path	Commands
2	enqTC.IM.DOCUMENT.IMAGE	TC.IM.DOCUMENT.IMAGE	ENQUIRY	IM	/{companyid}/enqTclmDocumentImages()	GETEntities filter="{filter}":method=GET
3	enqTC.IM.DOCUMENT.IMAGE	TC.IM.DOCUMENT.IMAGE	ENQUIRY	IM	/{companyid}/enqTclmDocumentImages('{id}')	GETEntity method=GET
4	enqTC.IM.DOCUMENT.IMAGE	TC.IM.DOCUMENT.IMAGE	ENQUIRY	IM	/{companyid}	ImageDownload
5	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs()/metadata	T24FieldMetadata method=POST
6	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs('{id}')/error	NoopGET method=GET
7	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tc/ContextEnquiries	NoopGET method=GET
8	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}	T24SilentState method=GET
9	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_TcEntry	NoopGET method=GET
10	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs()	GETEntities filter="{filter}":method=GET
11	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs_IAuth()	GetlauthEntities filter="{filter}":method=GET
12	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs_HAuth()	GetHauthEntities filter="{filter}":method=GET
13	verIM.DOCUMENT.IMAGE,TC	IM.DOCUMENT.IMAGE,TC	VERSION	IM	/{companyid}/verImDocumentImage_Tcs('{id}')	GetEntity method=GET,InterimTransition method=G

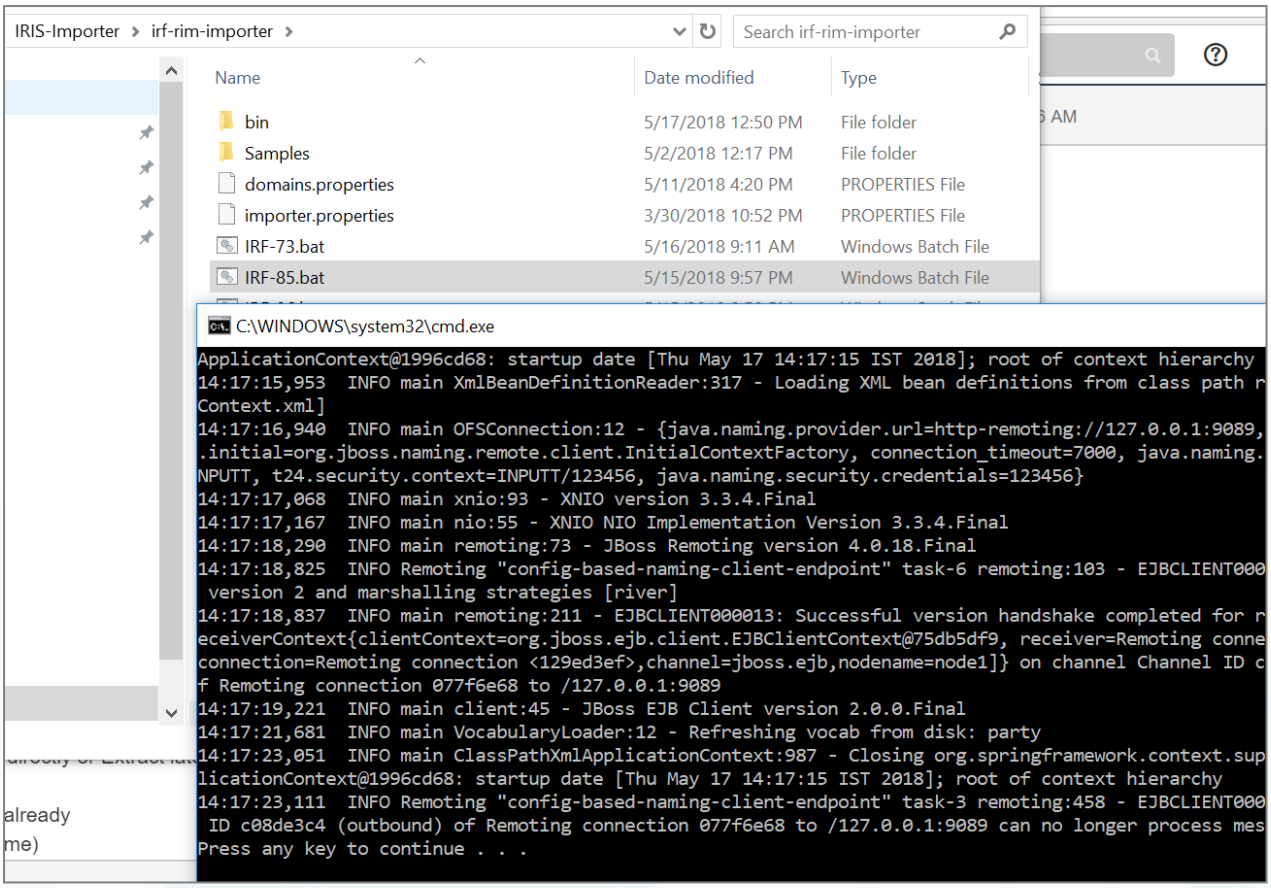
H	I	J	K	L	M
Name	Command	Method	Version	Namespace	URLMapper
enqTclmDocumentImages	GETEntities	GET		T24.enqTclmDocumentImage.enqTclmDocumentImages	T24-enqTclmDocumentImage
enqTclmDocumentImage	GETEntity	GET		T24.enqTclmDocumentImage.enqTclmDocumentImage	T24-enqTclmDocumentImage
TCImages	ImageDownload	GET		T24.enqTclmDocumentImage.TCImages	T24-enqTclmDocumentImage
verImDocumentImage_Tcs_metadata	T24FieldMetadata	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tcs_metadata	T24-verImDocumentImage_Tc
verImDocumentImage_Tc_errorHandler	NoopGET	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tc_errorHandler	T24-verImDocumentImage_Tc
verImDocumentImage_Tc_ContextEnquiries	NoopGET	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tc_ContextEnquiries	T24-verImDocumentImage_Tc
verImDocumentImage_Tc_silent	T24SilentState	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tc_silent	T24-verImDocumentImage_Tc
verImDocumentImage_TcEntry	NoopGET	GET		T24.verImDocumentImage_Tc.verImDocumentImage_TcEntry	T24-verImDocumentImage_Tc
verImDocumentImage_Tcs	GETEntities	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tcs	T24-verImDocumentImage_Tc
verImDocumentImage_Tcs_IAuth	GetIauthEntities	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tcs_IAuth	T24-verImDocumentImage_Tc
verImDocumentImage_Tcs_HAuth	GetHauthEntities	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tcs_HAuth	T24-verImDocumentImage_Tc
verImDocumentImage_Tc	GetEntity	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tc	T24-verImDocumentImage_Tc
verImDocumentImage_Tc_Live	GetEntity	GET		T24.verImDocumentImage_Tc.verImDocumentImage_Tc_Live	T24-verImDocumentImage_Tc

Note: Editable Columns Domain, Parent Resource, Resource, ResourceId, Verb, OperationId, OperationKey, Selections

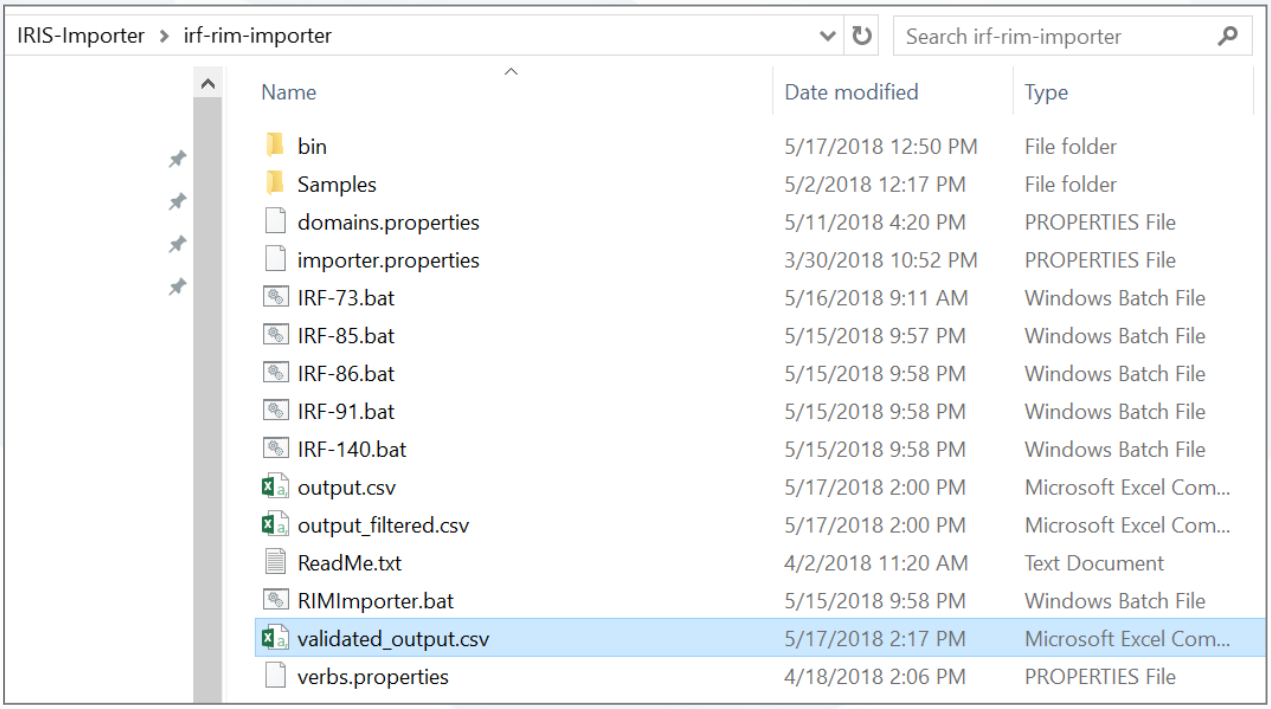
O	P	Q	R	S	T	U	
Domain	Parent Resource	Resource	ResourceId	Verb	Url	OperationId	Op
party		imageLists		get	/party/imagelists	getEnqTclmDocumentImages	IM.API.TC.IM.DO
party		imageList	{imageId}	get	/party/imagelist/{imageId}	getEnqTclmDocumentImage	IM.API.TC.IM.DO
party		imageList		get	/party/imagelist	getTCImages	IM.API.TC.IM.DO
meta		documentimage		get	/meta/screens/IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0	getVerImDocumentImageTcsMetadata	IM.DOCUMENT.I
party		documentimage	{imageId}	get	/party/documentimage/{imageId}	getVerImDocumentImageTcErrorHandler	IM.DOCUMENT.I
party		documentimage		get	/party/documentimage	getVerImDocumentImageTcContextEnquiries	IM.DOCUMENT.I
party		documentimage		get	/party/documentimage	getVerImDocumentImageTcSilent	IM.DOCUMENT.I
party		documentimage		get	/party/documentimage	getVerImDocumentImageTcEntry	IM.DOCUMENT.I
party		documentimages		get	/party/documentimages	getVerImDocumentImageTcs	IM.DOCUMENT.I
party		documentimages		get	/party/documentimages	getVerImDocumentImageTcsIAuth	IM.DOCUMENT.I
party		documentimages		get	/party/documentimages	getVerImDocumentImageTcsHAuth	IM.DOCUMENT.I
party		documentimage	{imageId}	get	/party/documentimage/{imageId}	getVerImDocumentImageTc	IM.DOCUMENT.I
party		documentimage	{imageId}	get	/party/documentimage/{imageId}	getVerImDocumentImageTcLive	IM.DOCUMENT.I

S	T	U	V	W	X	Y
Verb	Url	OperationId	OperationKey	Selections	Status	Reason
get	/party/imagelists	getEnqTclmDocumentImages	IM.API.TC.IM.DOCUMENT.IMAGE.1.0.0	[	FALSE	
get	/party/imagelist/{imageId}	getEnqTclmDocumentImage	IM.API.TC.IM.DOCUMENT.IMAGE.1.0.0	[	FALSE	
get	/party/imagelist	getTCImages	IM.API.TC.IM.DOCUMENT.IMAGE.1.0.0	[	FALSE	
get	/meta/screens/IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0	getVerImDocumentImageTcsMetadata	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage/{imageId}	getVerImDocumentImageTcErrorHandler	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage	getVerImDocumentImageTcContextEnquiries	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage	getVerImDocumentImageTcSilent	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage	getVerImDocumentImageTcEntry	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimages	getVerImDocumentImageTcs	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimages	getVerImDocumentImageTcsIAuth	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimages	getVerImDocumentImageTcsHAuth	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage/{imageId}	getVerImDocumentImageTc	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	
get	/party/documentimage/{imageId}	getVerImDocumentImageTcLive	IM.DOCUMENT.IMAGE,IM.API.TC.1.0.0		FALSE	

7. Validate the extracted API's against the IRISX conventions for conformance, Run **"2.validate-apis.bat"**



8. The validation status along with the reason will be updated in file "**validated_output.csv**"



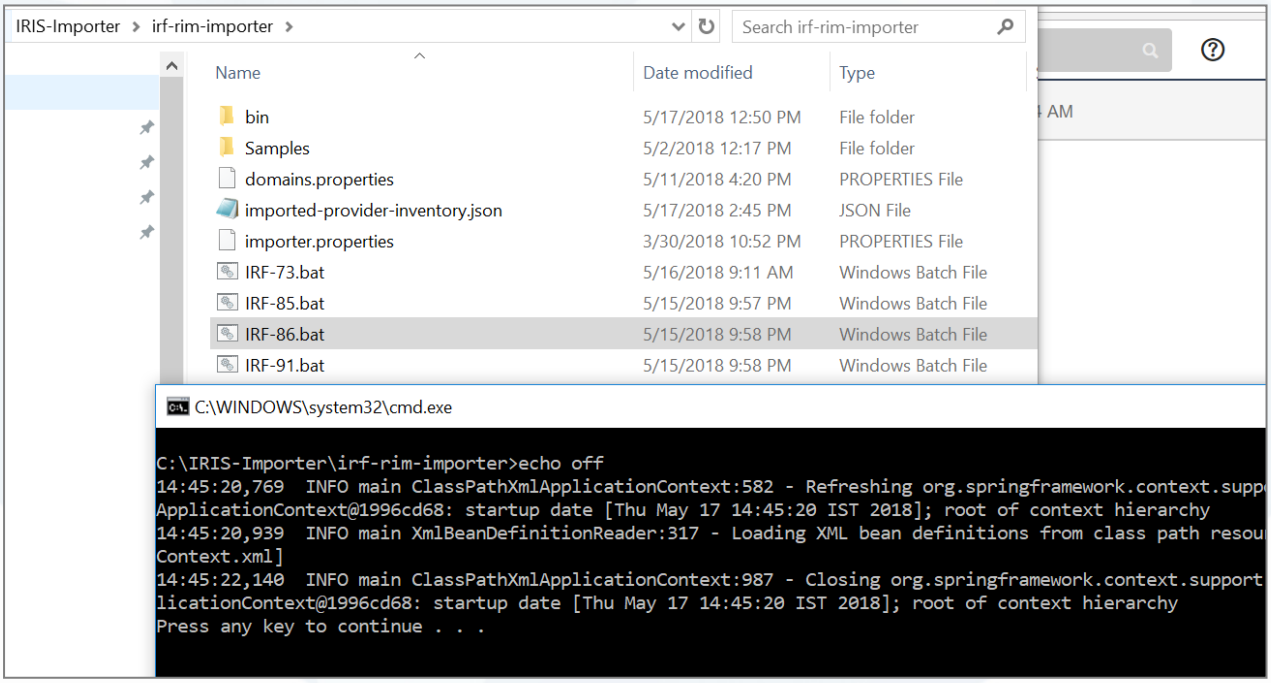
9. Refer Status column for conformance status and refer Reason column for the failure cause.

