**Yugandhar**

**Open Master Data Management (MDM) Hub**

**Code Generation Guide**

Yugandhar Open MDM Hub Release - V1.0.0

Date – 27/12/2017

+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

Copyright [2017] [Yugandhar Open MDM Hub]

Licensed under the Apache License, Version 2.0 (the "License");

you may not use this file except in compliance with the License.

You may obtain a copy of the License at

http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS,

WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

See the License for the specific language governing permissions and limitations under the License.

Contents

[1. Create New Java Project for Code Generation 3](#_Toc502406377)

[2. Hibernate Console Configuration 8](#_Toc502406378)

[3. Create hibernate Configuration file 17](#_Toc502406379)

[4. Create Hibernate Code Generation Configurations 20](#_Toc502406380)

[Set the exporters (Freemarker Templates) 25](#_Toc502406381)

[Common Generators 28](#_Toc502406382)

[ Abstract DO Generator 28](#_Toc502406383)

[ DO Generator 28](#_Toc502406384)

[Data Tables Object Generators 29](#_Toc502406385)

[ ServiceGenerator 29](#_Toc502406386)

[ ComponentGenerator 29](#_Toc502406387)

[ RepositoryGenerator 29](#_Toc502406388)

[ RuleGenerator 29](#_Toc502406389)

[ artifacts 29](#_Toc502406390)

[Reference Data tables Object generators 29](#_Toc502406391)

[ ServiceGenerator\_ref 30](#_Toc502406392)

[ ComponentGenerator\_ref 30](#_Toc502406393)

[ RepositoryGenerator\_ref 30](#_Toc502406394)

[ RuleGenerator\_ref 30](#_Toc502406395)

[ artifacts\_ref 30](#_Toc502406396)

[5. Editing the Reverse Engineering xml and Generating Code 44](#_Toc502406397)

# About Yugandhar Open MDM Hub Project

Master Data Management came a long way in last decade or so. There are currently more than 20 MDM solutions catering to various specializations of MDM like Customer Data Integration (CDI), Product Information Management (PIM), vendor and supplier management etc. However most of these solutions come with licensing costs amounting to thousands of dollar. To offer a completely free solution which would be made available through Apache 2.0 license, A Project is started in 2017 under the name ‘Yugandhar Open MDM Project’ to build Open Source MDM solutions catering to CDI, PIM and Data Governance Capabilities. Yugandhar in Sanskrit means Ever Lasting and the strongest of its time. Our vision is to build the strongest, Open Source, Multi Domain, Cross Industry and completely free MDM Solution.

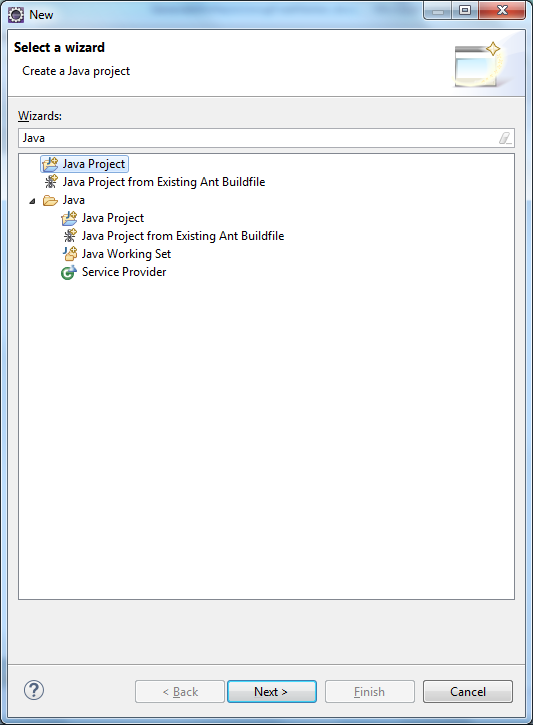
We are happy to announce that the first release of the Yugandhar MDM Hub catering to CDI solution is built with Open source technologies like Spring and Hibernate etc, inbuilt data Model, 400+ ready to use services and having incredible Out of the Box capabilities is currently being distributed. We aim to make the current CDI offering the strongest and Planning to bring Data Stewardship and PIM solutions in upcoming years.

# About this document

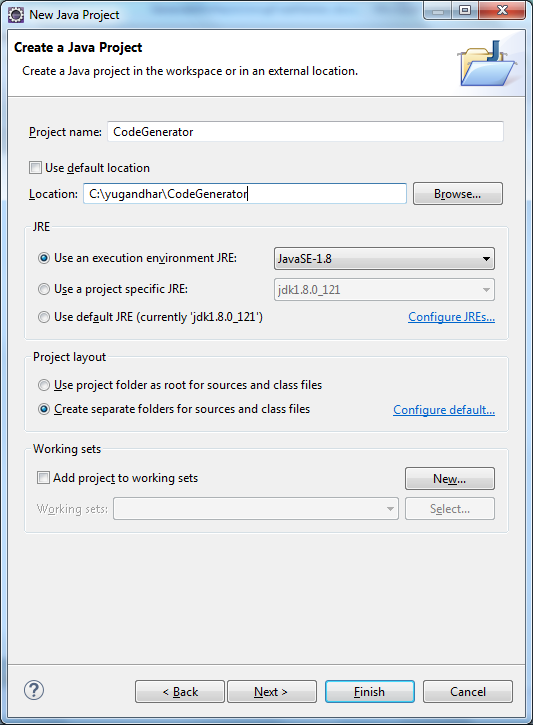
This document provides a step by step guide for generating the code using Hibernate reverse engineering tool and Yugandhar Freemarker templates. The generator will generate the artifacts which have services for add, update and retrieve services.

# Create New Java Project for Code Generation

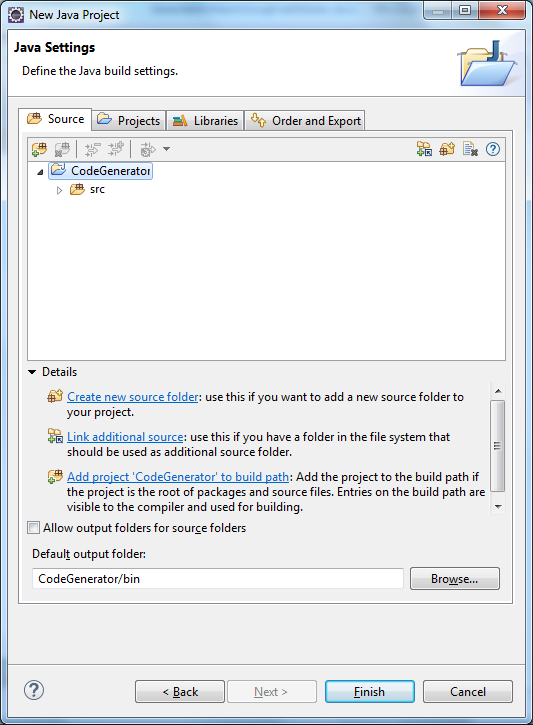
File 🡪 new 🡪 Java Project



Click Next

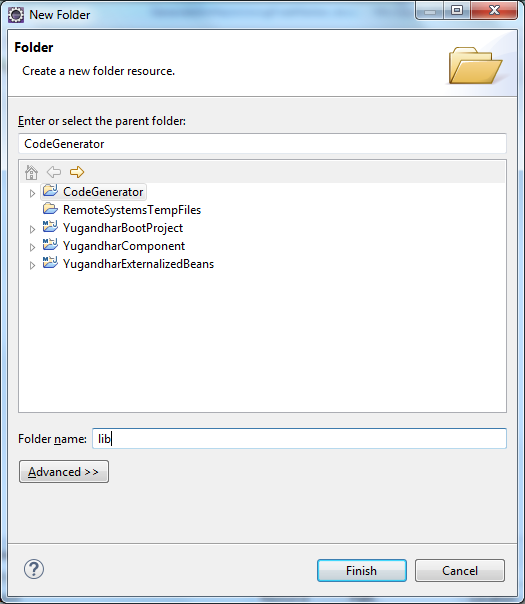


Click Next

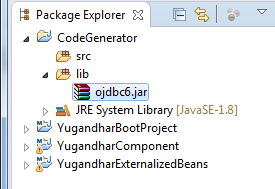


Click Finish

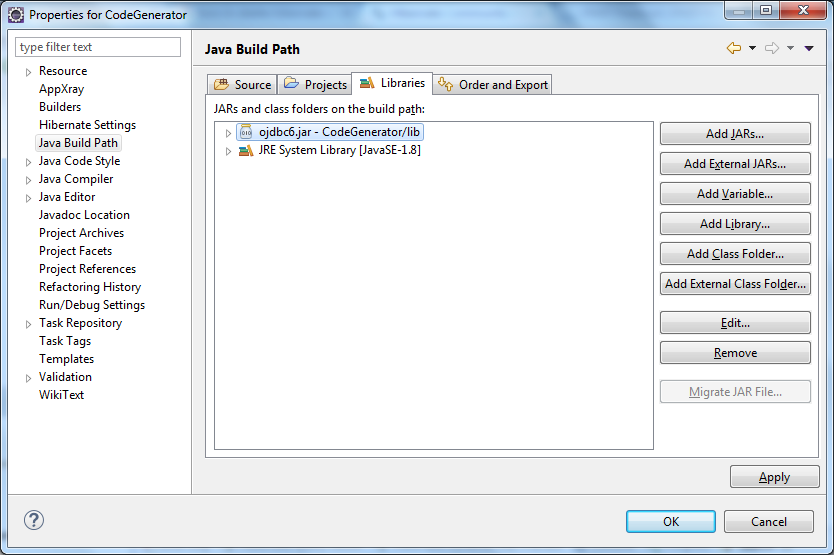
Create New folder named ‘lib’ in the project



Add ojdbc6.jar in the lib folder (download the jar from oracle website if you don’t have already)



Make sure that this ojdbc6.jar file is present in the libraries. If not then add it using ‘add JARs…’ button

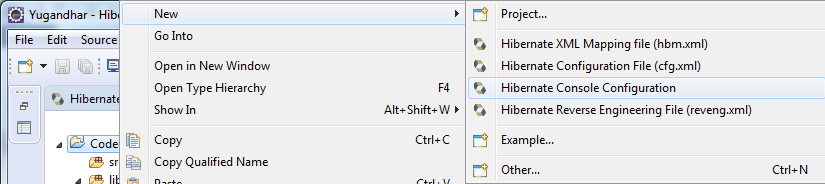


# Hibernate Console Configuration

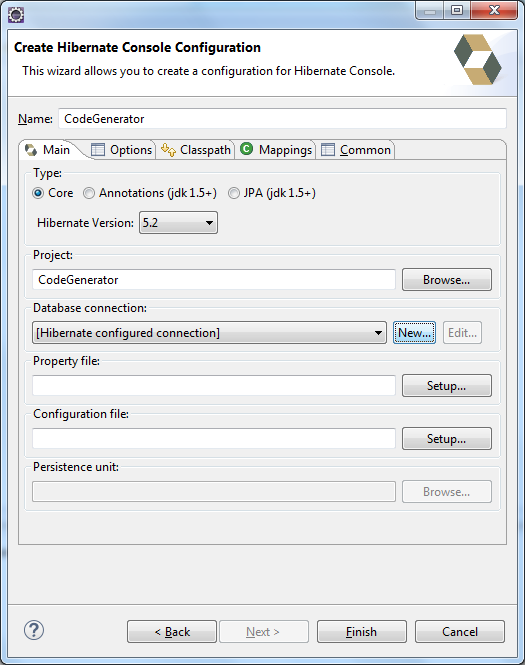
Switch to Hibernate view



Right click on CodeGenerator project and click ‘Hibernate Console Configuration’