O	P	Q	R	S	T	U	V	W	X	Y
Domain	Parent Re	Resource	Resourcel	Verb	Url	Operation	Operation	Selections	Status	Reason
party		contexten	{entity}	get	/party/cor	getContex	XX.API.CONTEXTENQ		FALSE	Artefact doesn't exists or invalid
party		errors		get	/party/err	getWill	XX.API.ERRORS.1.0.0		FALSE	Artefact doesn't exists or invalid
party		errors		get	/party/err	getCheckIf	XX.API.ERRORS.1.0.0		FALSE	Artefact doesn't exists or invalid
party		errors		get	/party/err	getProces	XX.API.ERRORS.1.0.0		FALSE	Artefact doesn't exists or invalid
party		errors		get	/party/err	getErrors	XX.API.ERRORS.1.0.0		FALSE	Artefact doesn't exists or invalid
party		messages		get	/party/me	getProces	XX.API.MESSAGES.1.(		FALSE	Artefact doesn't exists or invalid
party		nextstate		get	/party/ne	getNextSt	XX.API.NEXTSTATE.1.		FALSE	Artefact doesn't exists or invalid
party		imageLists		get	/party/im	getEnqTcl	IM.API.TC	[	TRUE	
party		imageList	{imageId}	get	/party/im	getEnqTcl	IM.API.TC	[	TRUE	
party		imageList		get	/party/im	getTCImag	IM.API.TC	[	TRUE	
meta		documentimage		get	/meta/scr	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		document	{imageId}	get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		documentimage		get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		documentimage		get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		documentimage		get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		document	{imageId}	get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	
party		documentimages		get	/party/do	getVerImD	IM.DOCUMENT.IMA(		TRUE	

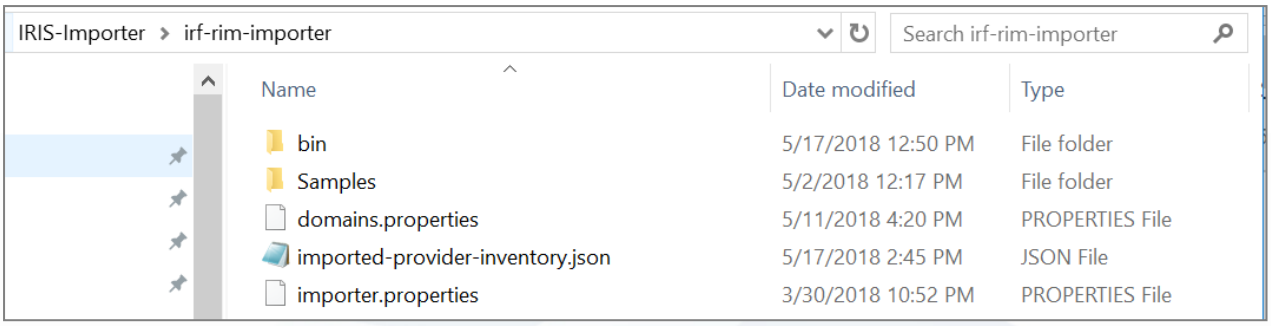
10. Correct and re-run steps 7-9 until Status columns gets TRUE.

> Note: Update vocabulary or artifact if resource or property doesn't exist (VERSION, ENQUIRY)

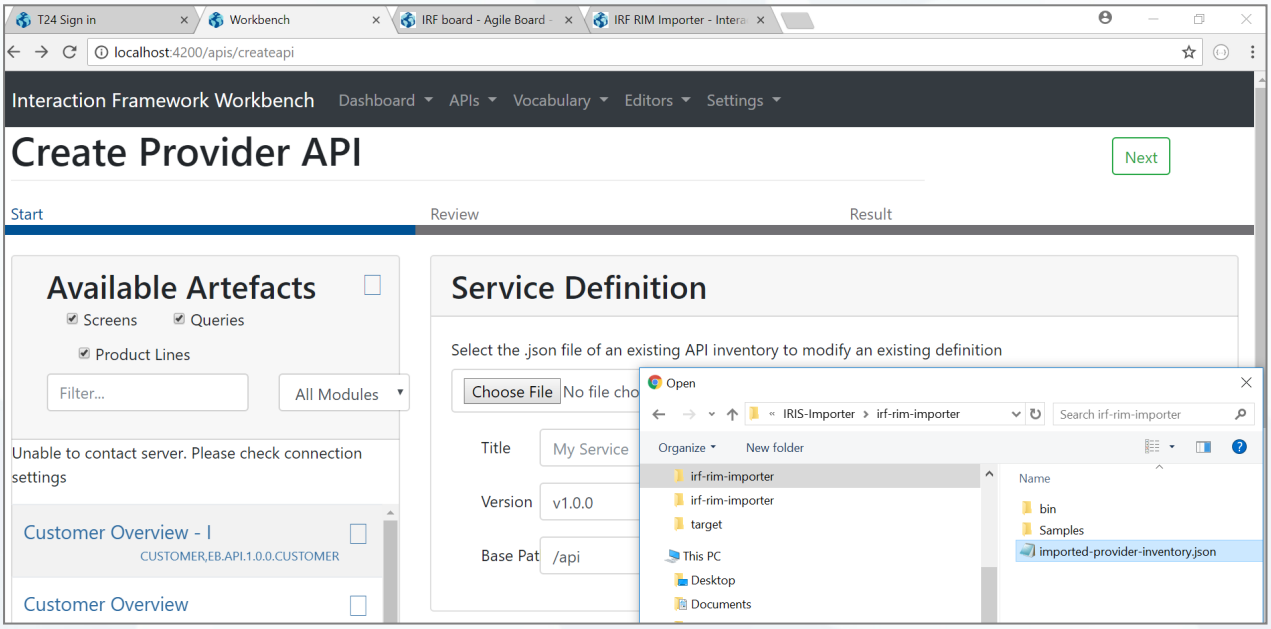
11. The extracted API's with **Status = TRUE** are ready for the inventory generation, Run **"3.generate-inventory.bat"**

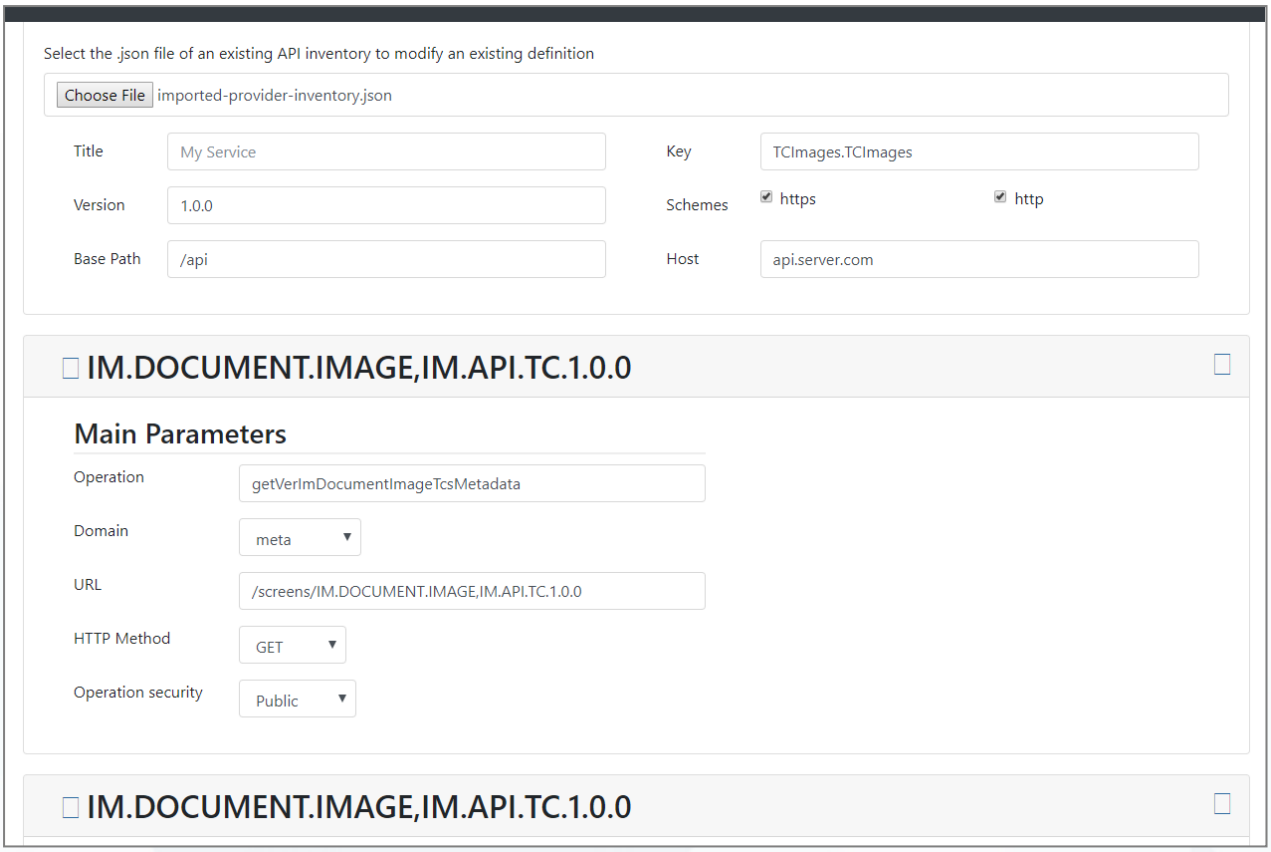


12. The inventory file **"imported-provider-inventory.json"** will be generated in IRISX service generation schema.



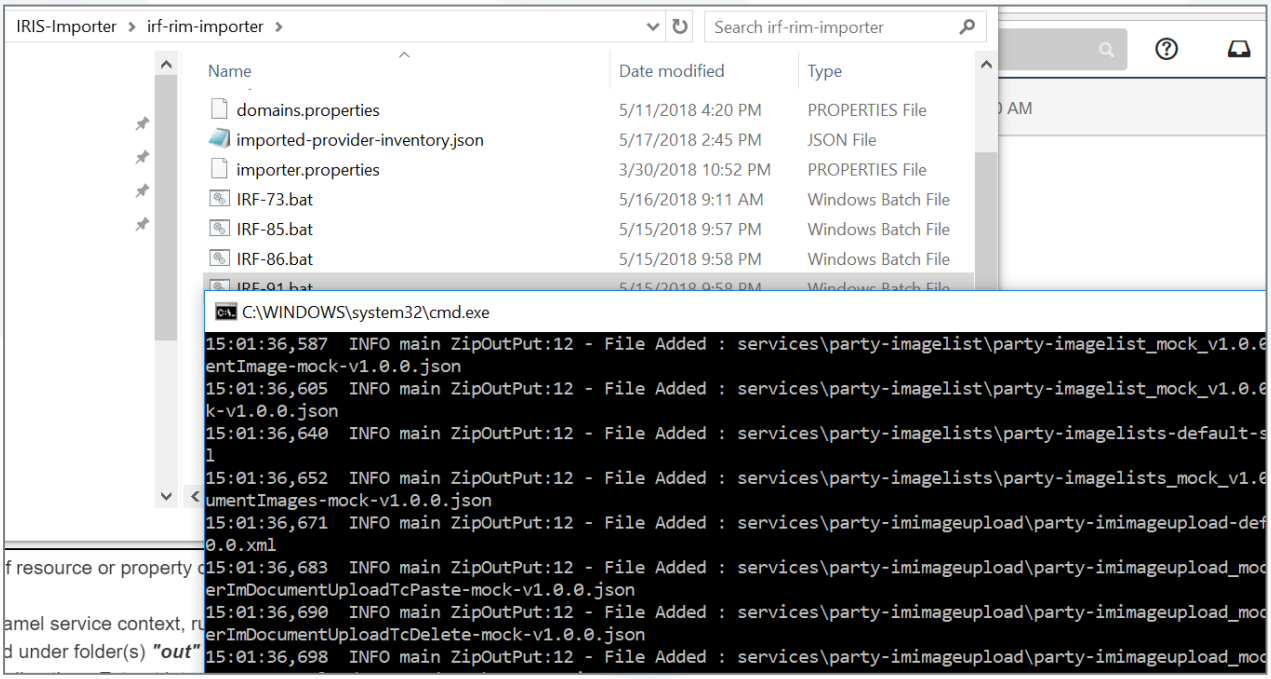
Note: Verify the correctness of import via **Workbench** & to generate the service using **Workbench**



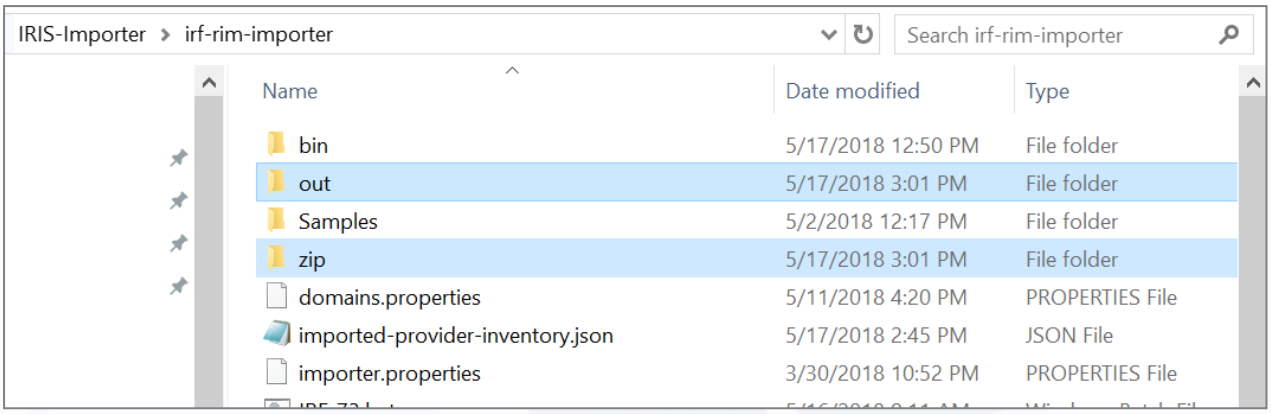


Note: Update vocabulary or artifact if resource or property doesn't exist (VERSION, ENQUIRY)

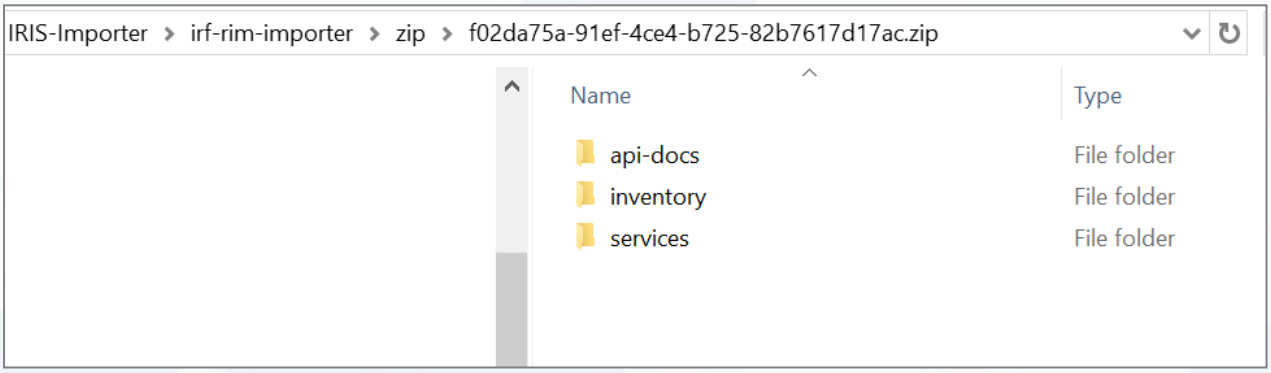
13. The inventory is ready to generate camel service context, run **"4.generate-service.bat"**

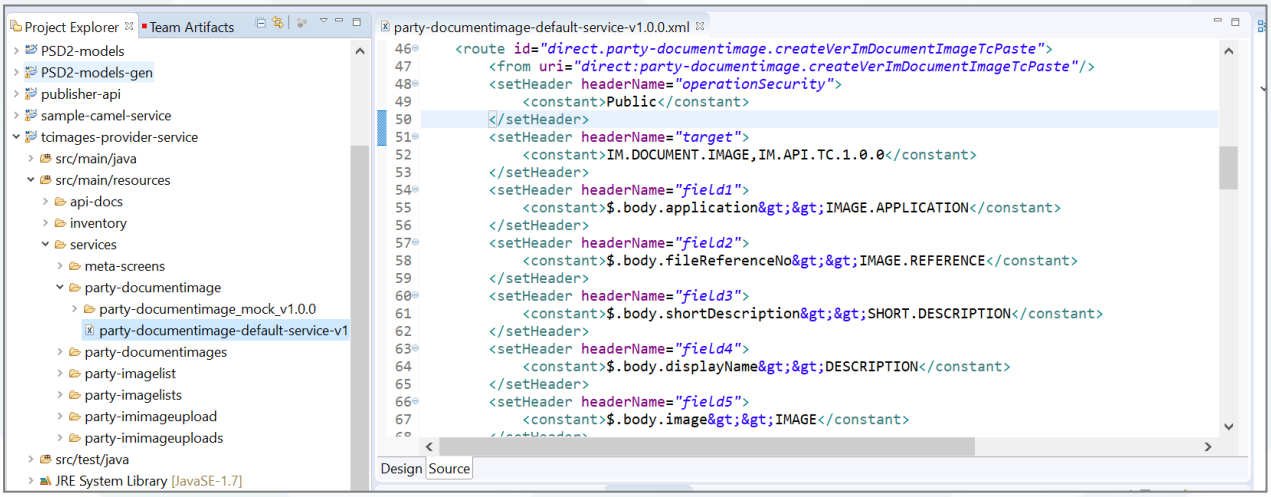


14. The service context will be generated under folder(s) **"out"** and **"zip"** respectively.



15. Copy the service context from **"out"** directly or Extract latest archive from **"zip"** to IRF Service Project **"src/main/resources"**





16. Build service project.

> Note: Refer documentation on how to create and build IRF Service Project

17. Run container project, if not running already.

Note: Refer documentation on how to create, build and run IRF Service Container Project

18. Open REST Client or Browser (Chrome)

Note: In case, you choose Browser then appropriate extensions are installed to view JSON e.g JSON Formatter

19. Enter http://localhost:8080/api/v1.0.0/meta/apis"

20. Find the created service listed with all imported endpoints listed.

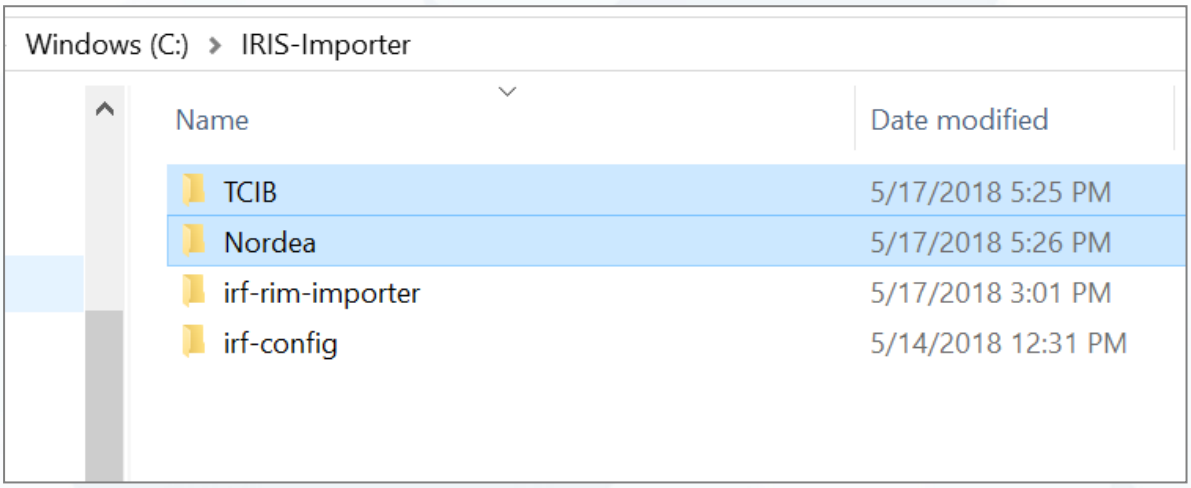
Things to remember

1. Click Download to download the latest rimimporter release package from the Temenos Maven Repository

   Link : http://maven.temenosgroup.com/content/groups/temenos/com/temenos/irf/irf-rim-importer-packager/DEV.0.0-SNAPSHOT/

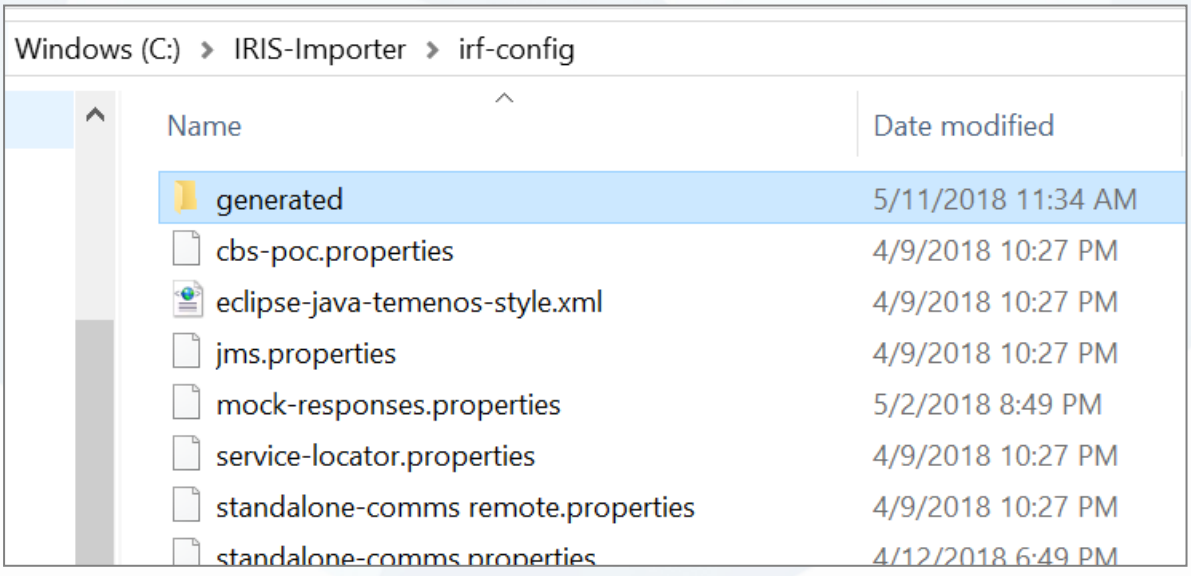
2. Setup Import Configuration

   a. Copy all rims to any valid folder and edit **"1.extract-apis.bat"** to change the source path.



   b. Copy all corresponding generated properties files to **"..\irf-config\generated"**

> Note: Needed for hand-coded or modified rims not following the IRIS1 naming standards



3. T24 - Jboss is running **[Mandatory]**
4. IRF Service Container Project - Jetty is running **[Mandatory]**

> Note: Refer documentation on how to create IRF Service Container Project.

5. IRF Service Project - referenced in IRF Service Container Project **[Mandatory]**

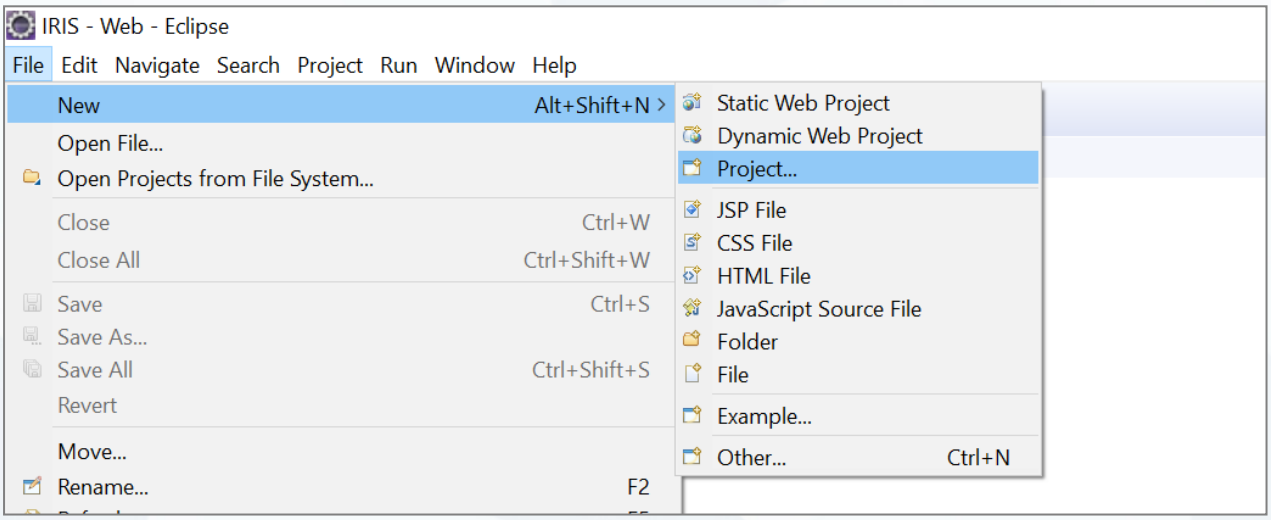
> Note: Refer documentation on how to create IRF Service Project.

6. Workbench - JBoss is running **[Optional]**

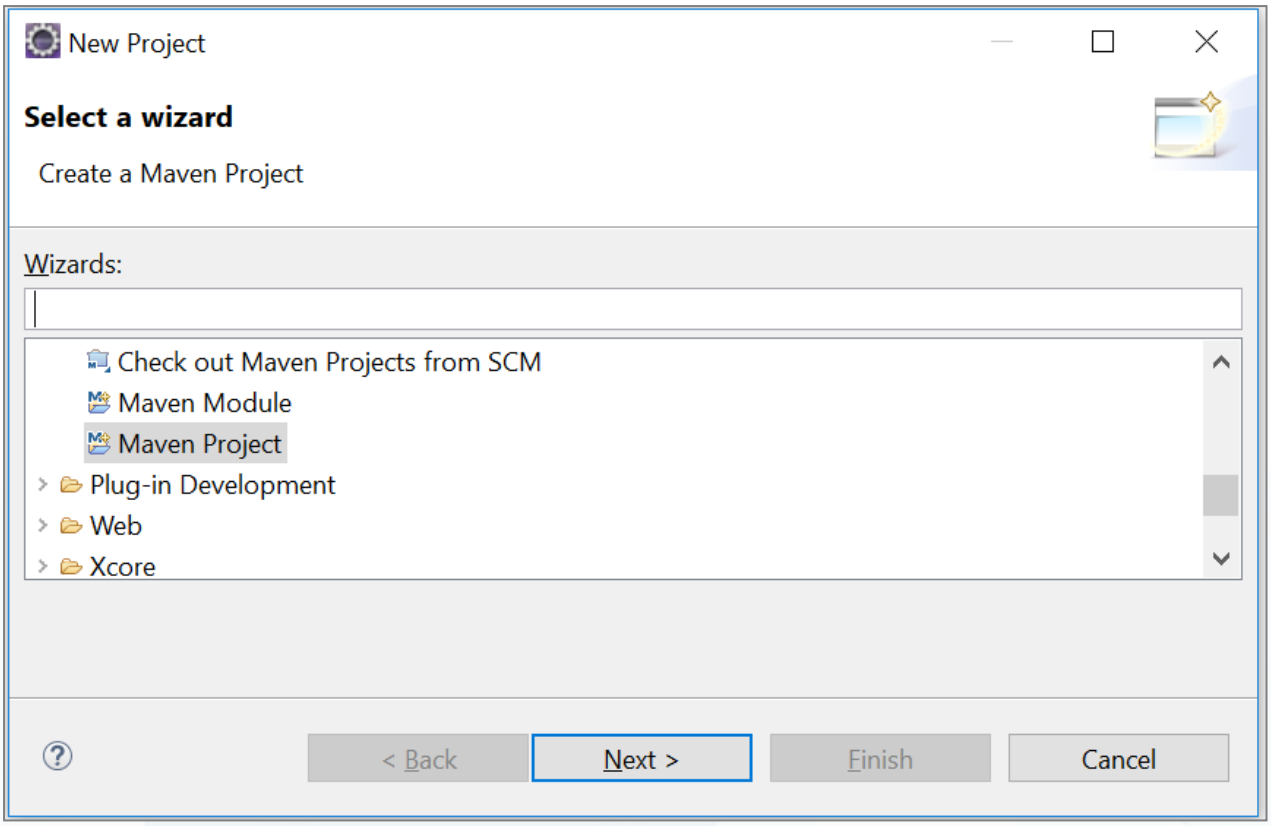
   Link: http://localhost:9089/irf-web-client/

How to create IRF Service Container Project

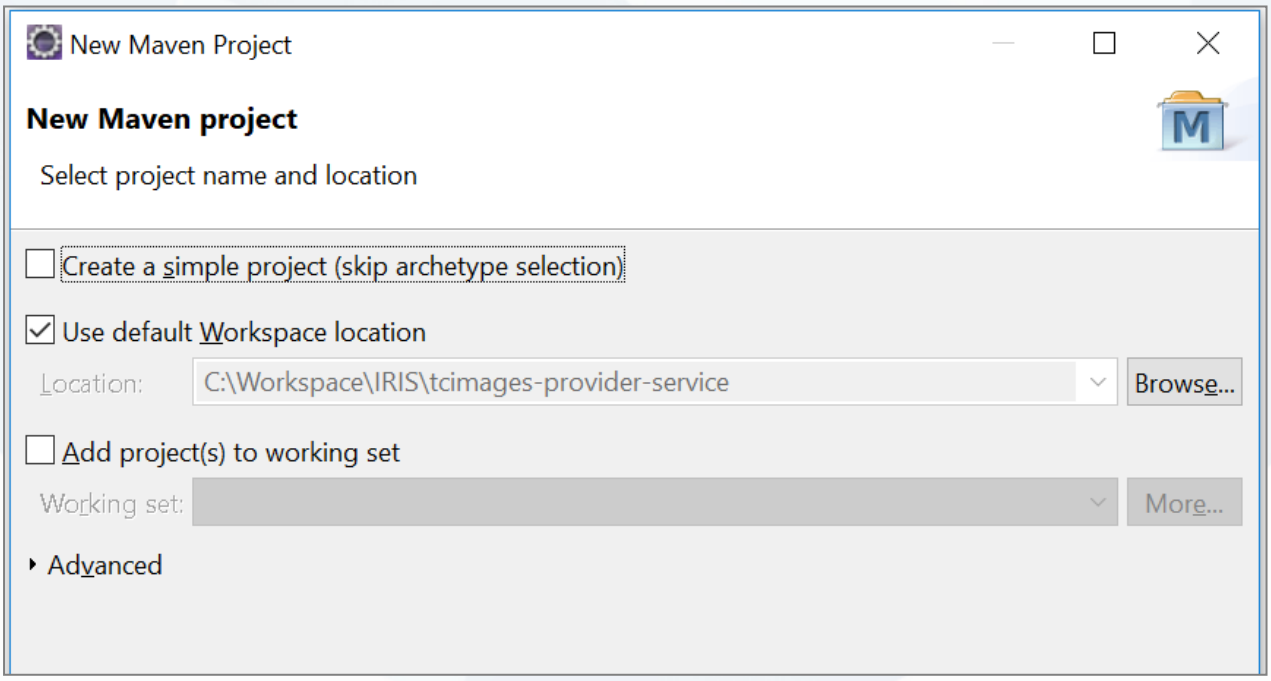
1. Open Eclipse or EDS
2. Click File > New Project



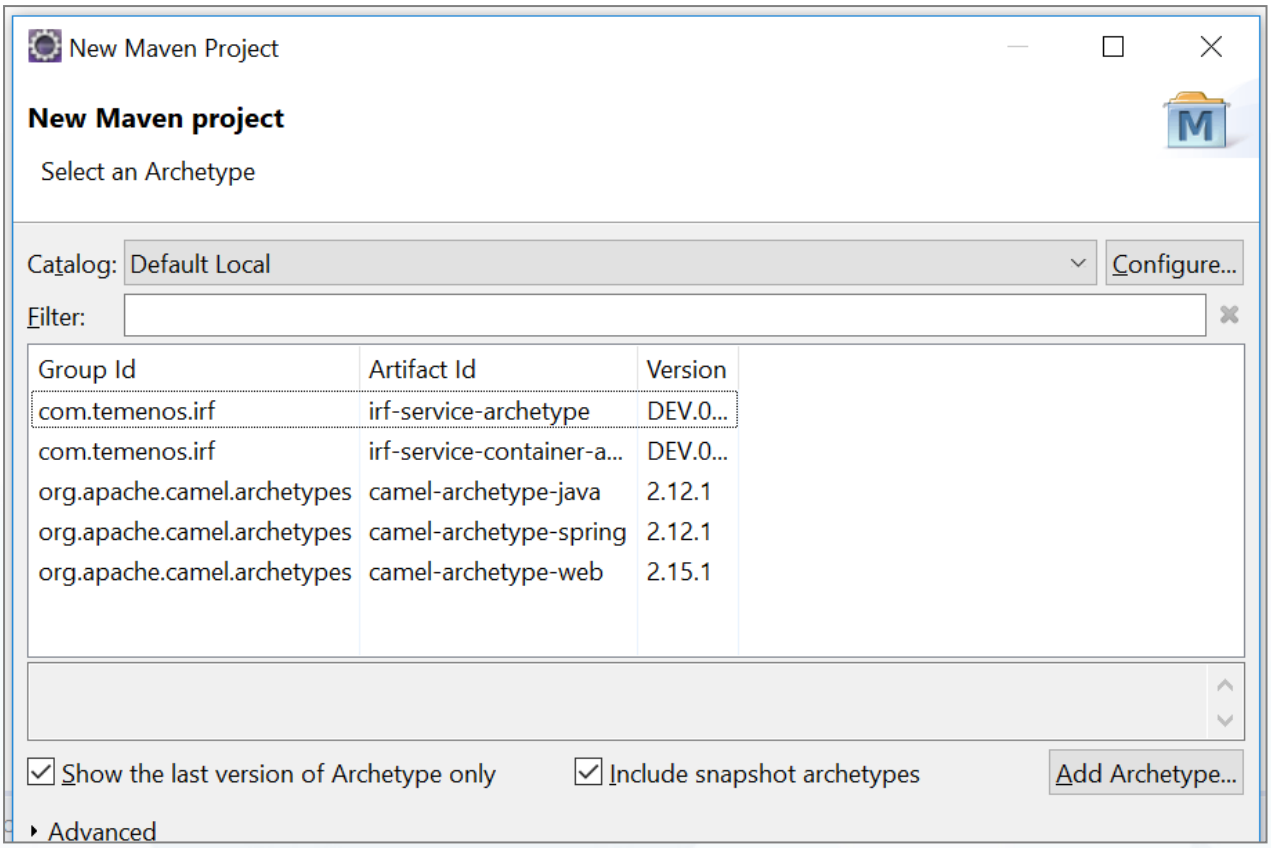
3. Select Maven Project
4. Click "**Next**"