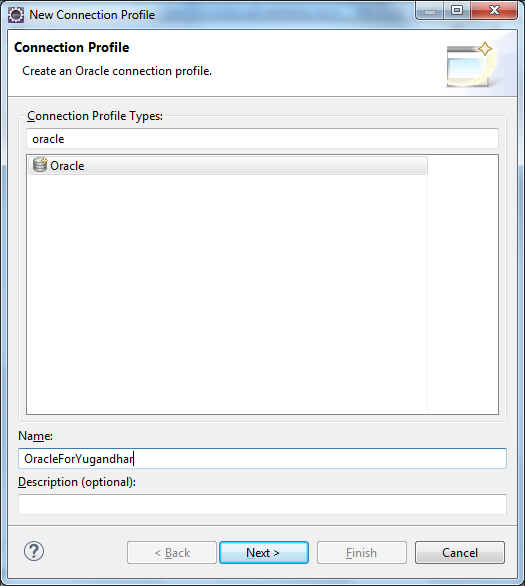


This will open a Hibernate Console Configuration window



Click on new button

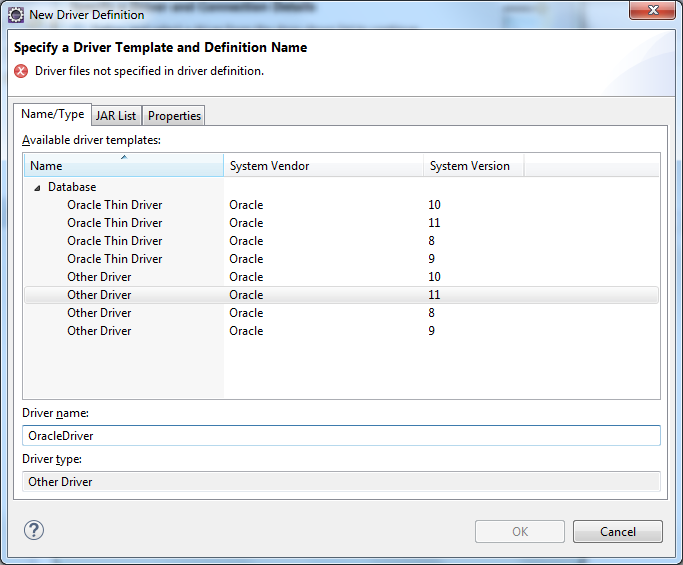


Select Oracle and Type the name of the connection  


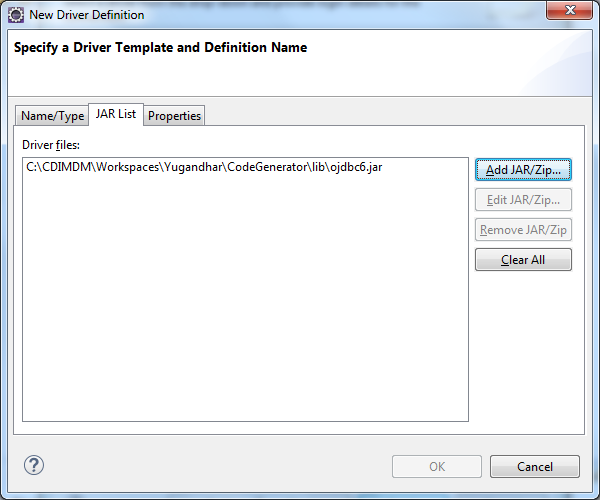
In the next page select the icon to add driver



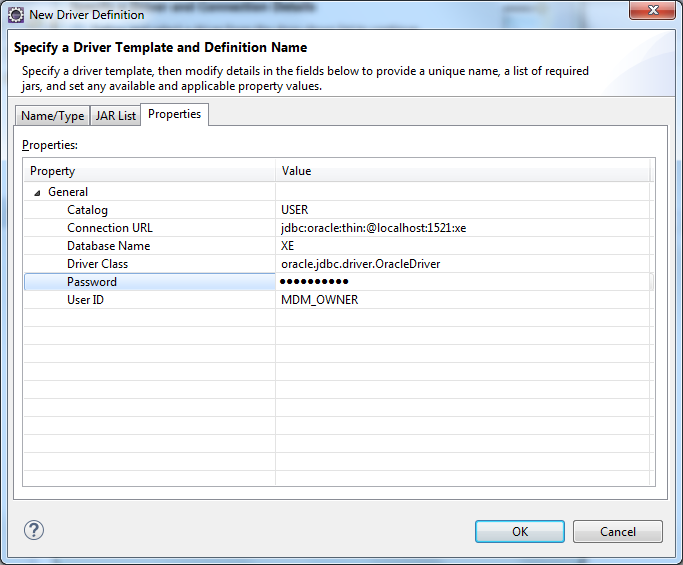
Select oracle 11 as driver



In the JAR list, add the ojdbc6.jar

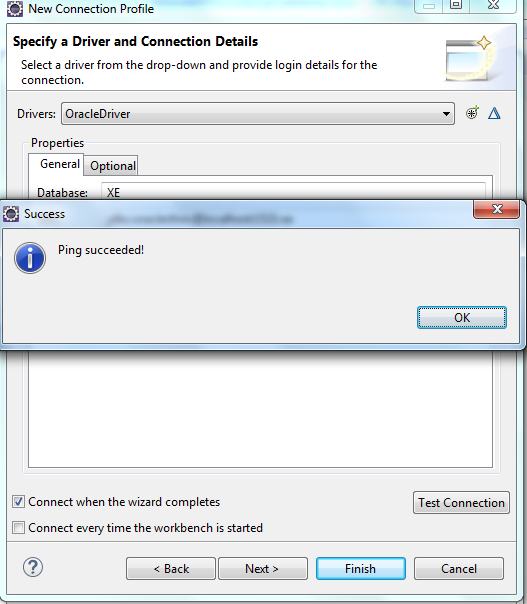


In the properties tab, enter values as per your database and schema

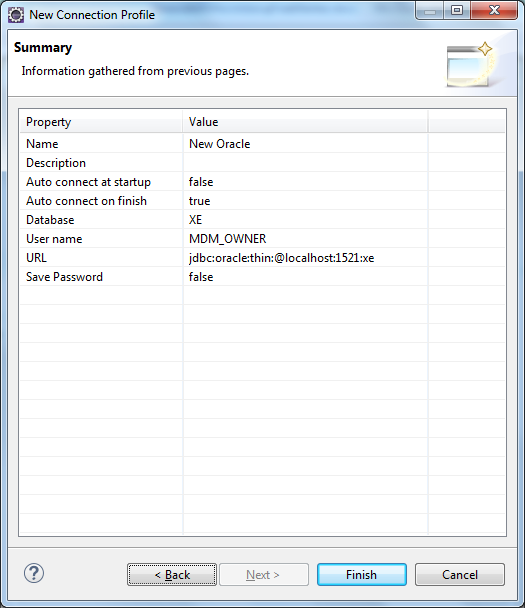


Click on the OK button

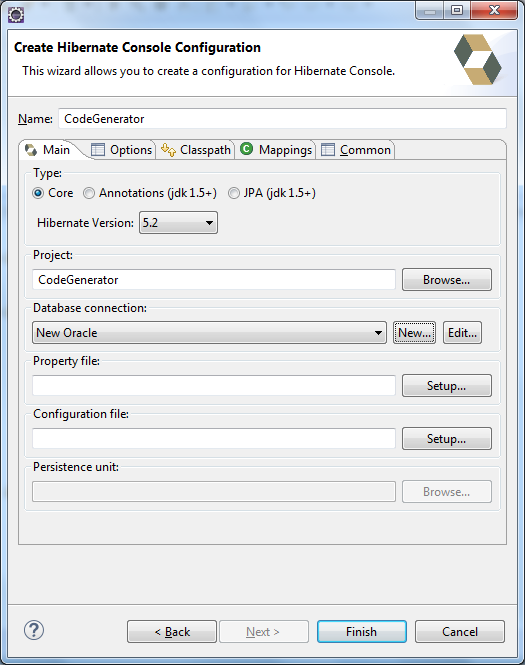
Click on the Test Connection button and check that the connection is successful



Click next and Finish

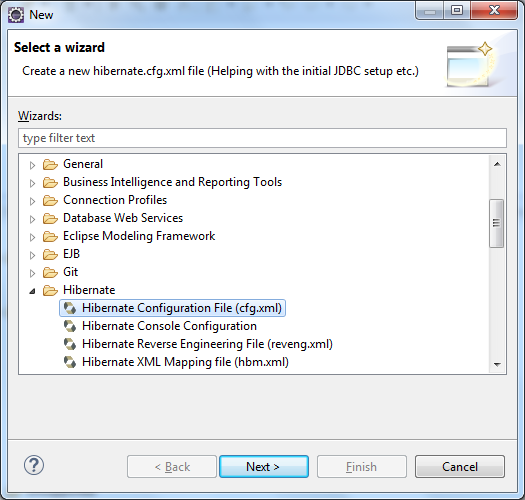


Now in the Hibernate Console Configuration you should see the OracleDriver in Database Connections. Click on Finish button.

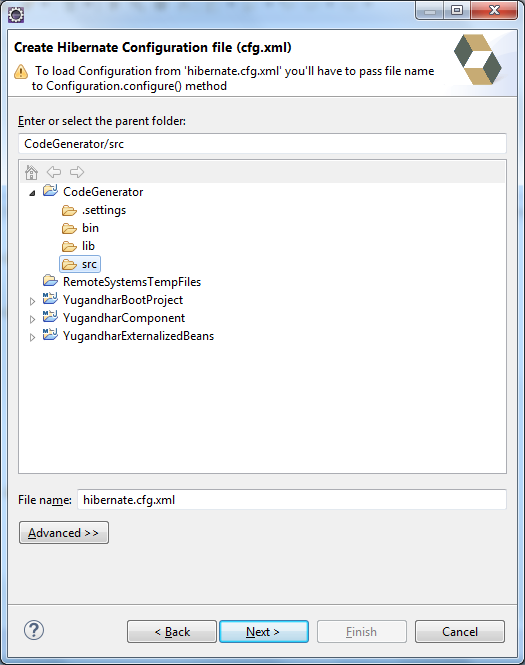


# Create hibernate Configuration file

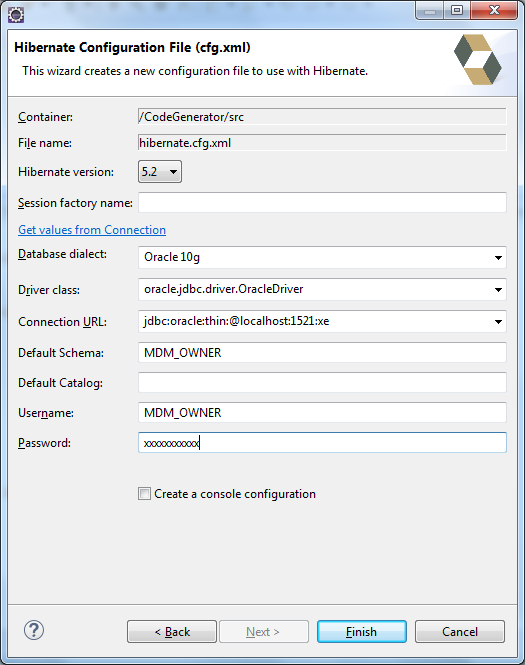
Right click code generator project and click New ‘Hibernate Configuration File (cfg.xml)



Select the src folder and click next

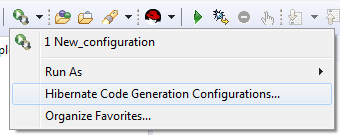


Provide the database details and Click Finish.