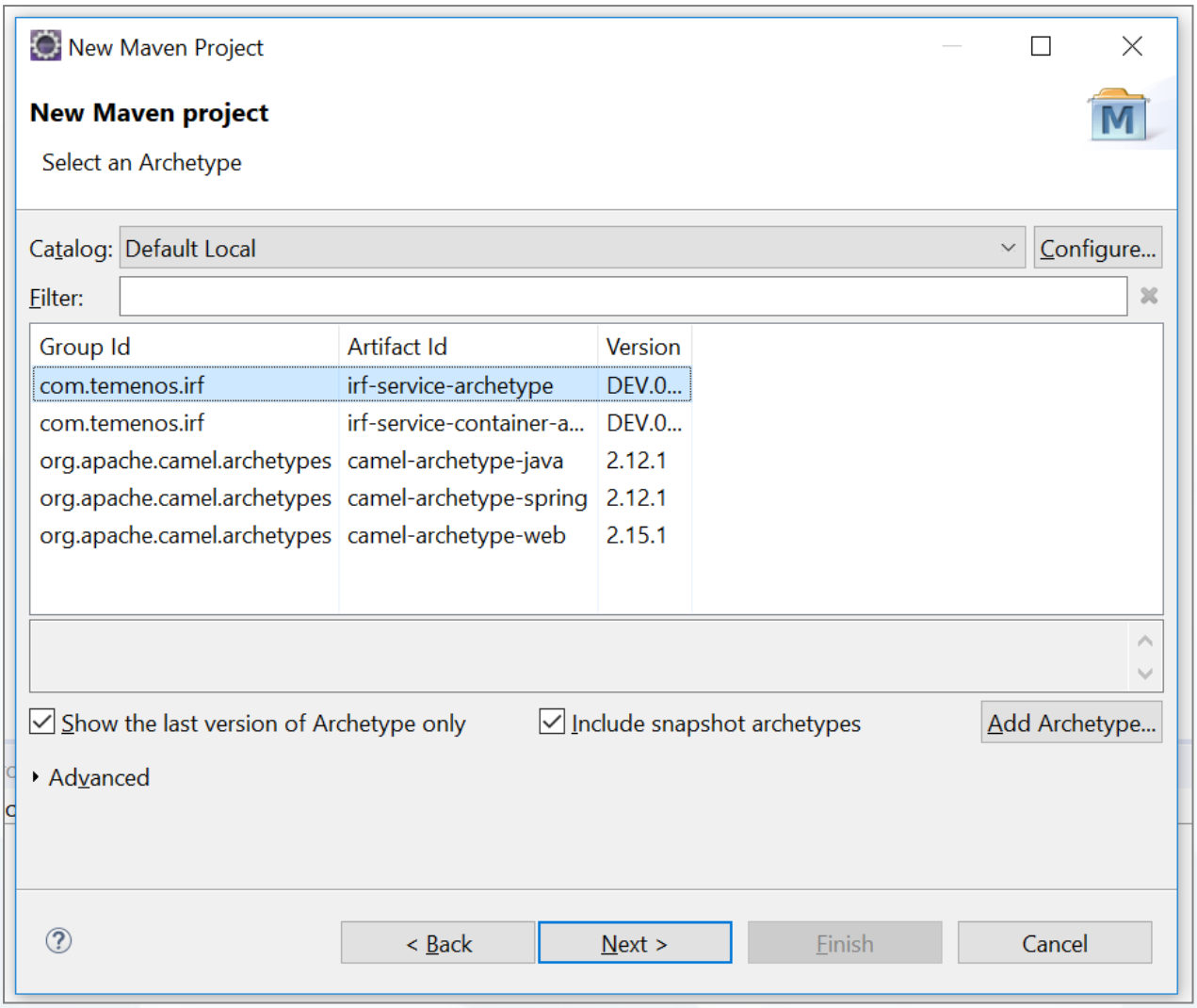
5. Uncheck "**Create a simple project**"
6. Click "**Next**"



7. Check "**Include snapshot archetypes**"



8. Select Item with Artifact Id "**irf-service-container-archetype**"
9. Click "**Next**"



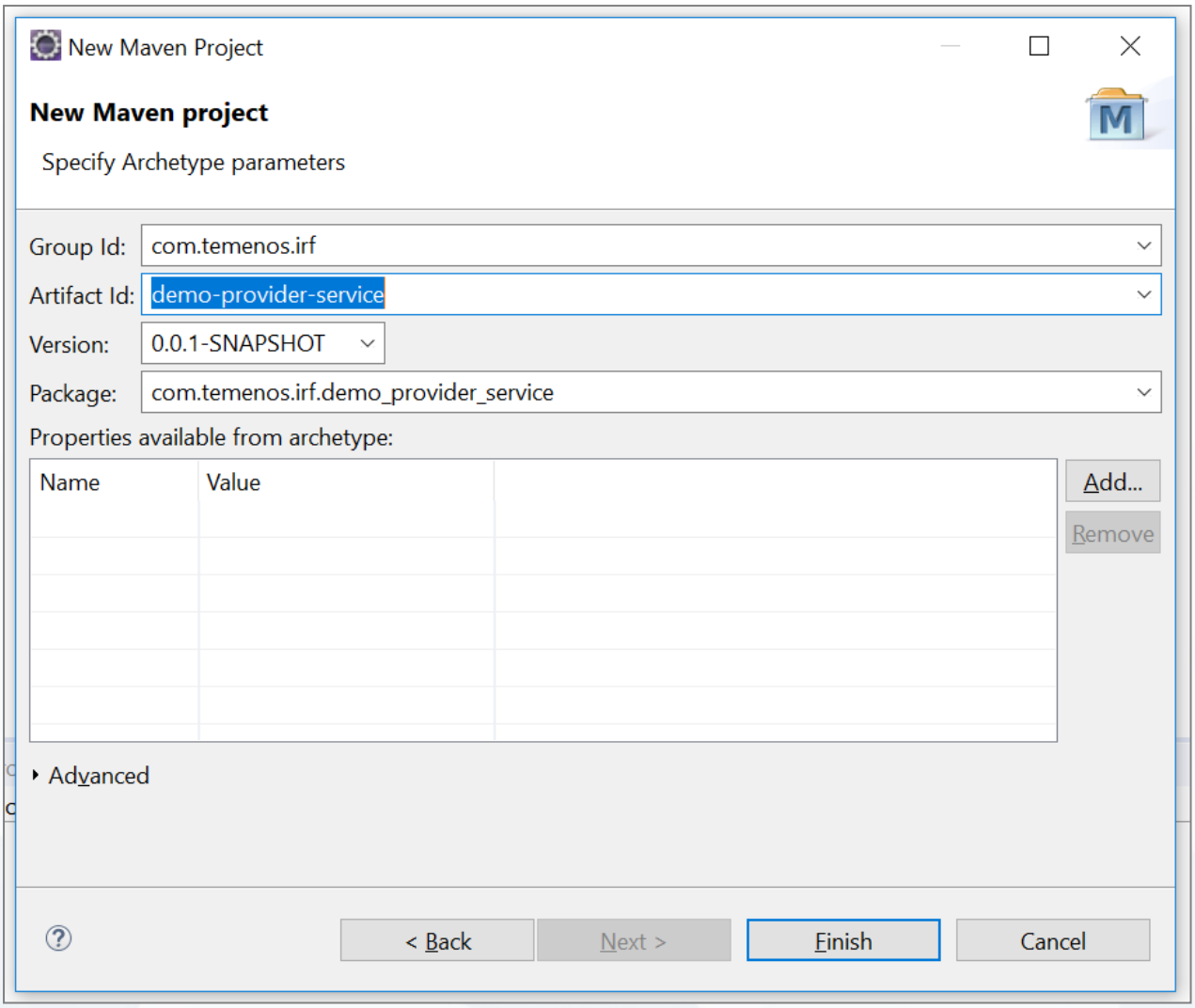
10. Enter project details

    GroupID : com.temenos.irf

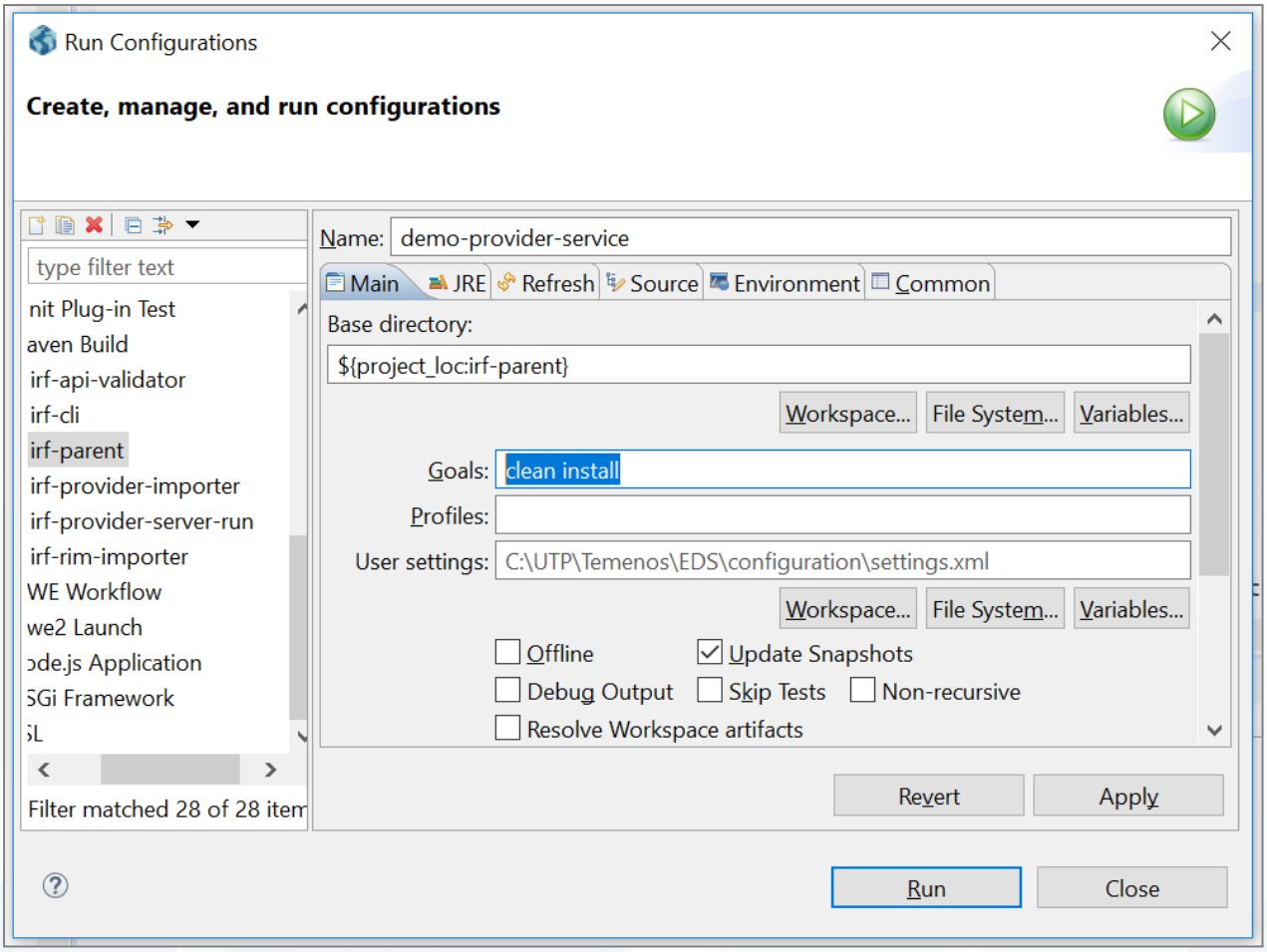
    ArtifactId : demo-provider-container

    Version : 0.0.1-SNAPSHOT [Replace appropriate version]

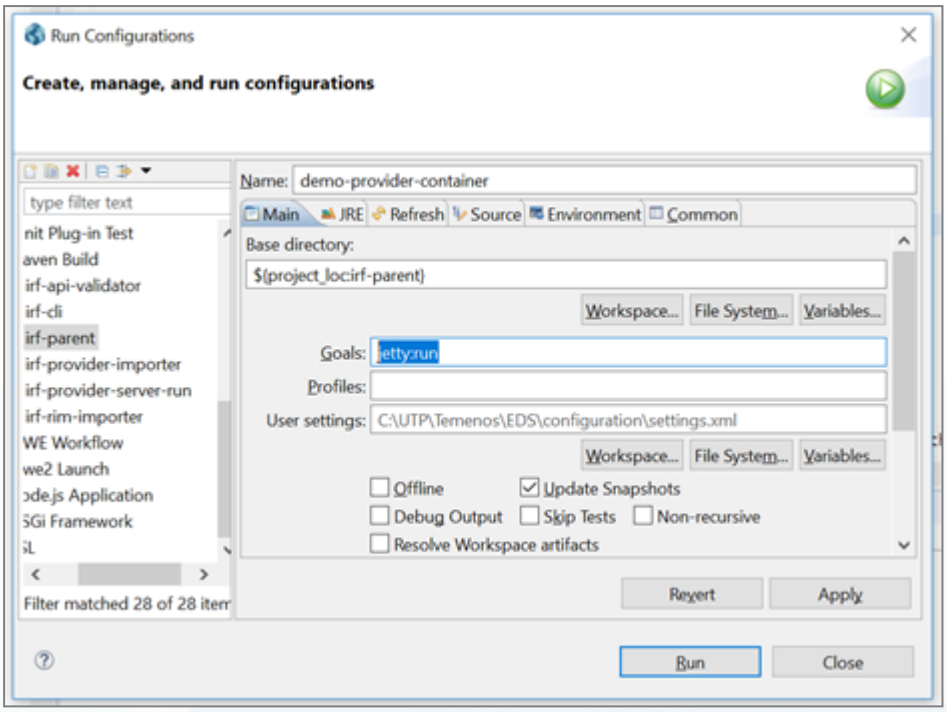
11. Click "**Finish**"



12. Find the irf service container project created
13. Setup Build Configuration
    a. Select the project
    b. Right click on project
    c. Select Run As > Maven Build... [First time only]
    d. Find Configuration Window Popup
    e. Go to Goals
    f. Enter "**clean install**"
    g. Click "**Apply**" & "**Run**"



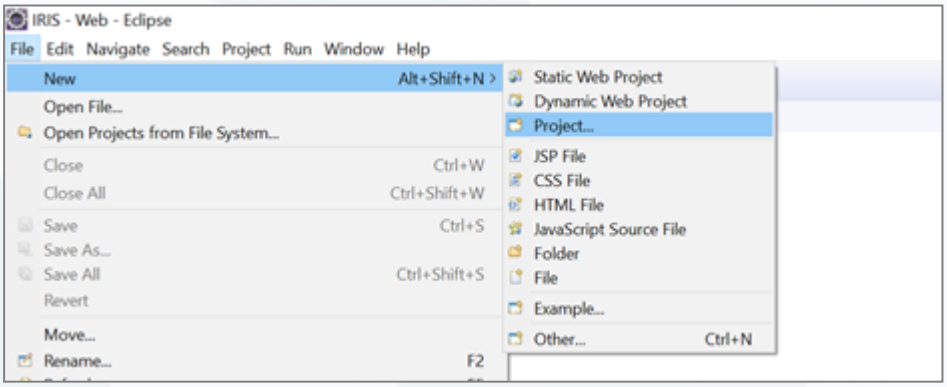
14. Setup Run Configuration
    a. Select the project
    b. Right click on project
    c. Select Run As > Maven Build... [First time only]
    d. Find Configuration Window Popup
    e. Go to Goals
    f. Enter "**jetty:run**"
    g. Click "**Apply**" & "**Run**"



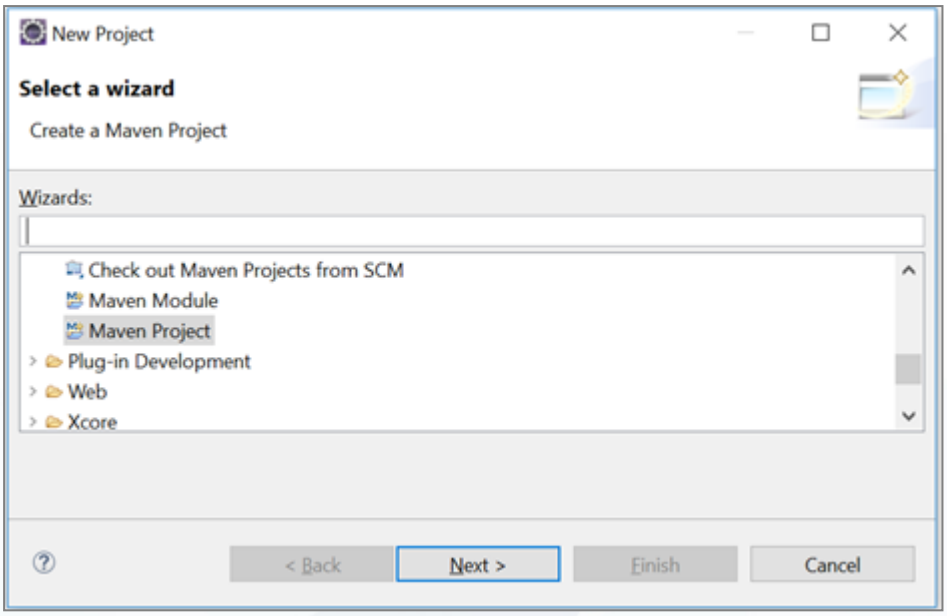
15. Right click and Select Run As > Maven Build
16. Check Eclipse or EDS Console Window for Message "**Jetty Started...**"