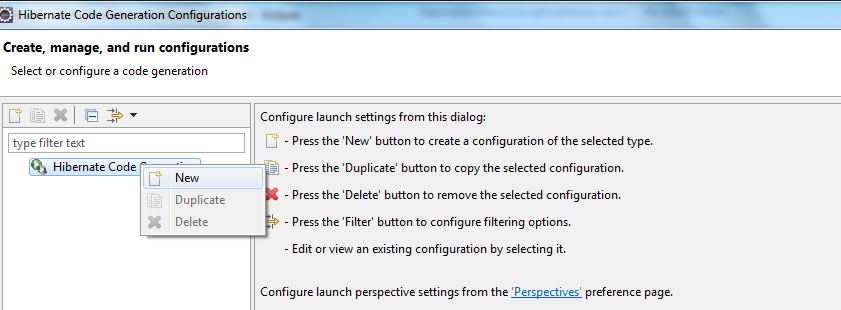


# Create Hibernate Code Generation Configurations

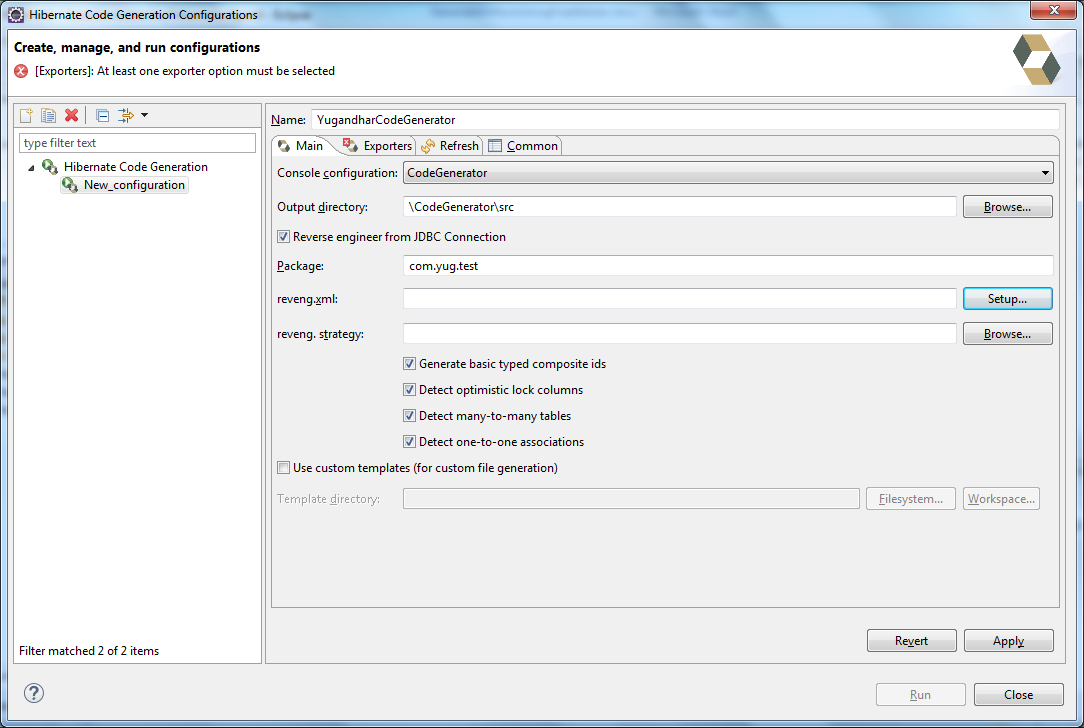
Switch to hibernate view and Click on the  button in the toolbar, it would give you below options



Right click and select New



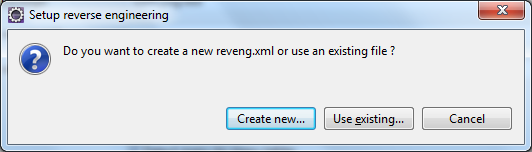
Select the New Configuration just created



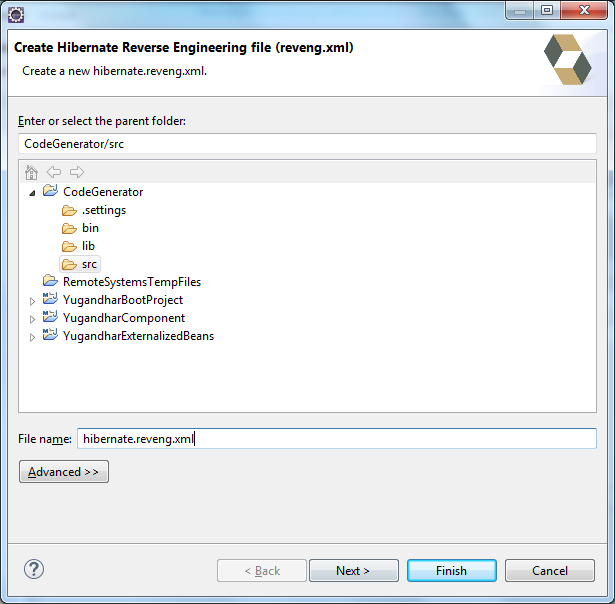
Select ‘Setup…’ button in the reveng.xml option



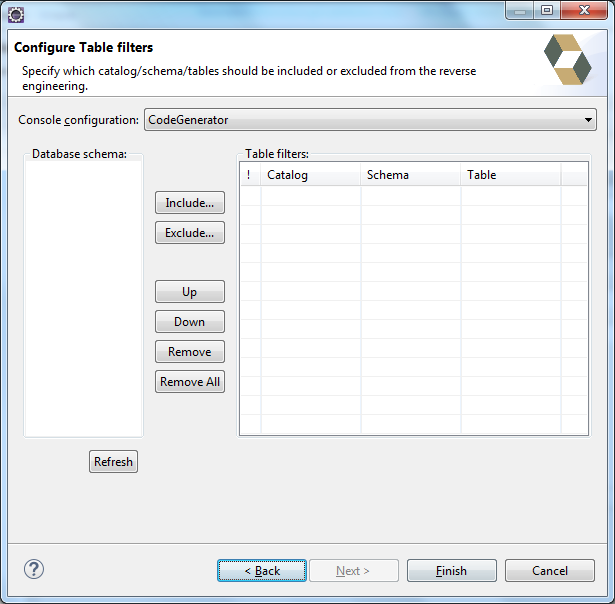
Click create new



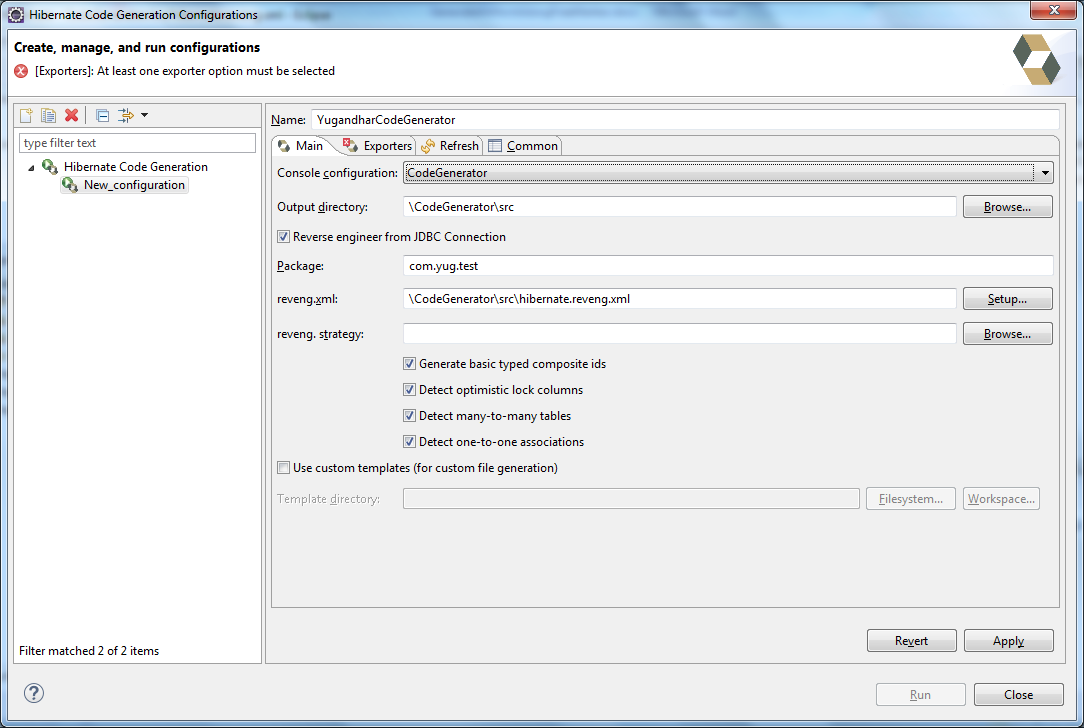
Select src folder, select next



Click Finish



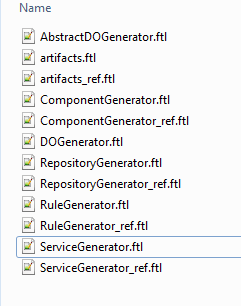
You should be back to Manage Configuration window, click on the exporters tab



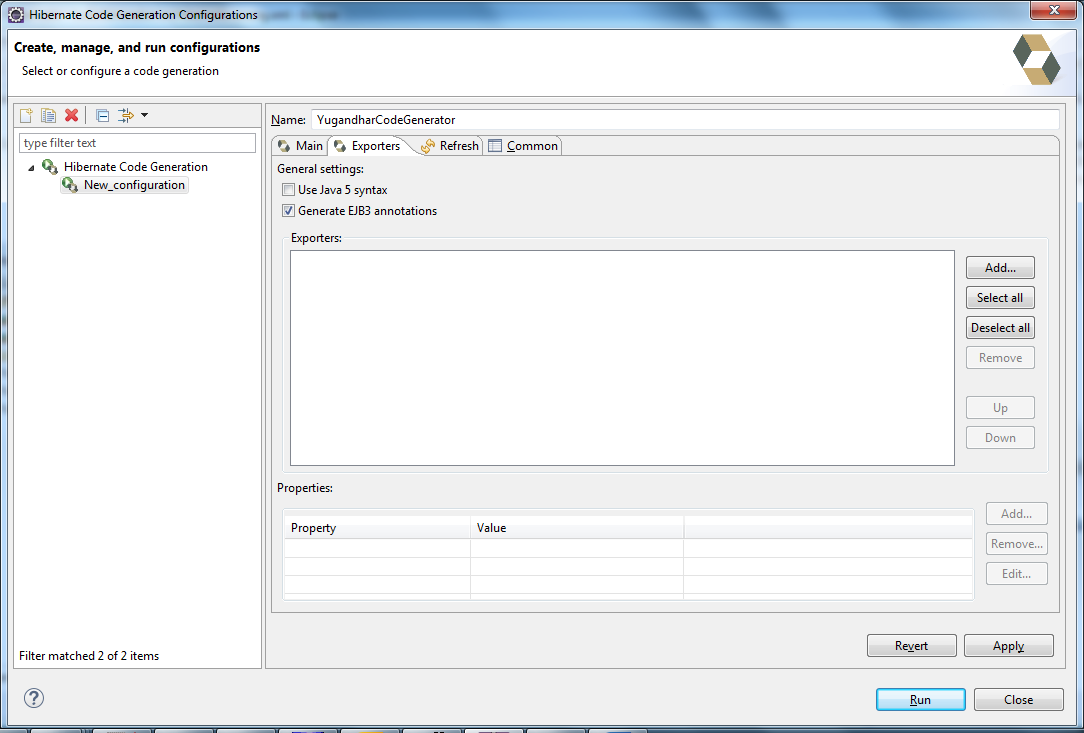
## Set the exporters (Freemarker Templates)

Now we are going to set the freemarker templates

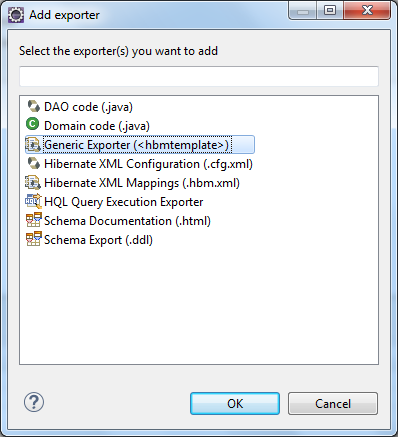
You should download the freemarker templates from the Yugandhar github repository from location. It would have below template files which will be used to generate code



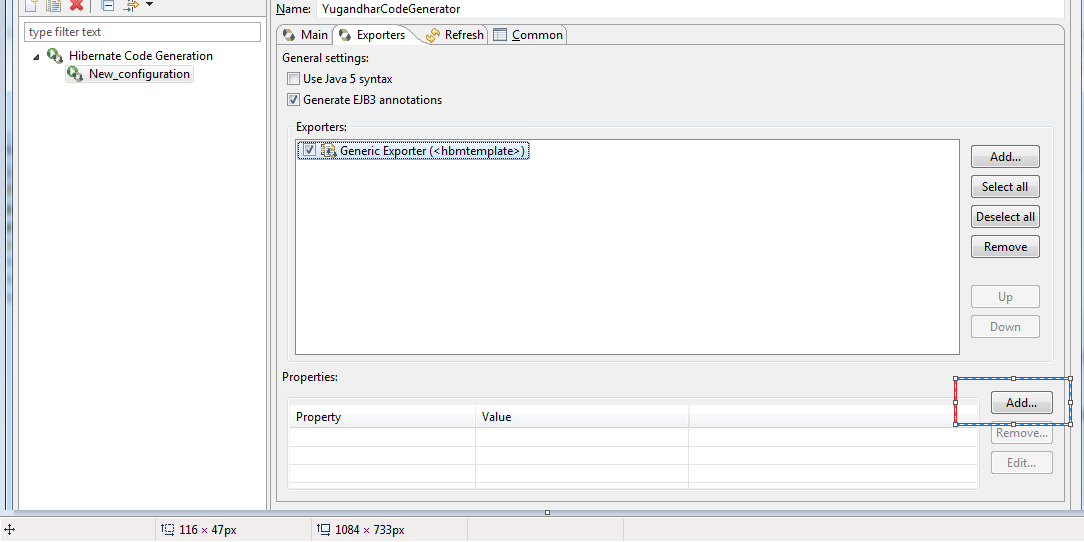
In the exporter tab, remove all the exporters if already present using the Remove button



Click on ‘Add…’ button and select generic exporter



Click on ‘Add…’ button next to properties tab at the botton. We need to add all the exporters mentioned above one by one. Let’s start with ‘AbstractDOGenerator.ftl’ template.



Add the properties as mentioned below for every exporter

### Common Generators

### Abstract DO Generator

File pattern [file\_pattern]- {package-name}/Abstract{class-name}DO.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - AbstractDOGenerator.ftl

### DO Generator

File pattern [file\_pattern]- {package-name}/{class-name}DO.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - DOGenerator.ftl

### Data Tables Object Generators

### ServiceGenerator

File pattern [file\_pattern]- {package-name}/{class-name}Service.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - ServiceGenerator.ftl

### ComponentGenerator

File pattern [file\_pattern]- {package-name}/{class-name}Component.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - ComponentGenerator.ftl

### RepositoryGenerator

File pattern [file\_pattern]- {package-name}/{class-name}Repository.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - RepositoryGenerator.ftl

### RuleGenerator

File pattern [file\_pattern]- {package-name}/{class-name}ComponentRule.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - RuleGenerator.ftl

### artifacts

File pattern [file\_pattern]- {package-name}/{class-name}\_artifacts.txt

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - artifacts.ftl

### Reference Data tables Object generators

### ServiceGenerator\_ref

File pattern [file\_pattern]- {package-name}/{class-name}Service.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - ServiceGenerator\_ref.ftl

### ComponentGenerator\_ref

File pattern [file\_pattern]- {package-name}/{class-name}Component.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - ComponentGenerator\_ref.ftl

### RepositoryGenerator\_ref

File pattern [file\_pattern]- {package-name}/{class-name}Repository.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - RepositoryGenerator\_ref.ftl

### RuleGenerator\_ref

File pattern [file\_pattern]- {package-name}/{class-name}ComponentRule.java

Output directory [outputdir]- \CodeGenerator\src

Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - RuleGenerator\_ref.ftl

### artifacts\_ref

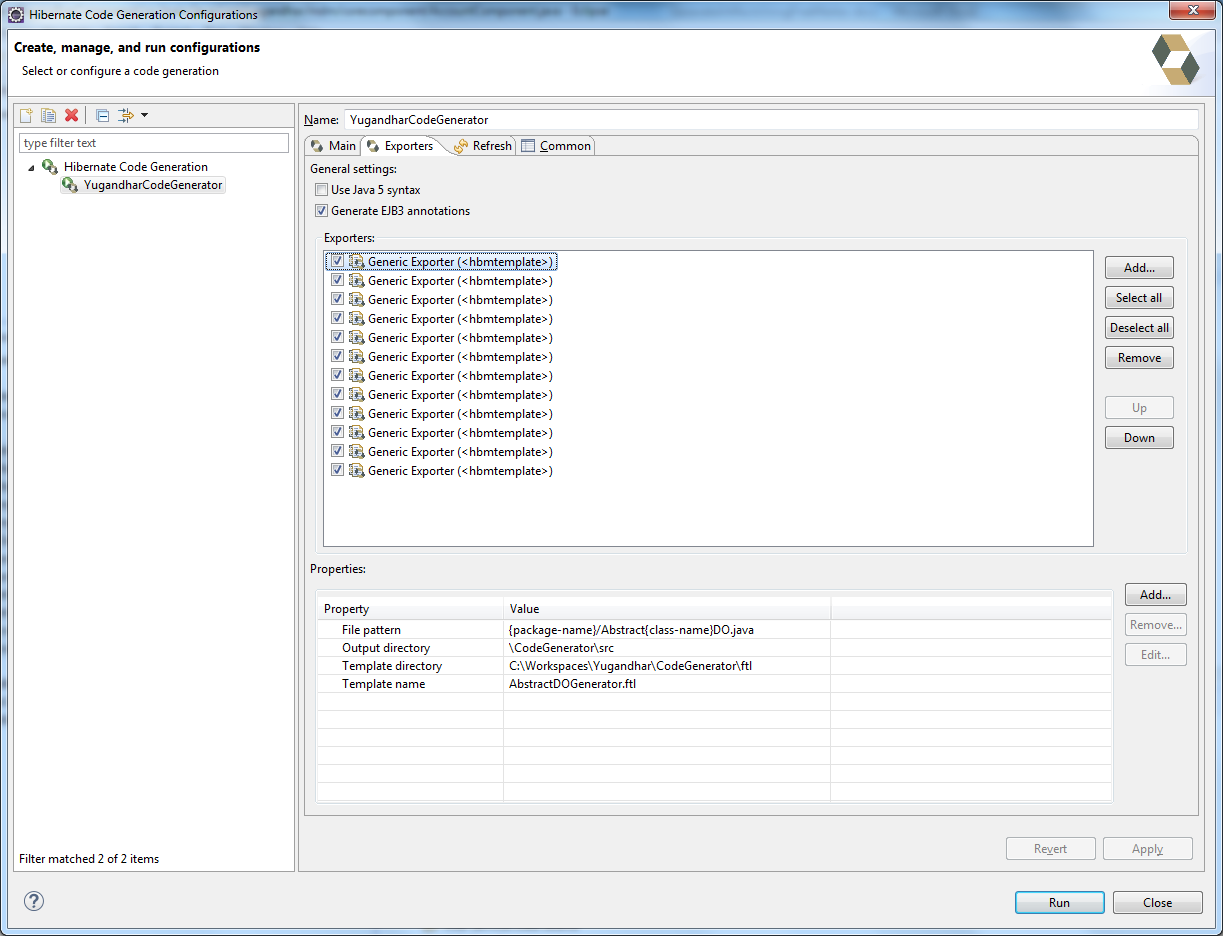
File pattern [file\_pattern]- {package-name}/{class-name}\_artifacts\_ref.txt

Output directory [outputdir]- \CodeGenerator\src

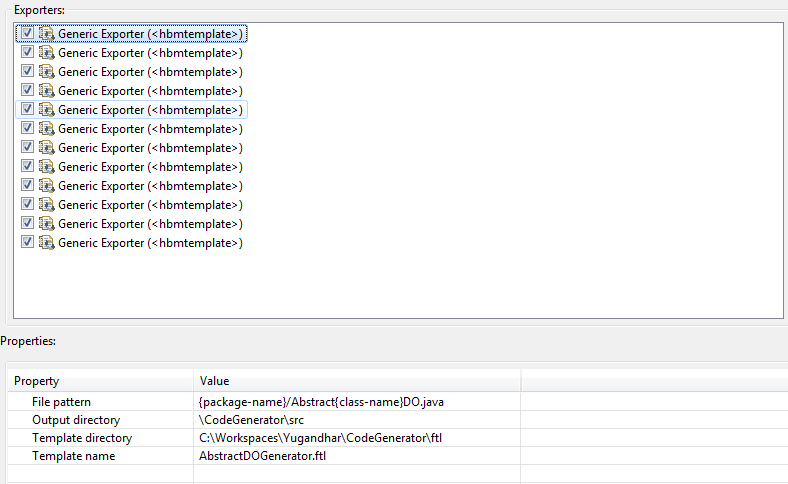
Template directory [template\_path] - C:\Workspaces\Yugandhar\CodeGenerator\ftl

Template name [template\_name] - artifacts\_ref.ftl

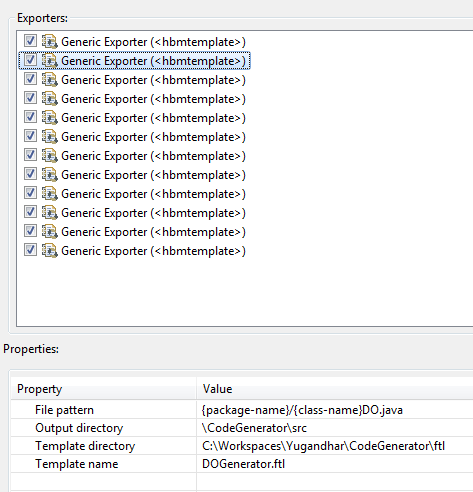
You must have total 12 exporters. You may choose to skip some exporters if not needed (e.g. if you want only reference LOV generation related exporter then you can skip the data table object generator exporters but it’s nice to setup all the exporters at one go). This is one time setup and can be reused as long as you use the same workspace.