How to create IRF Service Project

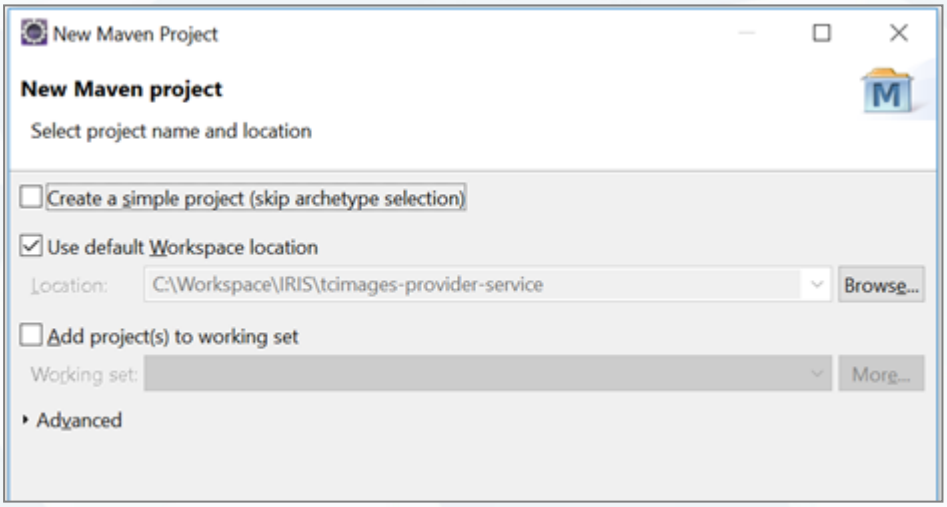
1. Open Eclipse or EDS
2. Click File > New Project



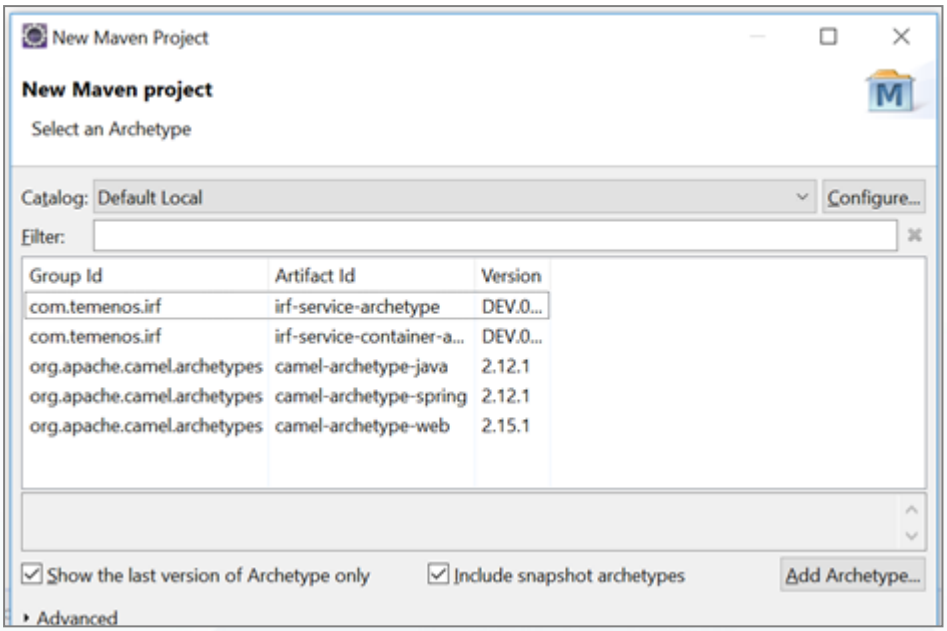
3. Select Maven Project
4. Click "**Next**"



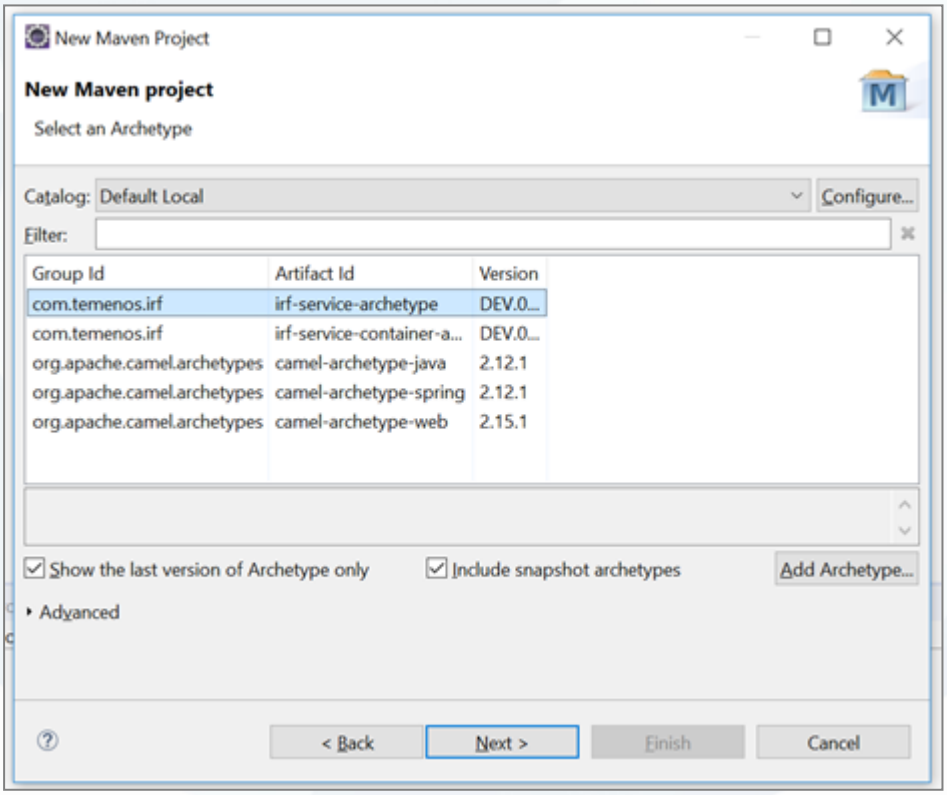
5. Uncheck "**Create a simple project**"
6. Click "**Next**"



7. Check "**Include snapshot archetypes**"



8. Select Item with Artifact Id "**irf-service-archetype**"
9. Click "**Next**"



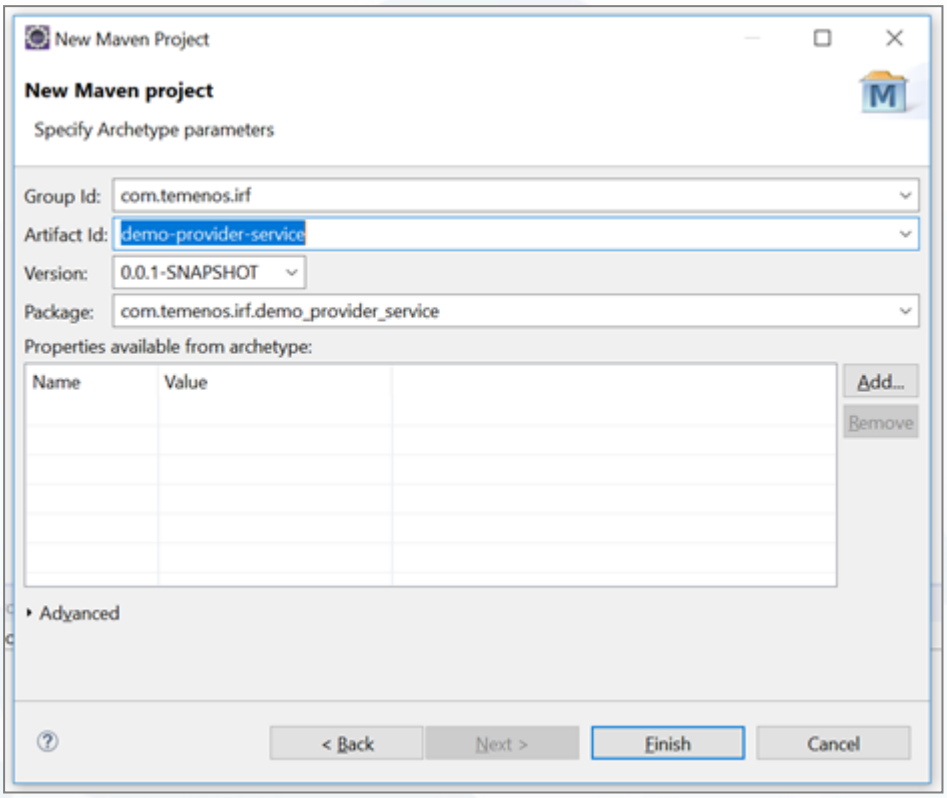
10. Enter project details

    GroupID : com.temenos.irf

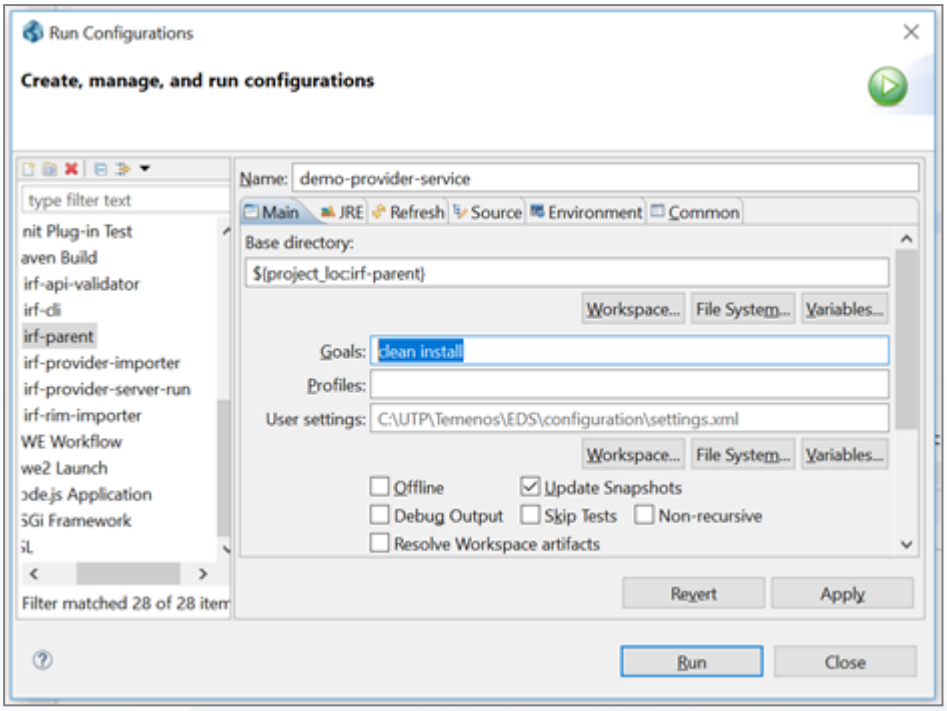
ArtifactId : demo-provider-service

Version : 0.0.1-SNAPSHOT [Replace appropriate version]

11. Click "**Finish**"



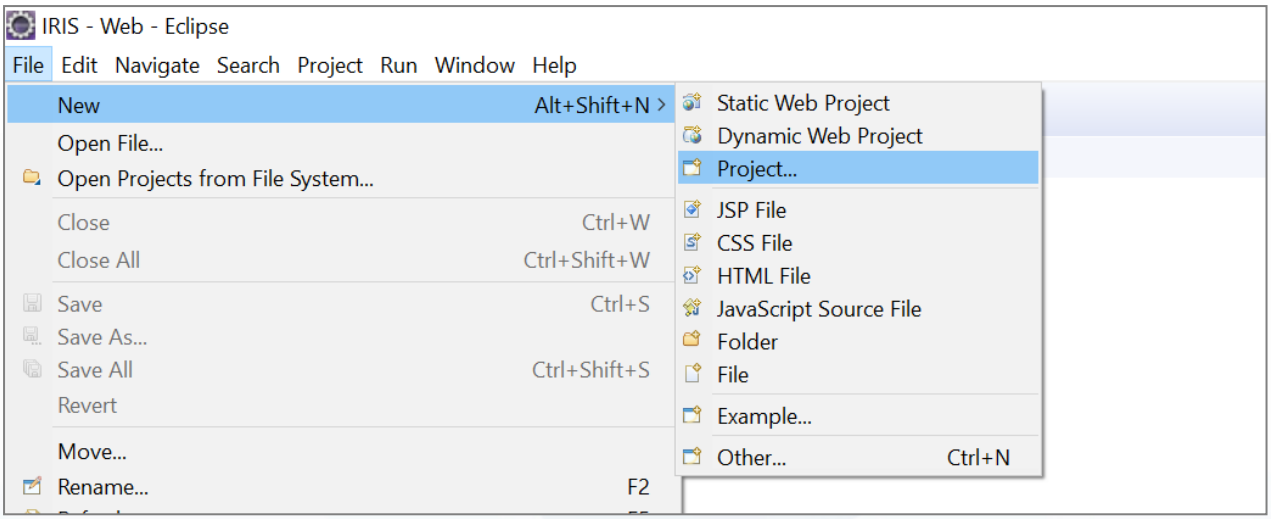
12. Find the irf service container project created
13. Setup Build Configuration
    a. Select the project
    b. Right click on project
    c. Select Run As > Maven Build... [First time only]
    d. Find Configuration Window Popup
    e. Go to Goals
    f. Enter "**clean install**"
    g. Click **Apply** and **Run**.



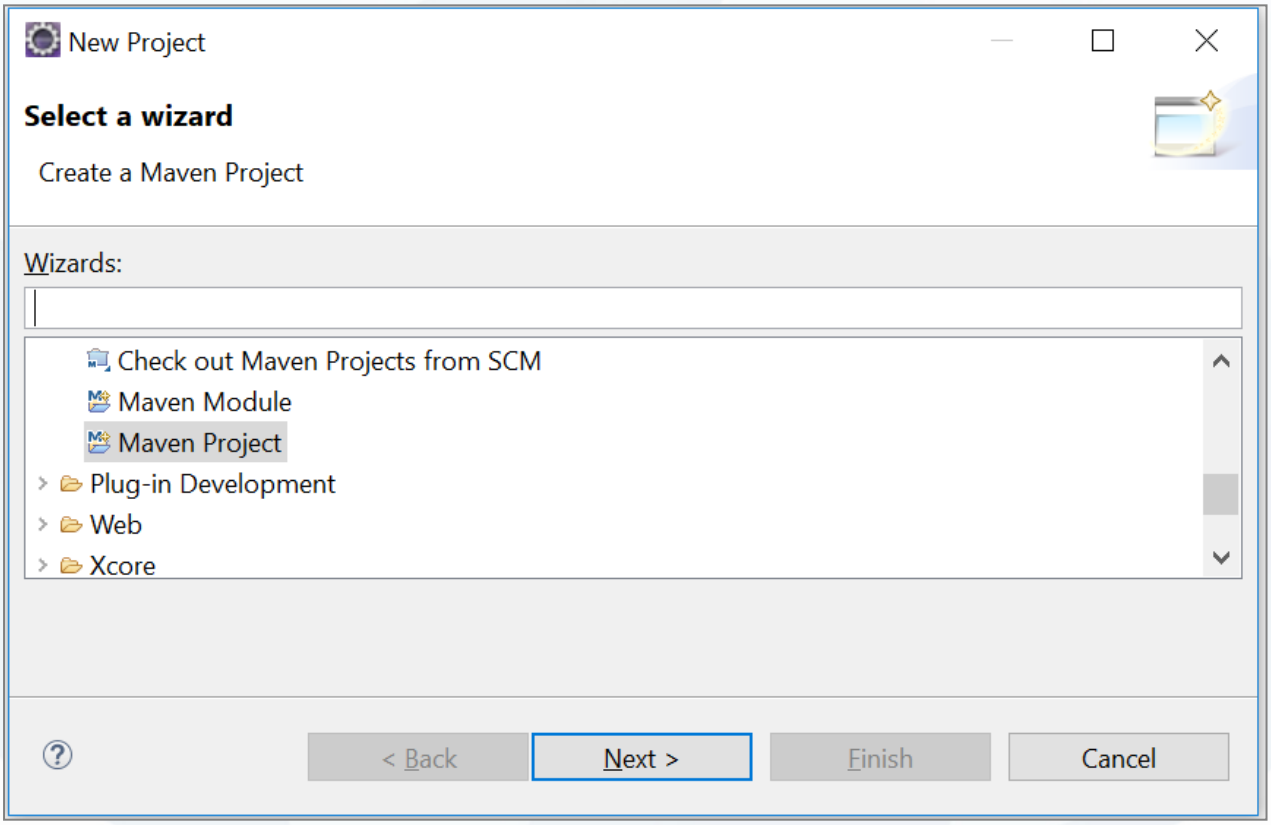
14. Reference IRF Service Project in IRF Service Container Project
    a. Open POM XML
    b. Copy the Artifact Details
    c. Go to IRF Service Contained Project
    d. Open POM XML
    e. Add an artifact reference under references
    f. Paste the details
        i. GroupId
        ii. ArtefactId
        iii. Version
    g. Save POM XML (Both Service and container Projects)
15. Right Click and Select Run As > Maven Build
16. Check Eclipse or EDS Console Window for Message "**Jetty Restarted...**"