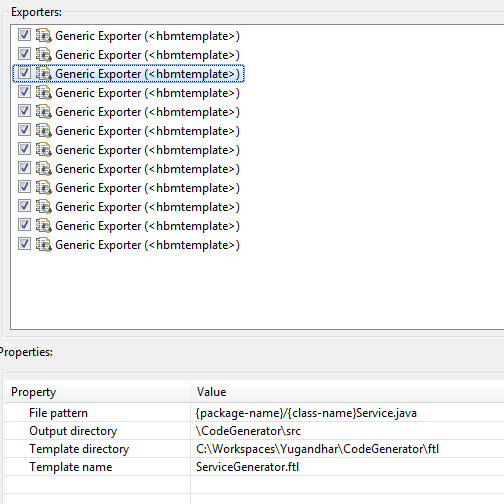
Exporter 1



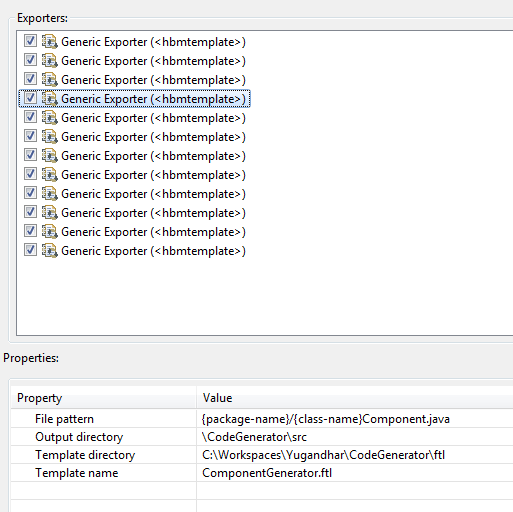
Exporter 2



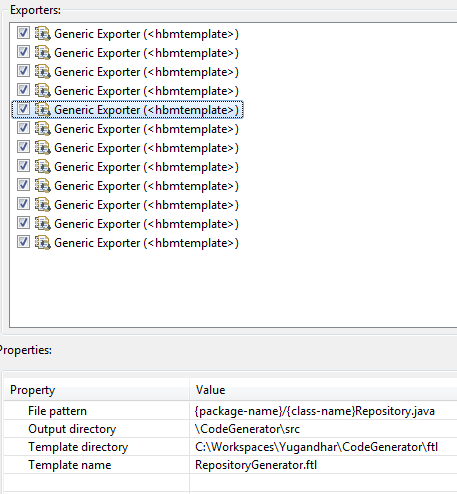
Exporter 3



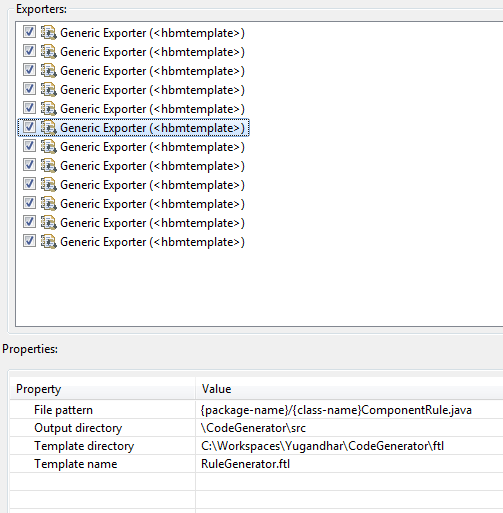
Exporter 4



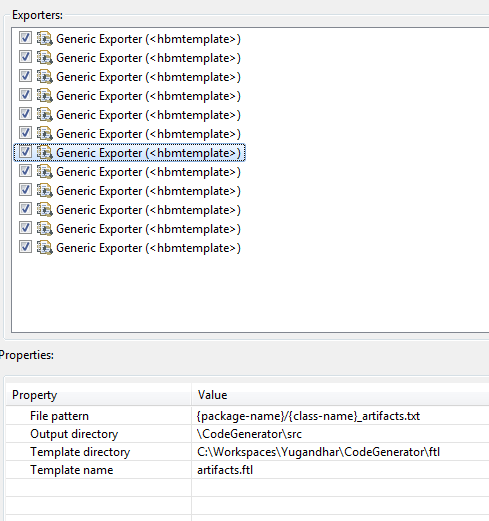
Exporter 5



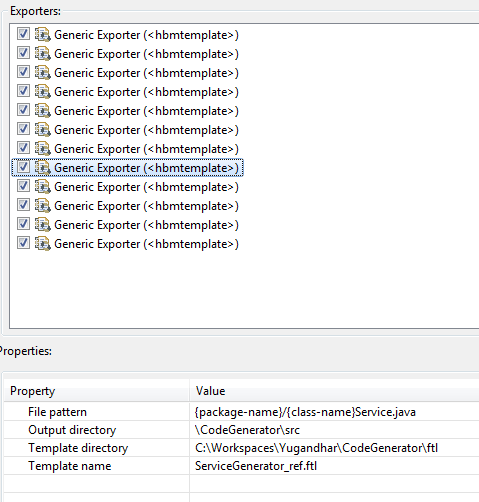
Exporter 6



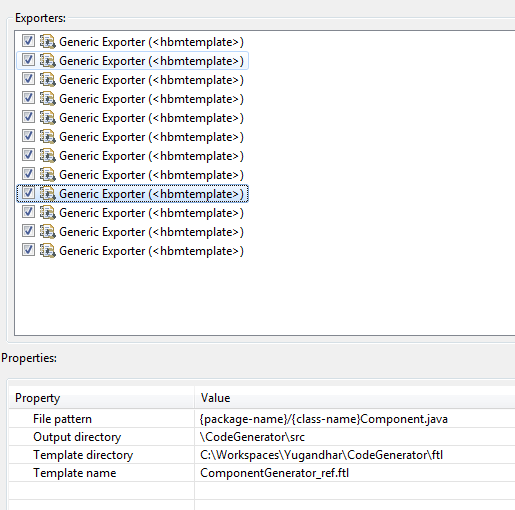
Exporter 7



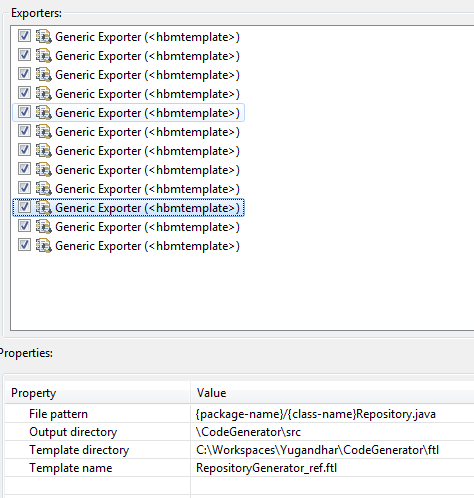
Exporter 8



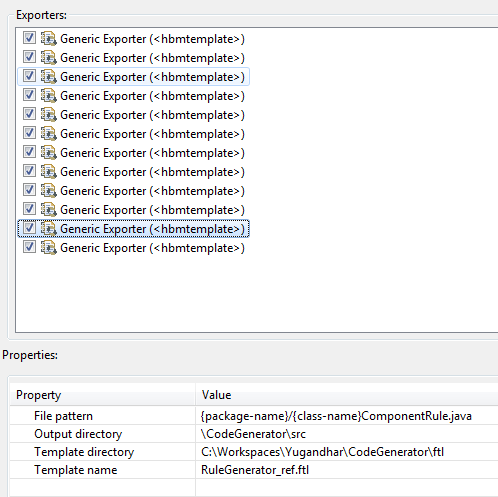
Exporter 9



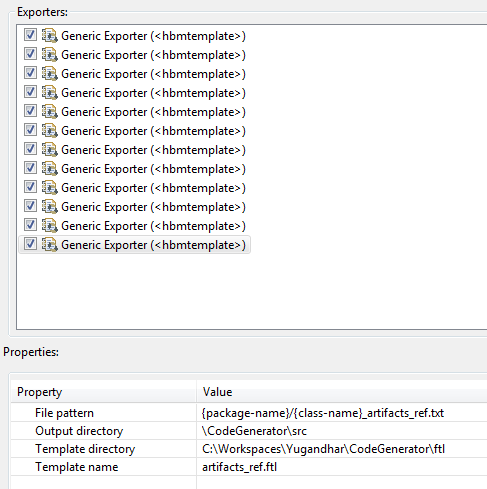
Exporter 10



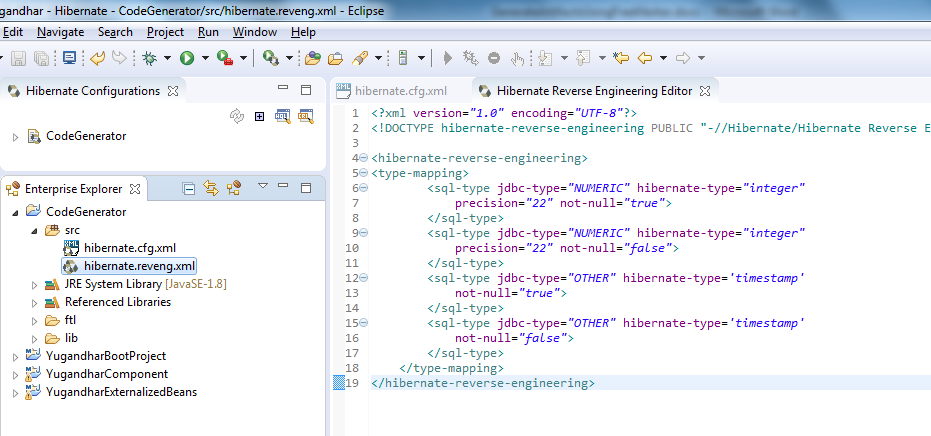
Exporter 11



Exporter 12



# Editing the Reverse Engineering xml and Generating Code



Open hibernate.reveng.xml, go to source tab And paste the below type maps inside <hibernate-reverse-engineering> tags

<type-mapping>

<sql-type jdbc-type=*"NUMERIC"* hibernate-type=*"integer"*

precision=*"22"* not-null=*"true"*>

</sql-type>

<sql-type jdbc-type=*"NUMERIC"* hibernate-type=*"integer"*

precision=*"22"* not-null=*"false"*>

</sql-type>

<sql-type jdbc-type=*"OTHER"* hibernate-type=*'timestamp'*

not-null=*"true"*>

</sql-type>

<sql-type jdbc-type=*"OTHER"* hibernate-type=*'timestamp'*

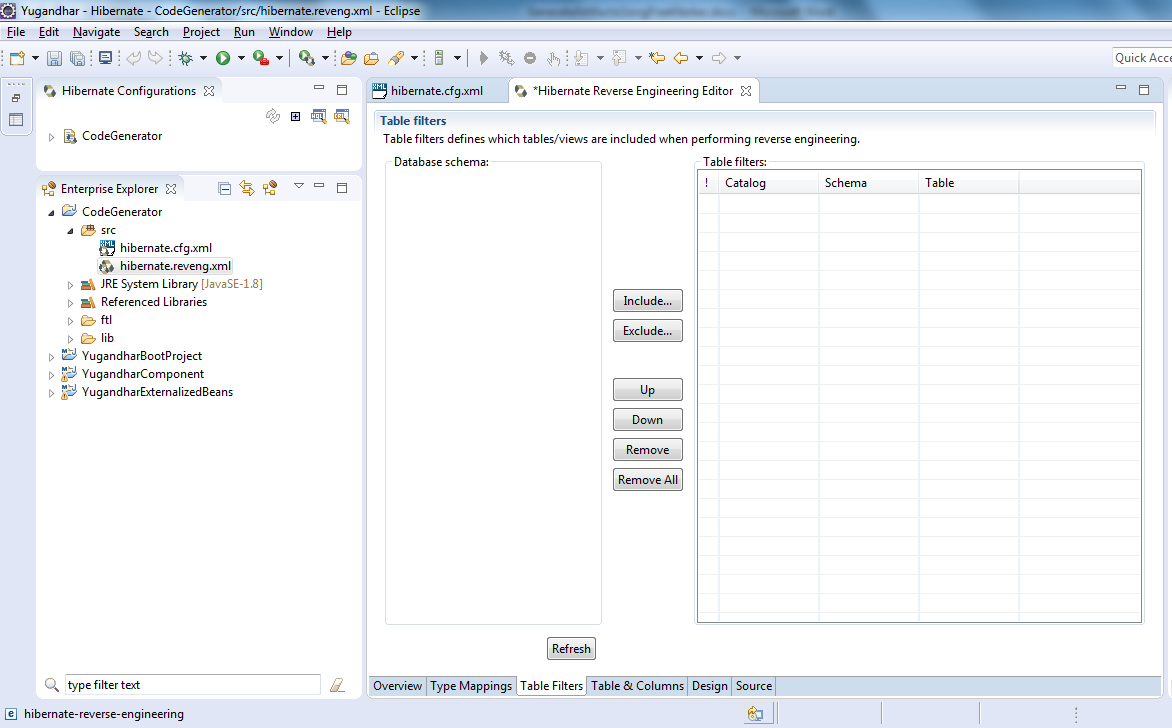
not-null=*"false"*>

</sql-type>

</type-mapping>

Now click on the Table filters tab and click on Refresh button. It should list all your tables in the left pane.

Then select the required tables for you want to generate the code



We have created FACILITYDEMO table (type of data table) for demo purpose

CREATE TABLE MDM\_OWNER.FACILITYDEMO

(

ID\_PK VARCHAR2(50 BYTE) PRIMARY KEY,

VERSION NUMBER NOT NULL,

CREATED\_TS TIMESTAMP(6) NOT NULL,

DELETED\_TS TIMESTAMP(6),

UPDATED\_TS TIMESTAMP(6) NOT NULL,

UPDATED\_BY\_USER VARCHAR2(50 BYTE) NOT NULL,

UPDATED\_BY\_TXN\_ID VARCHAR2(100 BYTE),

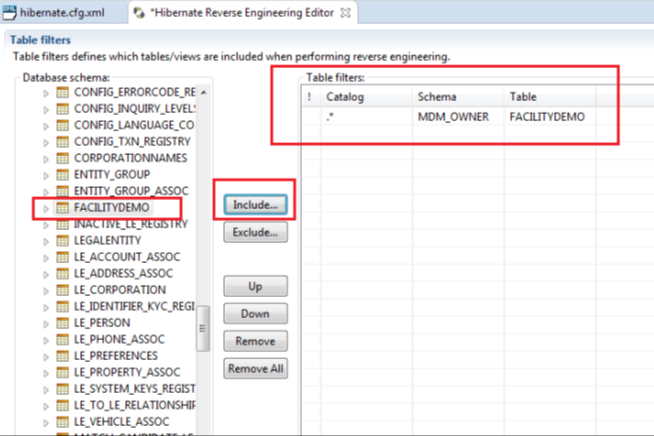
FACILITY\_NAME VARCHAR2(100 BYTE) NOT NULL,

LOCATION VARCHAR2(100 BYTE)

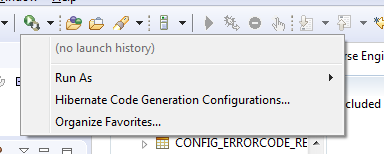
);

You may also create this table in the database before clicking on the refresh button using above sql script.

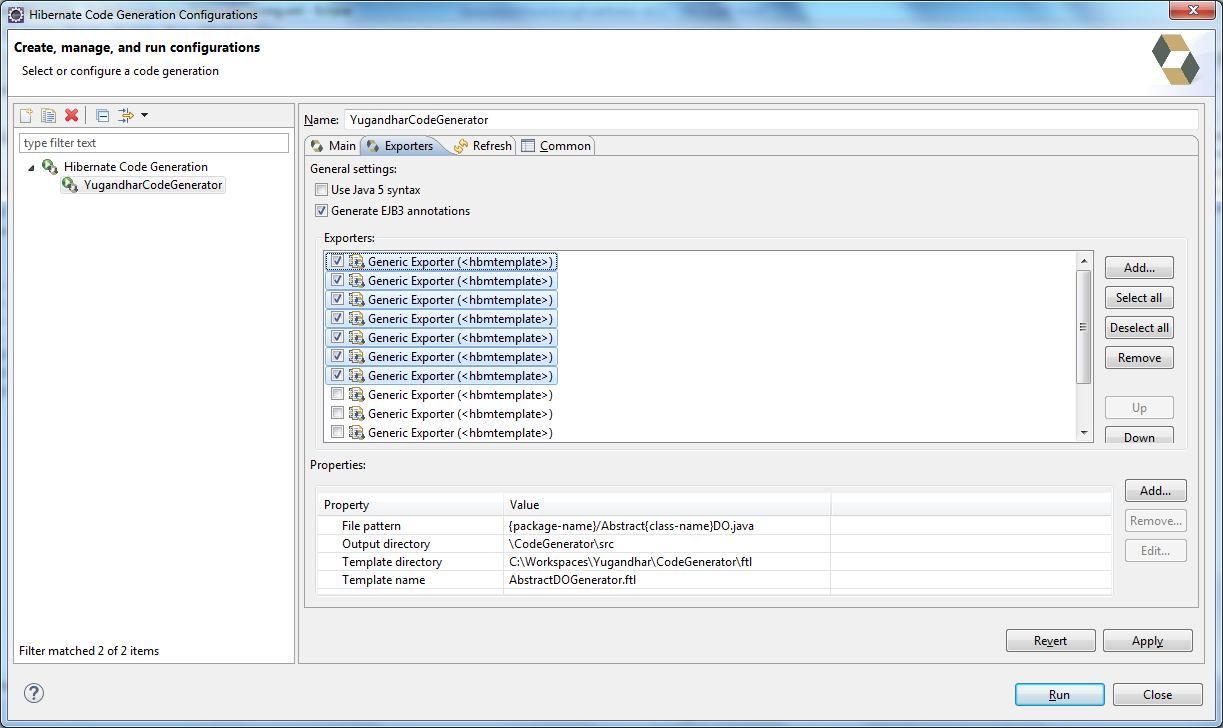
Once loaded, select FACILITYDEMO table and click on include. Save the file.



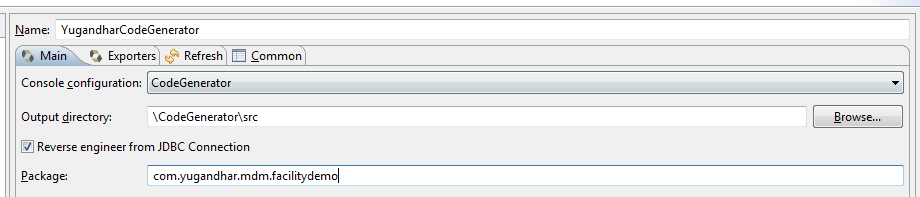
Now to generate the code, go to hibernate perspective and click on the below shown option which will open the exporters we had configured.



Now as we are generating the data table, we need not be choosing the reference table code generators. So deselect the exporters related to reference tables

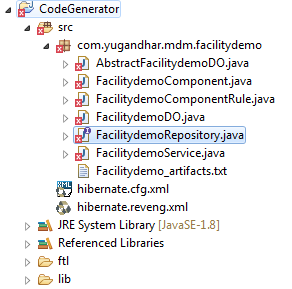


In the main tab, you have the option to change the package name of the generated artifacts



Once you verified everything, click on Apply and Run button. IT would generate the artifacts in given package

The generated code would look like below



Now move this code to your Maven project and link this to Yugandhar Open MDM project.

# Plugging generated artifacts

For the data entity the generated file <entityname>\_artifacts.txt (e.g. Facilitydemo\_artifacts.txt) will have the generated artifacts.

## The Artifacts for data entity

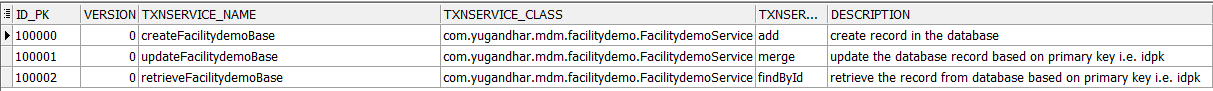
The artifacts for the data entity are mentioned below

* SQL to register newly created transactions in application transaction registry.
* Sample messages

You would see below sqls being generated in the artifacts.txt file. The sqls are for registering the auto generated create, update and retrieve transactions. All of the transactions are base so suffix ‘Base’ is added at the end of the transaction name.

|  |
| --- |
| Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION, CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'createFacilitydemoBase', 'com.yugandhar.mdm.facilitydemo.FacilitydemoService', 'add',  'create record in the database', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP, 'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'updateFacilitydemoBase', 'com.yugandhar.mdm.facilitydemo.FacilitydemoService', 'merge',  'update the database record based on primary key i.e. idpk', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP,'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'retrieveFacilitydemoBase', 'com.yugandhar.mdm.facilitydemo.FacilitydemoService', 'findById',  'retrieve the record from database based on primary key i.e. idpk', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP,'Generator', '000000000');  COMMIT; |

Replace the <SCHEMA\_NAME> in the script with the actual schema name e.g. MDM\_OWNER and execute on the database. Execute the above script on MDM\_OWNER schema and verify that the rows are inserted in CONFIG\_TXN\_REGISTRY table.



Note – The scripts for Insert, update, delete triggers is not created using artifacts and the same needs to be created manually.

## The Artifacts for Reference data entity

* SQL to register newly created transactions in application transaction registry.
* Ehcache configuration entries
* Sample messages

The sample generated artifact for reference data is shows below. (this is for information purpose only and not related to facilityDemo sample)

The reference data entity has couple of more transactions generated by code generated as below. The sample SQL are related to RefInactivationReason reference LOV.

|  |
| --- |
| /\* insert sqls for transaction registration \*/  //----------------------------------------------------  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION, CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'createRefInactivationReasonBase', 'com.yugandhar.mdm.match.componentref.RefInactivationReasonService', 'add',  'create record in the database', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP, 'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'updateRefInactivationReasonBase', 'com.yugandhar.mdm.match.componentref.RefInactivationReasonService', 'merge',  'update the database record based on primary key i.e. idpk', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP,'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'retrieveRefInactivationReasonBase', 'com.yugandhar.mdm.match.componentref.RefInactivationReasonService', 'findById',  'retrieve the record from database based on primary key i.e. idpk', CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP,'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'findRefInactivationReasonByBusinessKeyBase', 'com.yugandhar.mdm.match.componentref.RefInactivationReasonService', 'findByBusinessKey',  'find the unique record from dababase based on by business key',CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP, 'Generator', '000000000');  Insert into <SCHEMA\_NAME>.CONFIG\_TXN\_REGISTRY  (ID\_PK, VERSION, TXNSERVICE\_NAME, TXNSERVICE\_CLASS, TXNSERVICE\_CLASSMETHOD, DESCRIPTION,CREATED\_TS, UPDATED\_TS, UPDATED\_BY\_USER, UPDATED\_TXN\_REF\_ID)  Values  (YUG\_REGISTRY\_SEQ.nextval, 0, 'findAllRefInactivationReasonByLanguageCodeBase', 'com.yugandhar.mdm.match.componentref.RefInactivationReasonService', 'findAllRecordsByLanguageCode',  'find All records by language code',CURRENT\_TIMESTAMP,CURRENT\_TIMESTAMP, 'Generator', '000000000');    COMMIT; |

The reference data LOV gets cached so we will be making the ehcache configuration as below.