How to add IRF Archetypes

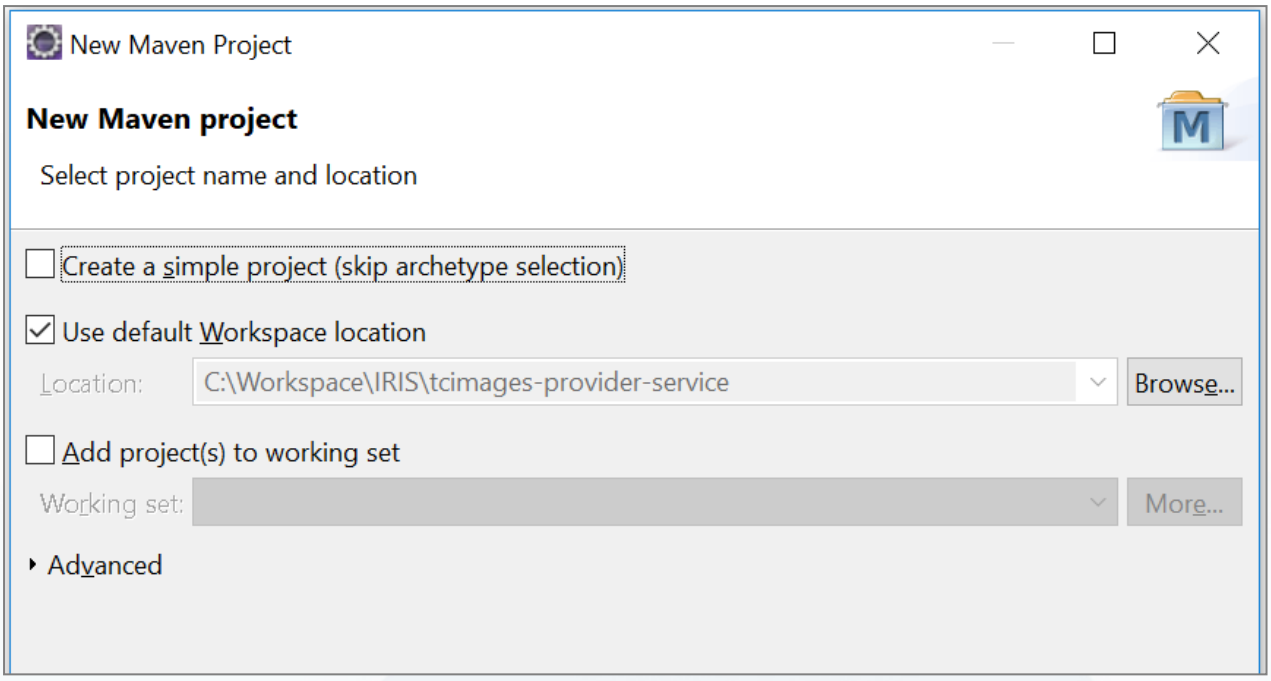
1. Open Eclipse or EDS
2. Click File > New Project



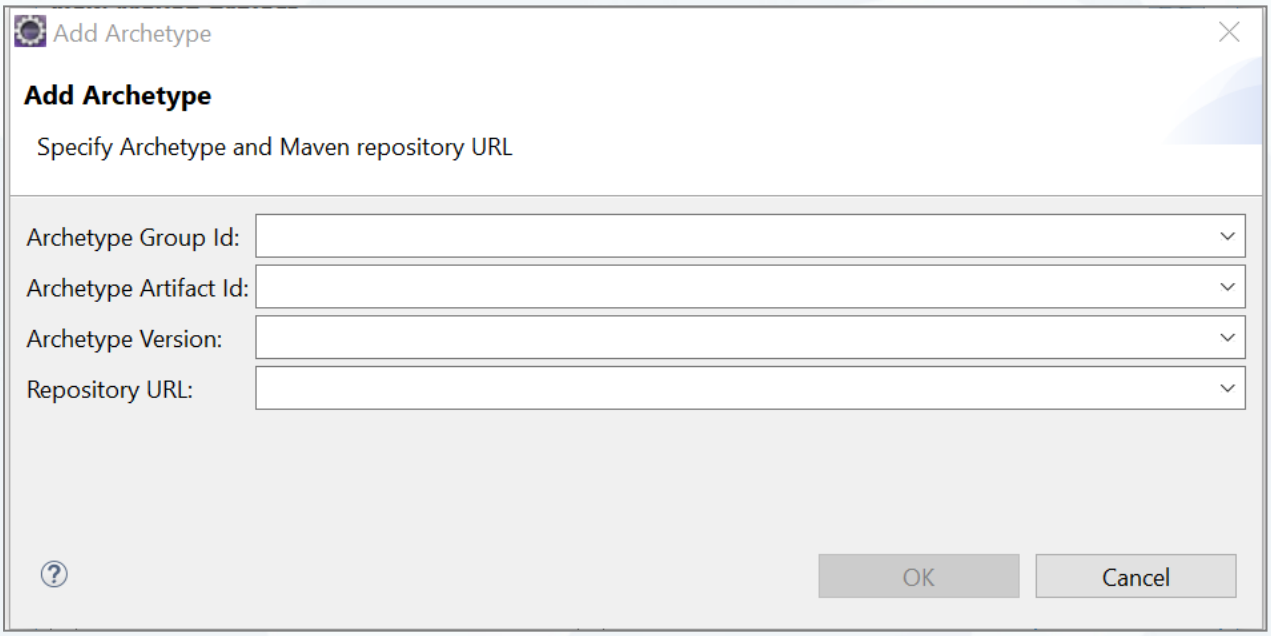
3. Select Maven Project
4. Click **"Next"**



5. Uncheck **"Create a simple project"**
6. Click **"Next"**

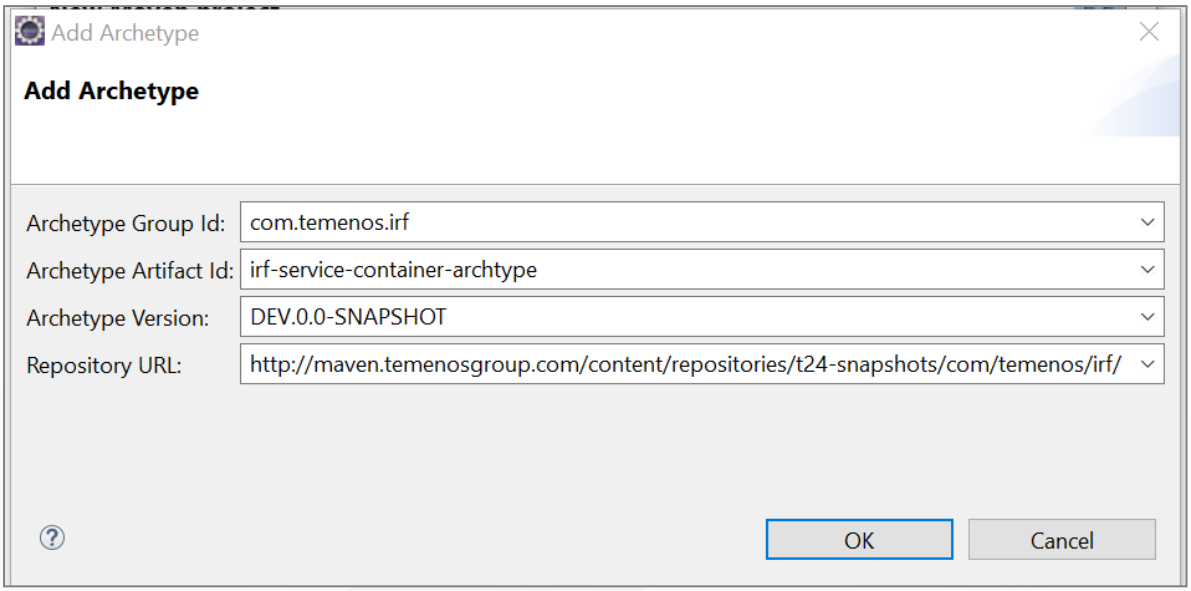


7. Click **"Add Archetype..."**



8. Provide Archetype details

a. IRF Service Container Project Archtype



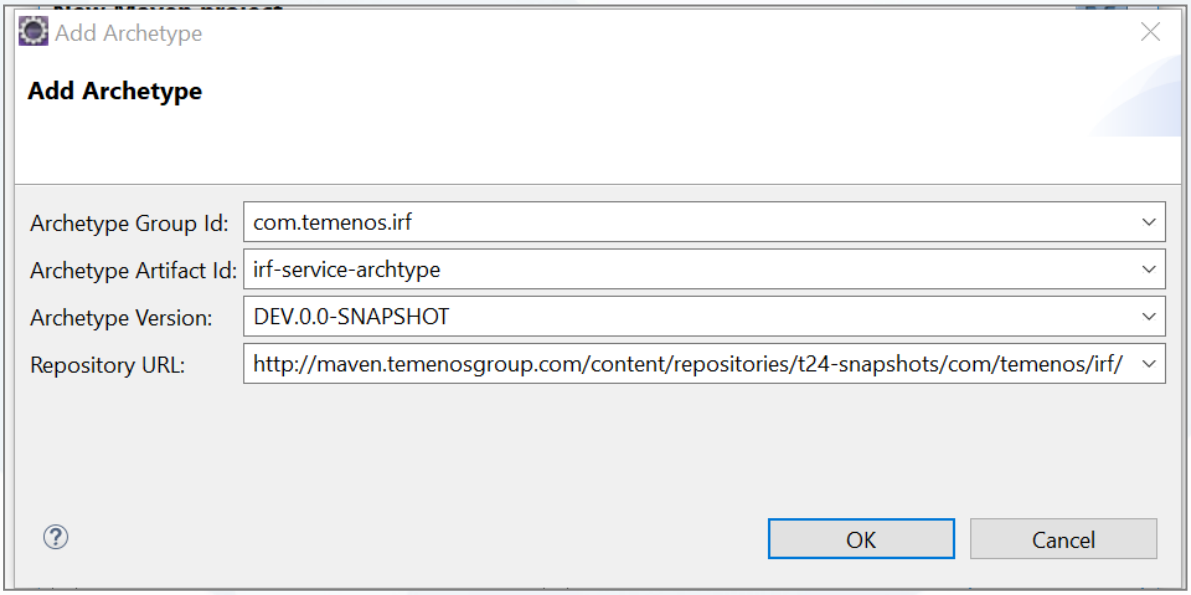
GroupID : com.temenos.irf

ArtifactId : irf-service-container-archtype

Version : DEV.0.0-SNAPSHOT [Replace appropriate version]

URL: http://maven.temenosgroup.com/content/repositories/t24-snapshots/com/temenos/irf/

b. IRF Service Project Archtype



GroupID : com.temenos.irf

ArtifactId : irf-service-archetype

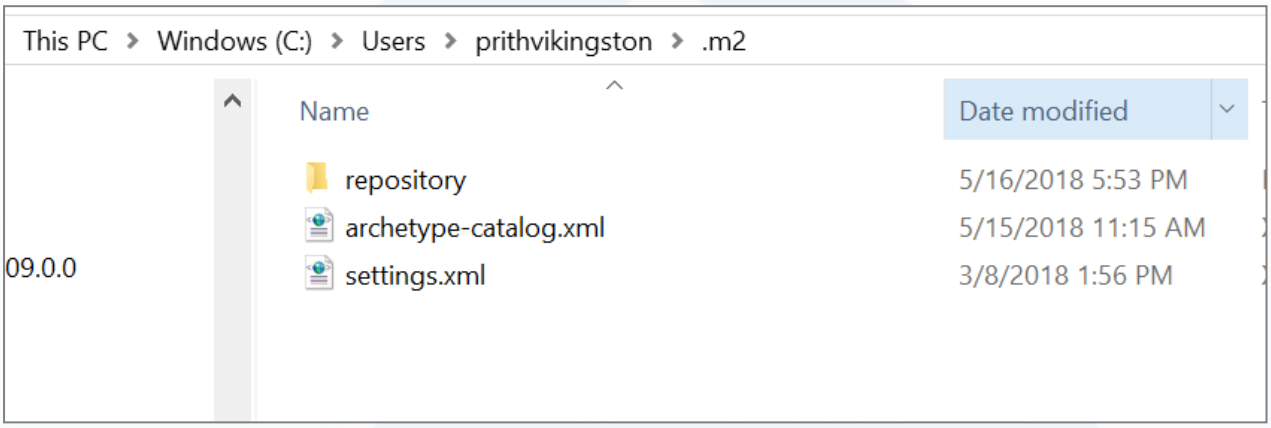
Version : DEV.0.0-SNAPSHOT [Replace appropriate version]

URL : http://maven.temenosgroup.com/content/repositories/t24-snapshots/com/temenos/irf/

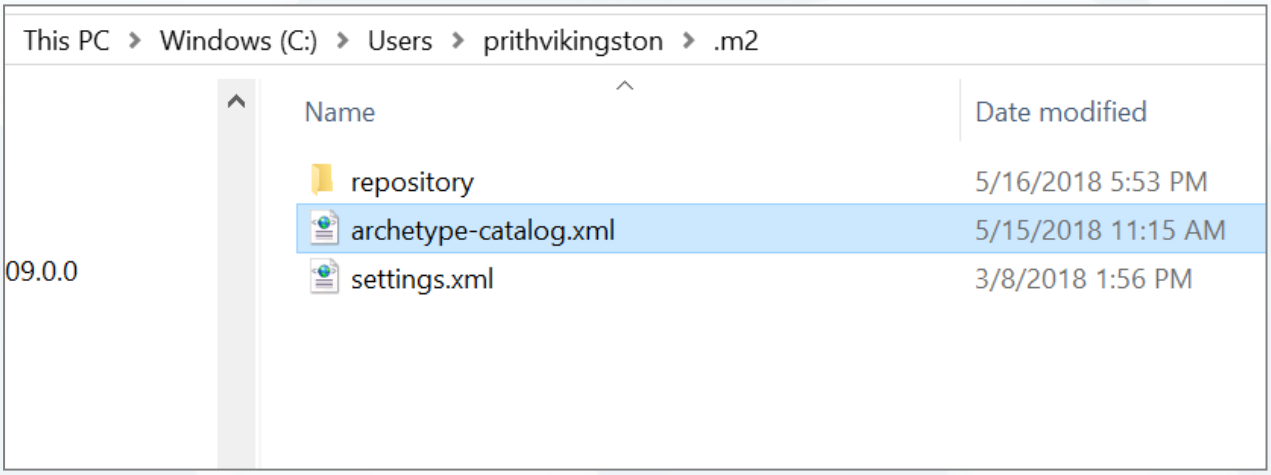
9. Click "OK" to add archetype

Alternatively

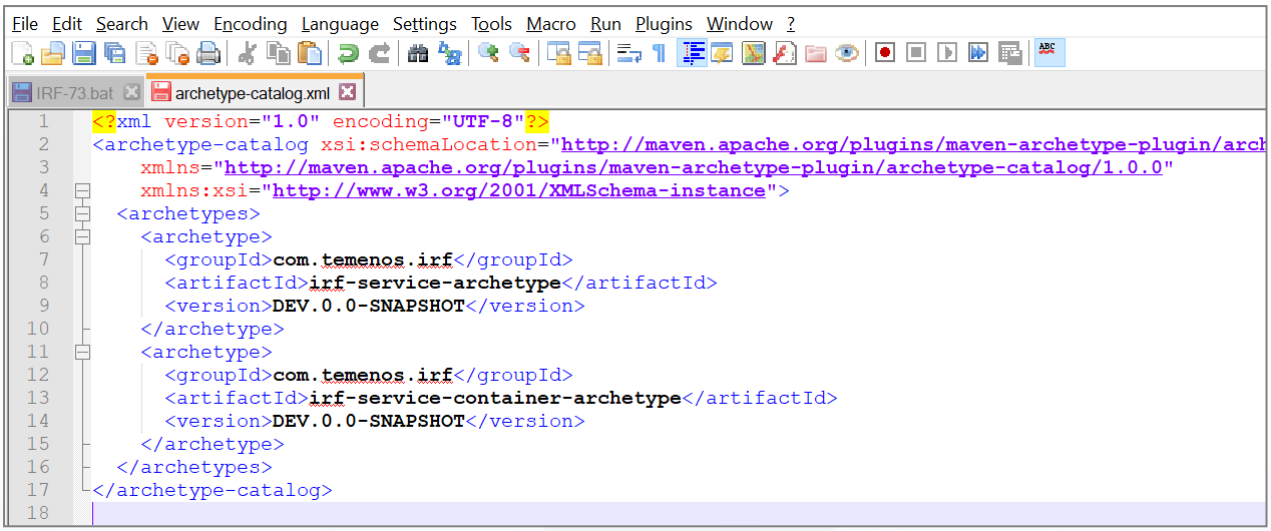
1. >Browse to **"C:\Users\<UserName>\.m2"** directory of maven