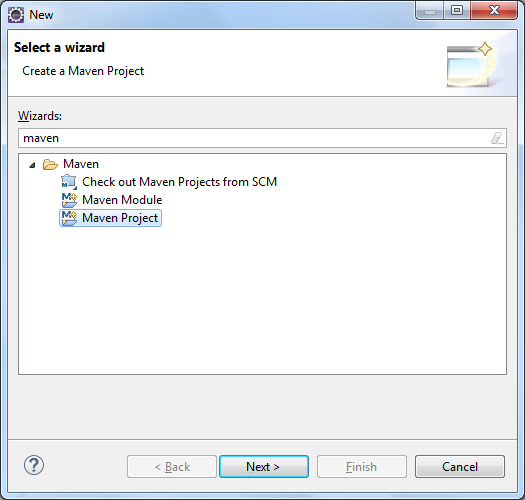
|  |
| --- |
| // Add below block of code in xpath <service><jsr107:defaults> of the /YugandharBootProject/src/main/resources/ehcache.xml  //----------------------------------------------------  <jsr107:cache name="REFINACTIVATIONREASON\_BUSKEY" template="heap-cache" />    // Add below block of code in xpath <Config> of the /YugandharBootProject/src/main/resources/ehcache.xml  //----------------------------------------------------  <cache alias="REFINACTIVATIONREASON\_BUSKEY" uses-template="heap-cache" >  <expiry>  <ttl unit="seconds">30</ttl>  </expiry>  </cache> |

Note – The scripts for Insert, update, delete triggers is not created using artifacts and the same needs to be created manually.

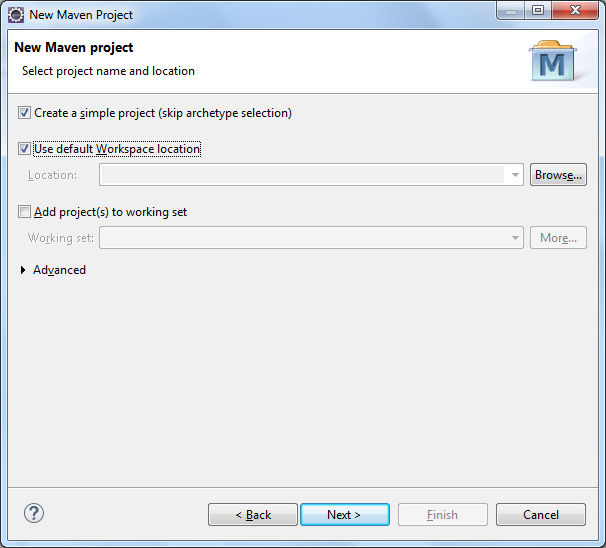
## Merging code with Yugandhar Open MDM Hub

### Create new Maven Project

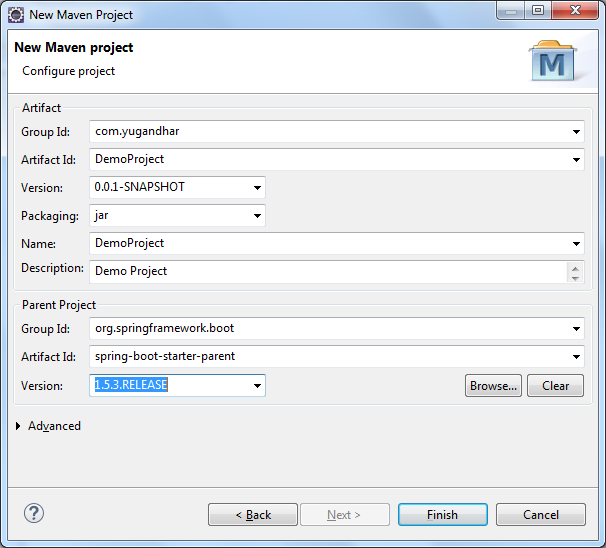
To merge the code we need to create a new Maven project. In eclipse, select File 🡪 new 🡪 Maven Project



Click Next

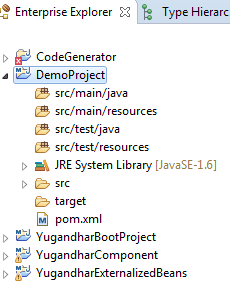


Select ‘Create a Simple Project(skip archetype selection)’ and Click Next



Click Finish.

You must see a new project created as below

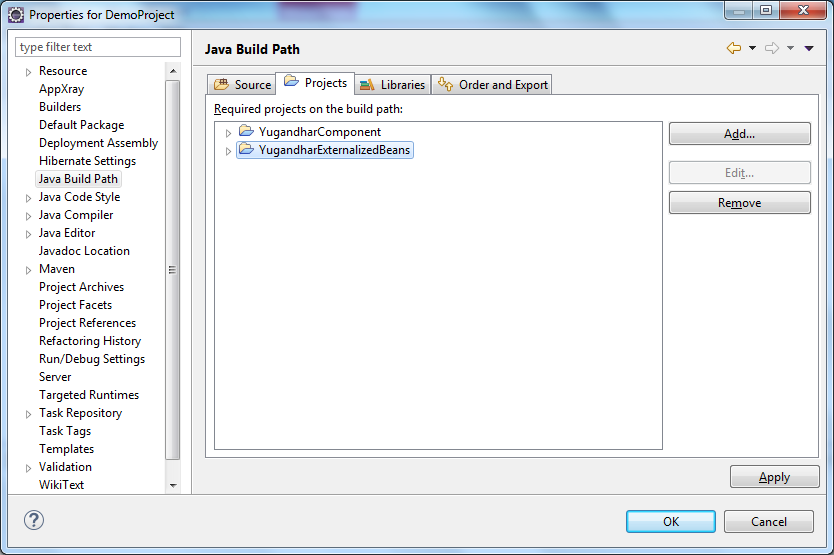


Edit pom.xml and add below entries in it

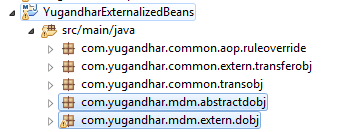
|  |
| --- |
| <properties>  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>  <project.reporting.outputEncoding>UTF-8</project.reporting.outputEncoding>  <java.version>1.8</java.version>  </properties>  <dependencies>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-web</artifactId>  <exclusions>  <exclusion>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-tomcat</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-test</artifactId>  <scope>test</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-aop</artifactId>  </dependency>  <dependency>  <groupId>javax.servlet</groupId>  <artifactId>servlet-api</artifactId>  <version>2.5</version>  <scope>provided</scope>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-data-jpa</artifactId>  <exclusions>  <exclusion>  <groupId>org.apache.tomcat</groupId>  <artifactId>tomcat-jdbc</artifactId>  </exclusion>  </exclusions>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-jta-narayana</artifactId>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-cache</artifactId> <!--Starter for using Spring Framework's caching support -->  </dependency>  <dependency>  <groupId>javax.cache</groupId> <!-- JSR-107 API -->  <artifactId>cache-api</artifactId>  </dependency>  <dependency>  <groupId>org.ehcache</groupId>  <artifactId>ehcache</artifactId>  <!-- <version>3.0.0</version> -->  </dependency>  <!-- For text comparision and matching. https://commons.apache.org/proper/commons-text/ -->  <dependency>  <groupId>org.apache.commons</groupId>  <artifactId>commons-text</artifactId>  <version>1.1</version>  </dependency>  <dependency>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-starter-activemq</artifactId>  </dependency>  <dependency>  <groupId>org.apache.activemq</groupId>  <artifactId>activemq-broker</artifactId>  </dependency>  </dependencies>  <build>  <plugins>  <plugin>  <groupId>org.springframework.boot</groupId>  <artifactId>spring-boot-maven-plugin</artifactId>  </plugin>  </plugins>  </build> |

Edit the build path of the Demo project and add the below two projects in the build path.

Right click Demo Project 🡪 properties 🡪 build Path 🡪 project 🡪 ‘Add…’ button 🡪 add the projects 🡪 click apply and Ok.



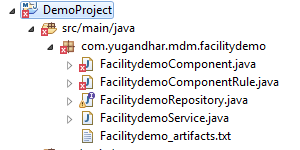
### Register the Data Objects in Yugandhar Externalized beans



Copy FacilitydemoDO.java to *com.yugandhar.mdm.extern.dobj* package

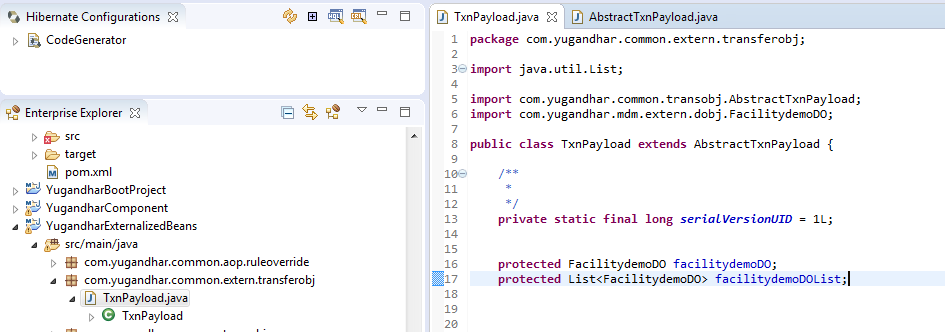
Copy AbstractFacilitydemoDO.java to *com.yugandhar.mdm.abstractdobj* package

Copy the other files in Demo project as shown in below screenshot.



You will see some errors as FacilitydemoDO is not registered in TxnPayload transfer object. So Open The TxnPayload class from below path

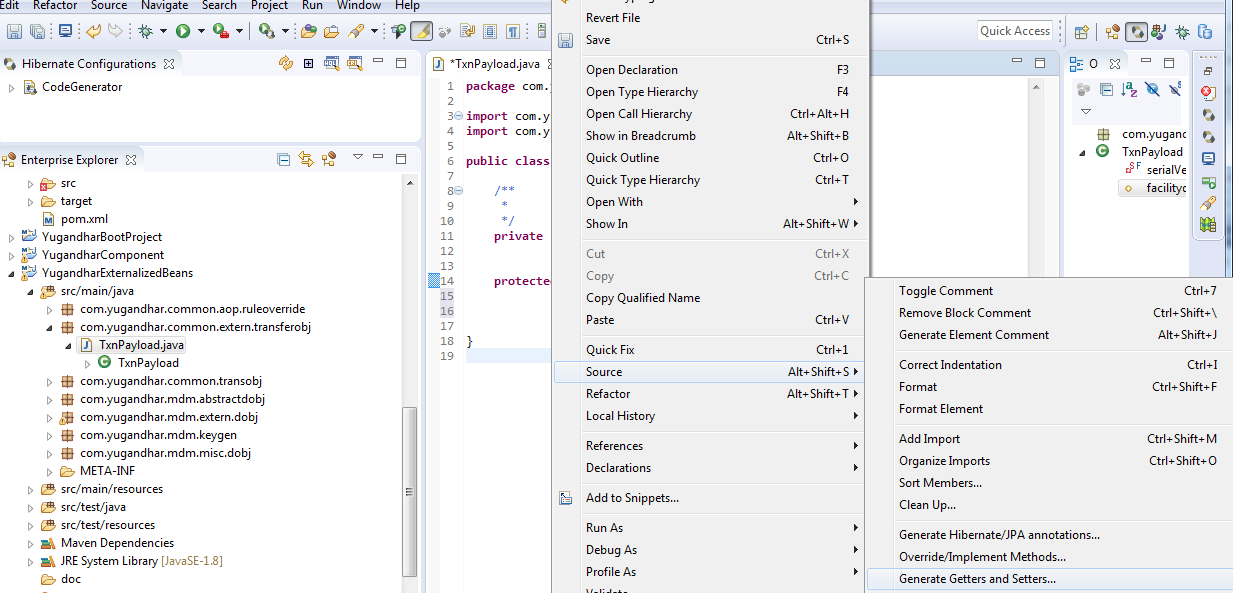
/YugandharExternalizedBeans/src/main/java/com/yugandhar/common/extern/transferobj/TxnPayload.java