2. Find and Open **"archetype-catalog.xml"**



3. Add the below details between <archetypes> ... </archetypes>

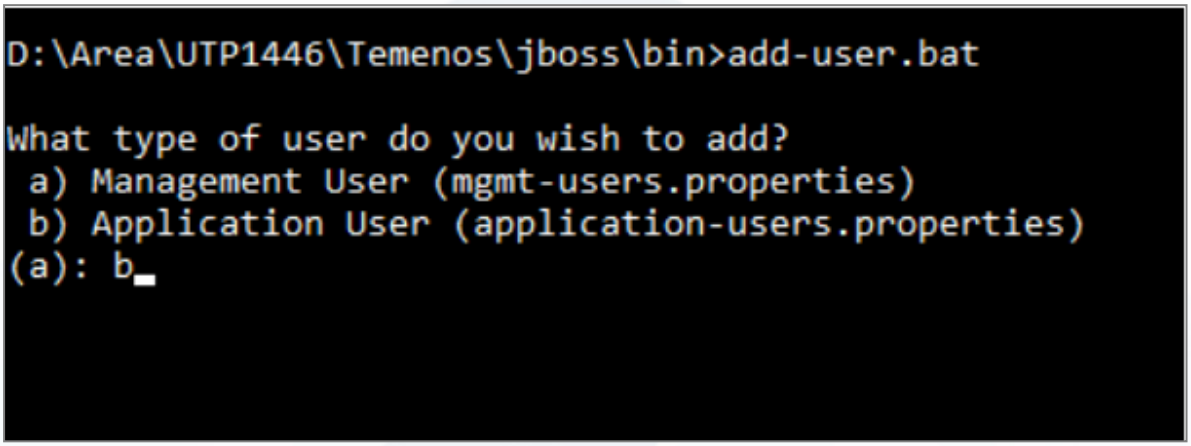



```
<?xml version="1.0" encoding="UTF-8"?>

<archetype-catalog

xsi:schemaLocation="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-
catalog/1.0.0 http://maven.apache.org/xsd/archetype-catalog-1.0.0.xsd"

xmlns="http://maven.apache.org/plugins/maven-archetype-plugin/archetype-catalog/1.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

<archetypes>

<archetype>

<groupId>com.temenos.irf</groupId>

<artifactId>irf-service-archetype</artifactId>

<version>DEV.0.0-SNAPSHOT</version>

</archetype>

<archetype>

<groupId>com.temenos.irf</groupId>

<artifactId>irf-service-container-archetype</artifactId>

<version>DEV.0.0-SNAPSHOT</version>

</archetype>

</archetypes>

</archetype-catalog>
```

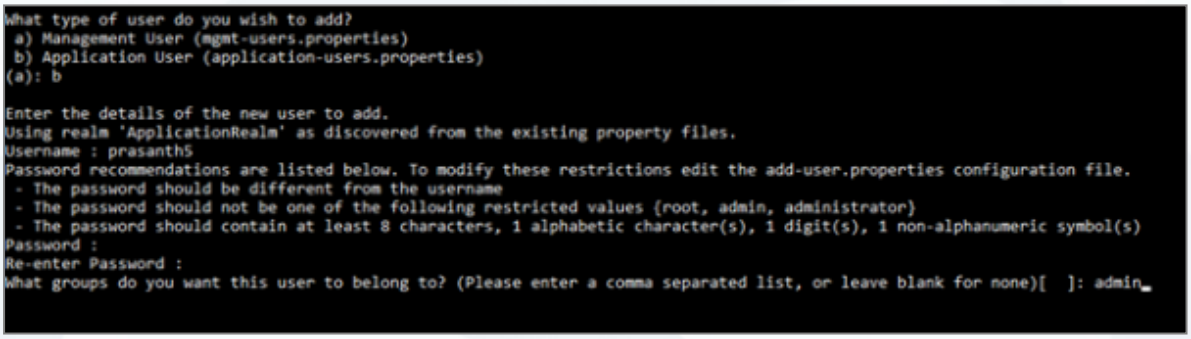
4. Save the file

How to configure RimImporter to communicate Remote T24 Server

1. Create a new "Application User" in JBoss (Remote Server i.e T24Server)
    a. Go to <JBOSS_HOME>/bin/add-user.bat



    b. Select option '**b**' to start creating an "**Application User**"
    c. Enter your username and password as per instruction



    d. Enter your group as "**admin**"

    e. Enter "**No**" for server-server EJB call

2. Configure the RimImporter Importer.properties

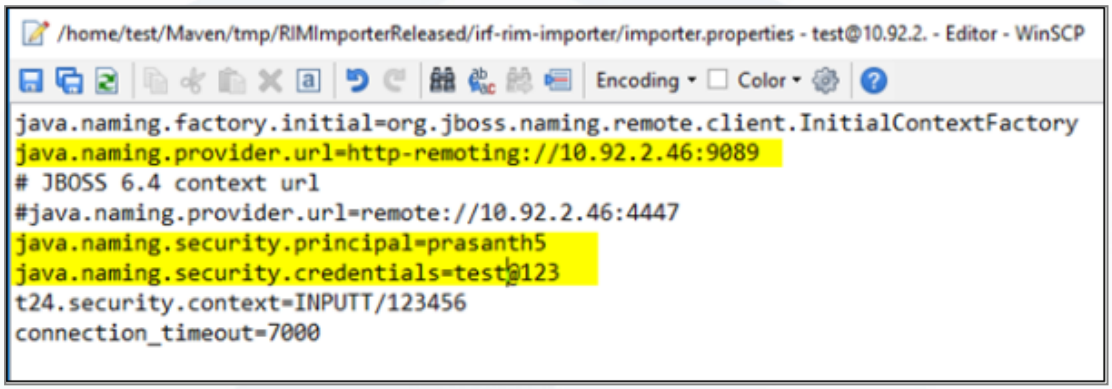
a. Go to irf-rim-importer directory

b. Open "importer.properties" configuration file

c. Update the below section details

i. url in java.naming.provider

ii. username and password in java.naming.security

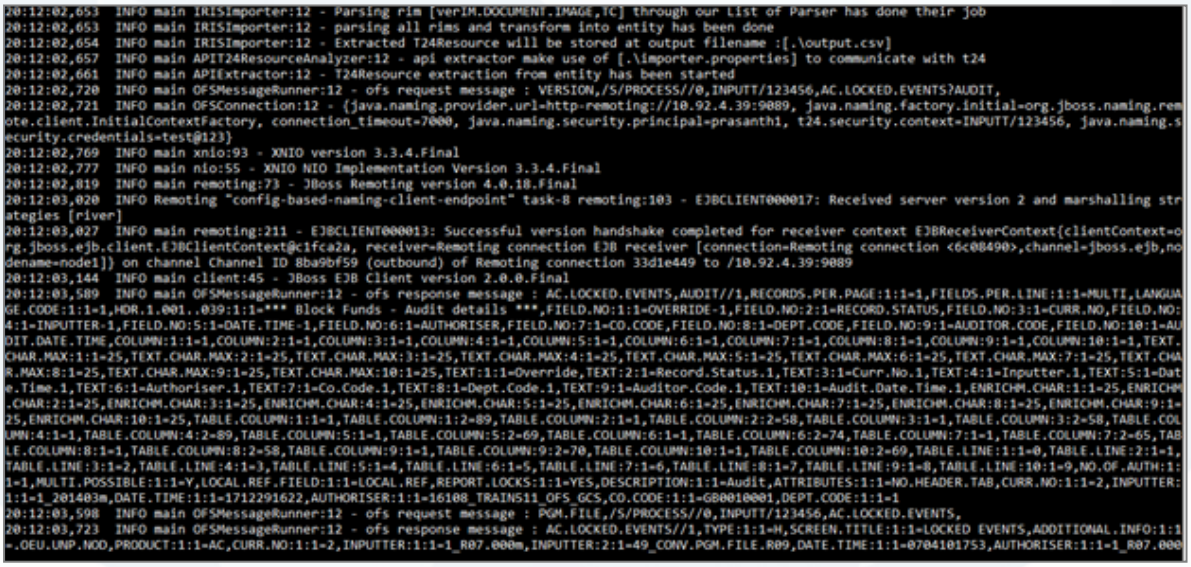


3. Verify the Configuration and Setup

a. Go to irf-rim-importer directory

b. Run 1.extract-api.bat to extract the API's from configured RIMS or Location

c. Ensure that No Errors on Connection



4. Troubleshoot after the configuration (Just in case of cross platform machines)

Issue : ActiveMQNotConnectedException[errorType=NOT_CONNECTED message=AMQ119007: Cannot connect to server(s). Tried with all available servers.]

Solution: Add outbound socket with actual IP and mention the binding name on http-connector

```
<socket-binding-group name="standard-sockets" default-interface="public" port-offset="${jboss.socket.binding.port-offset:0}">
<outbound-socket-binding name="jms-http">
<remote-destination host="10.92.2.46" port="${jboss.http.port:8080}"/>
</outbound-socket-binding>
    <socket-binding name="management-http" interface="management" port="${jboss.management.http.port:9990}"/>
    <socket-binding name="management-https" interface="management" port="${jboss.management.https.port:9993}"/>
    <socket-binding name="remoting" port="4447"/>
    <socket-binding name="ajp" port="${jboss.ajp.port:8009}"/>
    <socket-binding name="http" interface="public" port="${jboss.http.port:8080}"/>
    <socket-binding name="https" port="${jboss.https.port:8443}"/>
    <socket-binding name="iiop" interface="unsecure" port="3528"/>
    <socket-binding name="iiop-ssl" interface="unsecure" port="3529"/>
    <socket-binding name="txn-recovery-environment" port="4712"/>
    <socket-binding name="txn-status-manager" port="4713"/>
    <outbound-socket-binding name="mail-smtp">
        <remote-destination host="localhost" port="25"/>
    </outbound-socket-binding>
</socket-binding-group>
```

```
<subsystem xmlns="urn:jboss:domain:messaging-activemq:1.0">
    <server name="default" persistence-enabled="false">
        <security enabled="false"/>
        <security-setting name="#">
            <role name="guest" delete-non-durable-queue="true" create-non-durable-queue="true" consume="true" send="true"/>
        </security-setting>
        <address-setting name="#" message-counter-history-day-limit="10" page-size-bytes="2097152" max-size-bytes="10485760"
        expiry-address="jms.queue.ExpiryQueue" dead-letter-address="jms.queue.DLQ"/>
        <http-connector name="http-connector" endpoint="http-acceptor" socket-binding="jms-http"/>
        <http-connector name="http-connector-throughput" endpoint="http-acceptor-throughput" socket-binding="http">
            <param name="batch-delay" value="50"/>
        </http-connector>
        <in-vm-connector name="in-vm" server-id="0"/>
        <http-acceptor name="http-acceptor" http-listener="default"/>
        <http-acceptor name="http-acceptor-throughput" http-listener="default">
            <param name="batch-delay" value="50"/>
            <param name="direct-deliver" value="false"/>
        </http-acceptor>
```

RIMImporter Configuration Files

1. **importer.properties**

   The file has all configuration required to setup the rim importer utility. The properties file contains the connection details required to establish connection with JBoss and T24Server. Below are some of the connection parameters

   #Connection Credentials

   java.naming.security.principal=INPUTT

   java.naming.security.credentials=123456

   t24.security.context=INPUTT/123456

   #Connection Timeout

   connection_timeout=7000

   > Note: By default this file might not be available and will be picked using classpath resource. In case you differ from the default connection, it can be edited by copying the above details and creating a file with importer.properties with updated details and save. The file shall and must be located in the same directory as the batch files.

2. **domain.properties**

The file is used to categorize application list into domains like party, retail, holdings, etc. The domains are used to follow the convention of IRISX of URL resource pattern of \api\version\domain\resource\{property}\subresource\{property}. Ensure that all the applications are classified into respective domains so that the url formation is done in compliance to the IRISX conventions meeting REST Standards. Another key information that the domains are also used to refer the right vocabulary file for its resource and properties for validation. Below are some Applications and Meta information are classified into there respective domains.

party*=IM.DOCUMENT.IMAGE:IM.DOCUMENT.UPLOAD

meta=GETMetadata:T24FieldMetadata

> Note: (*) asterisks indicates that domain is default to any Application that is not classified or categorized into its respective domain, in above example "party" is default domain. The application lists are separated by (:) semicolon. The file shall and must be located in the same directory as the batch files.

3. **verb.Properties**

   The file is used to map the IRIS Legacy command with corresponding IRISX methods. The methods are used to determine the type of the http operation and name of the operation as per the IRISX conventions. The file shall and must be located in the same directory as the batch files. Below mentioned are few entries from the verb properties file

   GetEntities=get

   GetEntity=get

   CreateEntity=create

   DeleteEntity=delete

   GET=get

   POST=create

   PUT=update

   DELETE=delete

4. vobcabulary-<domain>.json

   The file is used to catalog all domain specific entries like the domain, resource and property associated with a URL or exchange parameters keeping the entire exchange well known and secured. The vocabulary is part of IRISX conventions, any field(s) and criteria(s) exchange must meet the vocabulary. So owners can either update the vocabulary with artifact (VERSION\ENQUIRY) details or update the artifact based on the vocabulary. The recommended approach is to first check the vocabulary for the details and update the artifact if not meeting the expectation the vocabulary shall be updated in accordance. Below mentioned are few entries from domain specific vocabulary file indicating resource and property type resources.

```
{
```

```
"key" : "transactionType",

"description" : "transactionType",

"plural" : "transactionTypes",

"insteadOf" : [ ],

"links" : [ ],

"usage" : [ ],

"entryType" : "resource",

"dataType" : "String",

"label" : null,

"generated" : false

}, {

"key" : "transactionTotal",

"description" : "transactionTotal",

"plural" : "transactionTotals",

"insteadOf" : [ ],

"links" : [ ],

"usage" : [ ],

"entryType" : "property",

"dataType" : "String",

"label" : null,

"generated" : false

},
```


RIMImporter FAQ'S

1. Where to get the latest release pack of RimImporter from?

   Click Download to download the latest rimimporter release package from the Temenos Maven Repository

   Link : http://maven.temenosgroup.com/content/groups/temenos/com/temenos/irf/irf-rim-importer-packager/DEV.0.0-SNAPSHOT/

2. How to configure RimImporter for Remote T24Server?

   Refer : How to configure RimImporter to communicate Remote T24 Server?

Appendix

T24 Enquiry

Making an ENQUIRY suitable as an API provider

1. Enquiry id must begin as XX.API (where XX is the module) and end with the full version number, 1.0.0 and should be named as collections, e.g. AC.API.ACCOUNTS.TRANSACTIONS.1.0.0
2. Turn SEL.LABEL into API style labels
3. remove HEADER
4. Turn FIELD.LABEL into API style labels
5. Use FIELD.DISPLAY.TYPE for data type (Alphanumeric / amount, date
6. remove any static labels - where OPERATION is a literal and COLUMN is set
7. Set ATTRIBS to ''
8. Remove drill down information (ENQUIRY.NAME,SEL.CRIT,LABEL.FIELD,NXT.DESC)
9. Set DESCRIPT to be the operation, e.g. getAccountBalances

API style labels: remove punctuation and spaces, turn into camelCase (My Label -> myLabel, MY.FIELD.NAME ->myFieldName). Also use full words (bal ->balance) so Cleared Bal -> clearedBalance

Overview

This section explains how the T24 ENQUIRY configuration affects the structure of the JSON that is produced.

ENQUIRY Field	Usage in API	Description
GB Descript	Operation	Used as the name of the operation in the API, should conform to the following pattern: verbCollectionResource e.g. GetAccountTransactions CreateAccountTransfer

ENQUIRY Field	Usage in API	Description
GB Field Lbl	Label to be used in JSON	The label to be used in the JSON response **Note:** • The text label provided must match with the entries in the vocabulary • Add new text label in IRIS R18 Vocabulary if doesn't exists else reuse closest matching entry as text label
COMMENTS	Group labels	If KEY is not set, then there the grouping & sub grouping can be configured as follows : (In this case all fields should be set to S mode) 1) Just the group name (no delimiter) eg : addresses 2) Just the group name and the group delimiter eg: addresses PIPE 3) the group name, group delimiter, sub-group name, sub-group delimiter eg : addresses PIPE addressLines STAR supported delimiters are: **SPACE**, **ASTERISK**, **PIPE**, **SEMICOLON**, **COLON**, **DOLLAR**, **TILDE**, **COMMA**. If KEY is set then there the grouping & sub grouping can be configured as follows : (In this case , multi-value & sub-value fields should be set to M mode , no need to set the delimiters) 1) group name. eg: addresses 2) group name and sub-group name eg: addresses addressLines
FIELD.DISPLAY.TYPE	Data type of the field.	Alphanumeric Date Amount TimeStamp Numeric

The Basics

A standard column based ENQUIRY is rendered as:

ENQUIRY Columns	JSON Response

```
{
        "fieldName": "accoun-
tNumber",
        "columnNumber": "1",
        "lengthMask": "16L",
        "columnName": "accountId"
    },
    {
        "fieldName": "CURRENCY",
        "columnNumber": "2",
        "dataType": "ALPHANUMERIC",
        "lengthMask": "3L",
        "columnName": "currency"
    },
    {
        "fieldName": "DAMOUNT",
        "columnNumber": "3",
        "dataType": "AMOUNT",
        "columnName": "amount"
    },
    {
        "fieldName": "VALUE.DATE",
        "columnNumber": "4",
        "dataType": "DATE",
        "lengthMask": "11R",
        "columnName": "valueDate"
    },
    {
        "fieldName": "BOOKING.DATE",
        "columnNumber": "5",
        "dataType": "DATE",
```

```
{
  "header": {
    "audit": {
      "T24_time": 638,
      "parse_time": 16
    },
    "page_start": 0,
    "page_token": "3517ee11-09c7-4c49-a8f2-
6691e354d623",
    "total_size": 39,
    "page_size": 50
  },
  "body": [
    {
      "reference": "FT170803R9LY",
      "accountId": "10944",
      "amount": 200000,
      "currency": "USD",
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
    {
      "reference": "FT17080GCFJJ",
      "accountId": "10944",
      "amount": 150000,
      "currency": "USD",
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
```

"lengthMask": "11R",
"columnName": "bookingDate"
},
{
"fieldName": "TRANS.REFERENCE",
"columnNumber": "6",
"dataType": "ALPHANUMERIC",
"lengthMask": "25L",
"columnName": "reference"
}	

Adding Data to the Body

Setting the DISPLAY.BREAK to ONCE creates a separate row in the response. Note the row count remains unchanged.

ENQUIRY Columns	JSON Response

```
{
        "displayBreak": "ONCE",
        "fieldName": "accountNumber",
        "columnNumber": "1",
        "lengthMask": "16L",
        "columnName": "accountId"
    },
    {
        "fieldName": "CURRENCY",
        "columnNumber": "2",
        "dataType": "ALPHANUMERIC",
        "lengthMask": "3L",
        "columnName": "currency"
    },
    {
        "fieldName": "DAMOUNT",
        "columnNumber": "3",
        "dataType": "AMOUNT",
        "columnName": "amount"
    },
    {
        "fieldName": "VALUE.DATE",
        "columnNumber": "4",
        "dataType": "DATE",
        "lengthMask": "11R",
        "columnName": "valueDate"
    },
    {
        "fieldName": "BOOKING.DATE",
        "columnNumber": "5",
        "dataType": "DATE",
```

```
{
  "header": {
    "audit": {
      "T24_time": 465,
      "parse_time": 7
    },
    "page_start": 0,
    "page_token": "9f8ea698-b52b-438e-817f-
16e7552191c0",
    "total_size": 40,
    "page_size": 50
  },
  "body": [
    {
      "accountId": "10944"
    },
    {
      "reference": "FT170803R9LY",
      "amount": 200000,
      "currency": "USD",
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
    {
      "reference": "FT17080GCFJJ",
      "amount": 150000,
      "currency": "USD",
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
```

"lengthMask": "11R",
"columnName": "bookingDate"
},
{
"fieldName": "TRANS.REFERENCE",
"columnNumber": "6",
"dataType": "ALPHANUMERIC",
"lengthMask": "25L",
"columnName": "reference"
}	

Adding Data to the Header

Setting the DISPLAY.BREAK to END and SECTION to FOOTER creates a data element in the header section of the response. Note the row count remains unchanged.

ENQUIRY Columns	JSON Response

```
{
    "displayBreak": "ONCE",
    "fieldName": "accountNumber",
    "columnNumber": "1",
    "lengthMask": "16L",
    "columnName": "accountId"
},
{
    "displayBreak": "END",
    "fieldName": "CURRENCY",
    "columnNumber": "2",
    "dataType": "ALPHANUMERIC",
    "section": "FOOTER",
    "lengthMask": "3L",
    "columnName": "currency"
},
{
    "fieldName": "DAMOUNT",
    "columnNumber": "3",
    "dataType": "AMOUNT",
    "columnName": "amount"
},
{
    "fieldName": "VALUE.DATE",
    "columnNumber": "4",
    "dataType": "DATE",
    "lengthMask": "11R",
    "columnName": "valueDate"
},
{
    "fieldName": "BOOKING.DATE",
```

```
{
  "header": {
    "data": {
      "currency": "USD"
    },
    "audit": {
      "T24_time": 468,
      "parse_time": 9
    },
    "page_start": 0,
    "page_token": "ec00f96f-15a5-40ff-8a73-
4ca22c89b967",
    "total_size": 39,
    "page_size": 50
  },
  "body": [
    {
      "accountId": "10944"
    },
    {
      "reference": "FT170803R9LY",
      "amount": 200000,
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
    {
      "reference": "FT17080GCFJJ",
      "amount": 150000,
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    }
```

"columnNumber": "5",
"dataType": "DATE",
"lengthMask": "11R",
"columnName": "bookingDate"
},
{
"fieldName": "TRANS.REFERENCE",
"columnNumber": "6",
"dataType": "ALPHANUMERIC",
"lengthMask": "25L",
"columnName": "reference"
}	

Changing the SECTION to HEADER for accountId removes the added row, and moves the data to the header section. NB if multiple data elements are present, i.e. no DISPLAY.BREAK is set, the last element will be added to the header. Note the row count remains unchanged.

ENQUIRY Columns	JSON Response

```
{
        "displayBreak": "ONCE",
        "fieldName": "accoun-
tNumber",
        "columnNumber": "1",
        "section": "HEADER",
        "lengthMask": "16L",
        "columnName": "accountId"
    },
    {
        "displayBreak": "END",
        "fieldName": "CURRENCY",
        "columnNumber": "2",
        "dataType": "ALPHANUMERIC",
        "section": "FOOTER",
        "lengthMask": "3L",
        "columnName": "currency"
    },
    {
        "fieldName": "DAMOUNT",
        "columnNumber": "3",
        "dataType": "AMOUNT",
        "columnName": "amount"
    },
    {
        "fieldName": "VALUE.DATE",
        "columnNumber": "4",
        "dataType": "DATE",
        "lengthMask": "11R",
        "columnName": "valueDate"
    },
```

```
{
  "header": {
    "data": {
      "accountId": "10944",
      "currency": "USD"
    },
    "audit": {
      "T24_time": 2176,
      "parse_time": 4185
    },
    "page_start": 0,
    "page_token": "87a33ba2-968f-4e68-b711-
979ed35adaae",
    "total_size": 39,
    "page_size": 50
  },
  "body": [
    {
      "reference": "FT170803R9LY",
      "amount": 200000,
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    },
    {
      "reference": "FT17080GCFJJ",
      "amount": 150000,
      "bookingDate": "2017-03-21",
      "valueDate": "2017-03-21"
    }
```

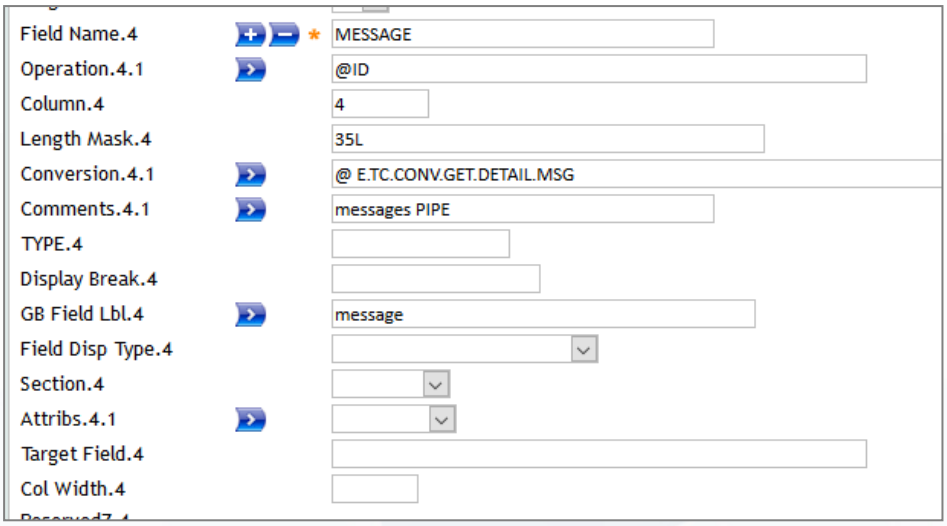
<table>
<tr>
<td>{
"fieldName": "BOOKING.DATE",
"columnNumber": "5",
"dataType": "DATE",
"lengthMask": "11R",
"columnName": "bookingDate"
},
{
"fieldName": "TRANS.REFERENCE",
"columnNumber": "6",
"dataType": "ALPHANUMERIC",
"lengthMask": "25L",
"columnName": "reference"
}</td>
<td></td>
</tr>
</table>

Supporting Multi-Values and Sub-Values

To create a multi-value/sub-value collection group, the COMMENTS field is used to define the collection label, e.g. currencies.

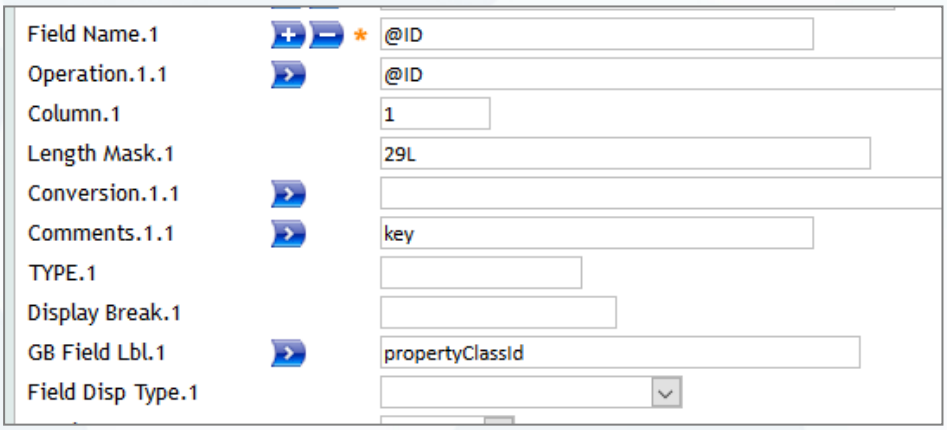
There are two mechanisms to support multi values:

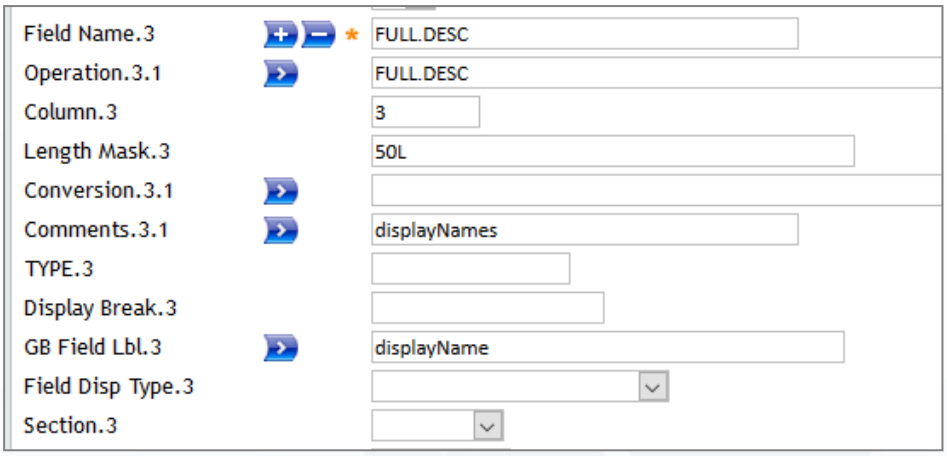
1. SINGLE.MULTI must be set to S and remove any width restrictions in LENGTH.MASK. By default, IRIS will separate multi-values base on a SPACE character (the T24 default) - however some ENQUIRYs are defined to delimit by a pipe character (|). If this is the case, the COMMENTS field is also used to specify the delimiter, hence a multi-value label of currencies used with a pipe delimiter would be specified as currencies PIPE. For sub-values the same must be done, however the COMMENTS section must contain the sub value group name and delimiter separated by a space from the multi-value group. e.g. currencies PIPE rate SPACE.



Currently (201810 build), the supported delimiters are: SPACE,ASTERISK,PIPE,SEMICOLON,COLON,DOLLAR,TILDE,COMMA.

2. Set SINGLE.MULTI to M and remove any width restrictions in LENGTH.MASK. Use the COMMENTS field to define the group name. However, an additional column (which has unique value for each row) must have the COMMENTS marked as either "key" or "record" to indicate the difference between record rows and multi-value rows. For sub-values the same must be done, however the COMMENTS section must contain the sub value group name separated by a space from the multi-value group. e.g. currencies rate.





T24 Version

Overview

This section explains how the T24 Version configuration affects the structure of the JSON that is produced.

The Basics

Version Field	Usage in API	Description
DESCRIPTION	Operation	Used as the name of the operation in the API, should conform to the following pattern: verbCollectionResource e.g. GetAccountTransactions CreateAccountTransfer
TEXT	Label to be used in JSON	The label to be used in the JSON response **Note**: • The text label provided must match with the entries in the vocabulary • Add new text label in IRIS R18 Vocabulary if it doesn't exists else reuse closest matching entry as text label

PROMPT.TEX-T	Group labels	Used to signify that a field is part of a collection (multi-value group) or collection of collections (sub-value group) and to provide the name of the collection.
		
		For example, the fields chargeFor, chargeType and chargeAmount are a multi value set. Setting the PROMPT.TEXT to **charges** would result in the following JSON strcuture:
		
		`"charges": [ { "chargeFor": "string", "chargeType": "string", "chargeAmount": "string" } ]`
		
		Where a sub-value is set, the group names are needed for both the multi value group and the sub value group, separated by a space.
		
		For example, a multi-value set of relatedMsg has a sub-value of timeIn-dicator. Setting the PROMPT.TEXT to relatedMessages for the relatedMsg field and to relatedMessages timeIndicators for timeIn-dicator, would result in the following JSON strcuture:
		
		`"relatedMessages": [ { "timeIndicators": [ { "timeIndicator": "string" } ], "relatedMsg": "string" } ]`
		
		NB If PROMPT.TEXT is not set, the fields are NOT noted as a collection

ATTRIBS	Data type of the field.	Alphanumeric Date Amount
TOOL.TIP	List of allowed values	Comma delimited list of allowed values

List of Important URL's

1. Provider Services catalog

   URL → GET: http://localhost:8080/api/v1.0.0/meta/apis
2. Internal Services catalog

   URL → GET: http://localhost:8080/api/v1.0.0/internal/meta/apis
3. API Status

   URL → GET: http://localhost:8080/api/v1.0.0/meta/apis/status
4. Service Catalog

   URL → GET: http://localhost:8080/api/v1.0.0/meta/apidocs
5. Schema definition

   URL → GET: http://localhost:8080/api/v1.0.0/meta/apidocs/order-service-v1.0.0
6. List of All Enquiries present

   URL → GET: http://localhost:8080/api/v1.0.0/meta/queries
7. List of All Version present

   URL → GET: http://localhost:8080/api/v1.0.0/meta/screens
8. List all fields of a given Enquiry

   URL → GET: http://localhost:8080/api/v1.0.0/meta/queries/{queryId}
9. List all fields of a given Version

   URL → GET: http://localhost:8080/api/v1.0.0/meta/screens/{screenId}
10. List of all the ProductLines

    URL → GET: http://localhost:8080/api/v1.0.0/product/productLines
11. List of all Product and its ProductGroup

    URL → GET: http://localhost:8080/api/v1.0.0/product/products
12. List for a particular productGroup based on product

    URL → GET : http://localhost:8080/api/v1.0.0/product/products/{productId}
13. List of all ProductGroups Associated with the ProductLine

    GET : http://localhost:8080/api/v1.0.0/product/productLines/{productLineId}/productGroups
14. List of all the ActivitiesClass Associated with a particular productLine

    GET: http://localhost:8080/api/v1.0.0/product/productLines/{productLineId}/activities
15. List of all Product Groups URL

    GET: http://localhost:8080/api/v1.0.0/product/productGroups
16. AA Payload based on the Activity and ProductGroupId

URL → GET: http://localhost:8080/api/v1.0.0/product/productGroups/{productGroupId}/activities/{activityId}/payload