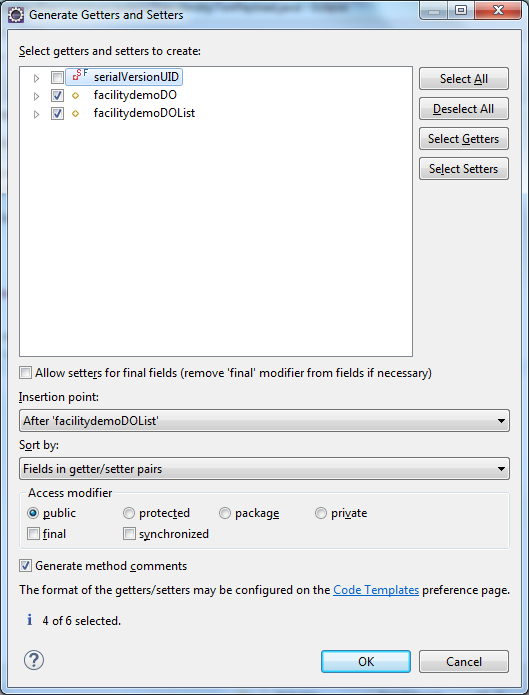
Register the FacilitydemoDO as single as well as list object. List object will be used when multiple objects needs to be present in the request or in response.

**protected** FacilitydemoDO facilitydemoDO;

**protected** List<FacilitydemoDO> facilitydemoDOList;

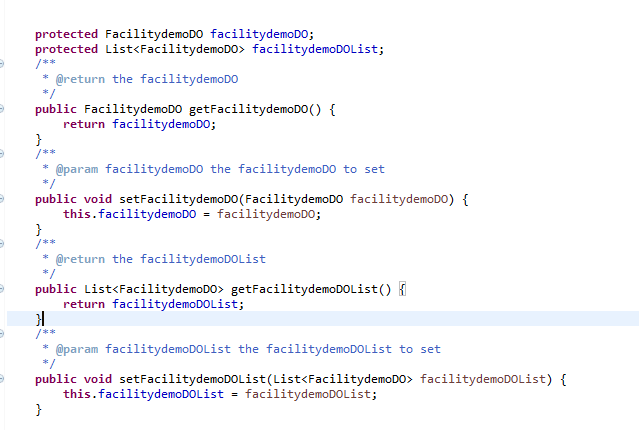


Click Generate Getters and setters…

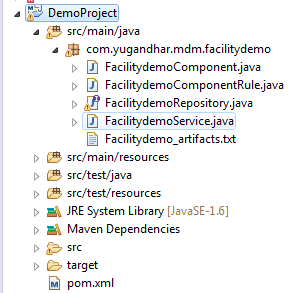


Select both the objects and Click ok

This will generate the getters and setters for the object

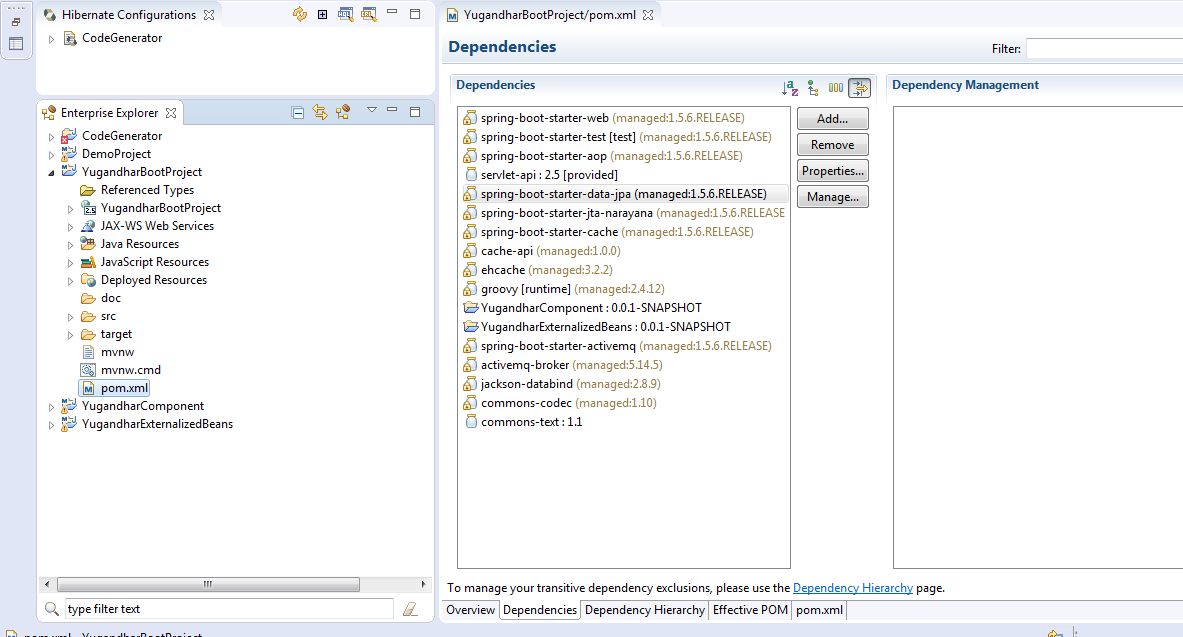


Build the workspace. You will see that the errors has vanished now



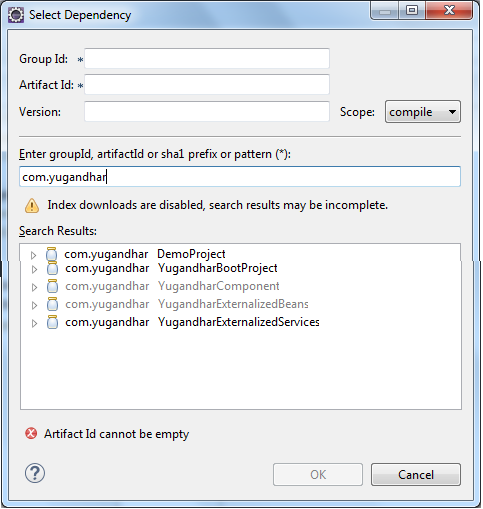
Now add DemoProject in the pom dependency of Yugandhar Boot project so that it will part of deployment

Open pom.xml in YugandharBootProject

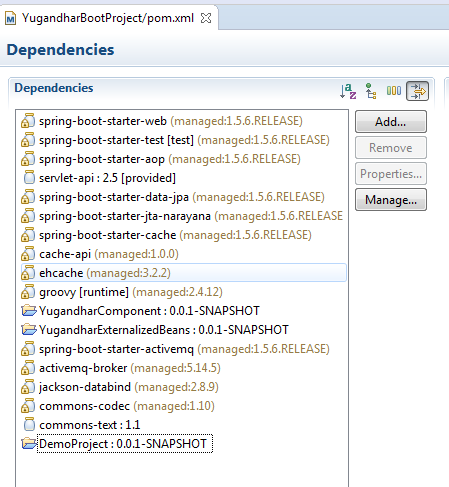


In the ‘Dependancies’ tab, click on ‘Add…’ button.

Type ‘com.yugandhar’ in the search box, which will give you the list of artifacts available



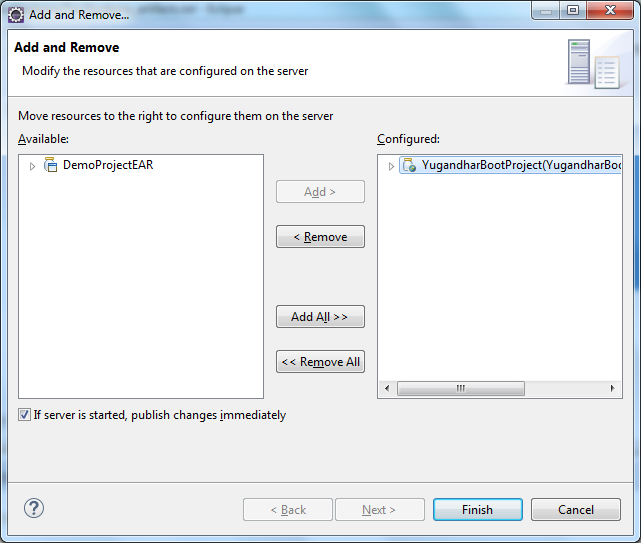
You would see the Demo project added in the spring boot project dependencies as below.



Save pom.xml

Deploying on Server

Right click on the jboss server and click add and Finish.



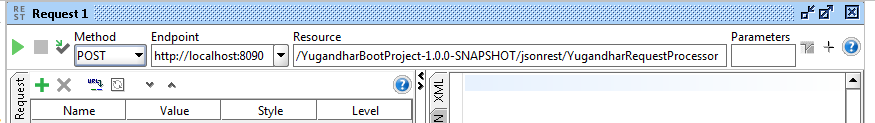
### Test using SOAPUI

Go to soap ui or the testing server and create new soap project. Create soap REST service.

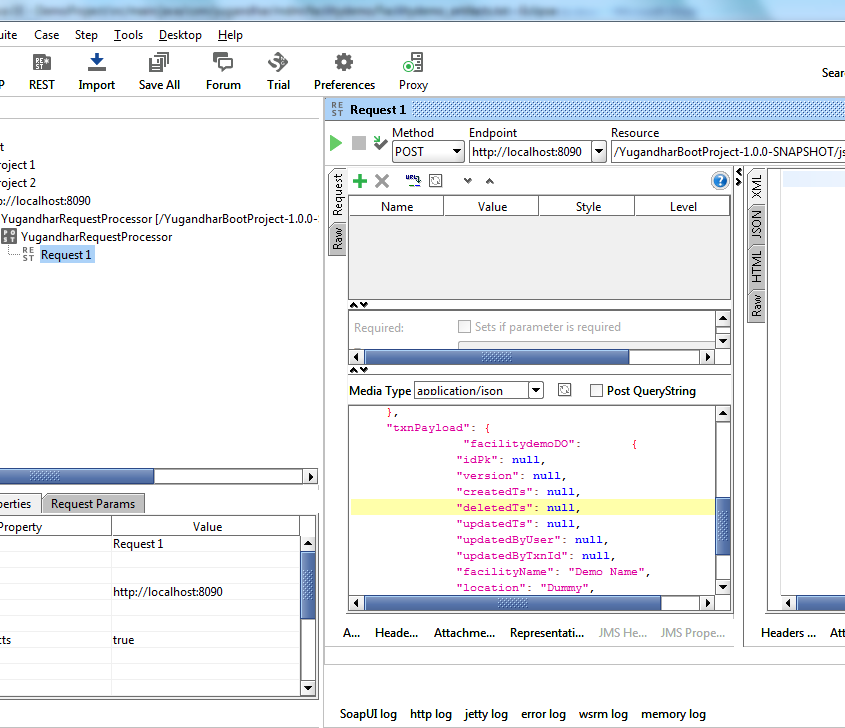
The REST url would be something like below

<http://localhost:8090/YugandharBootProject-1.0.0-SNAPSHOT/jsonrest/YugandharRequestProcessor>

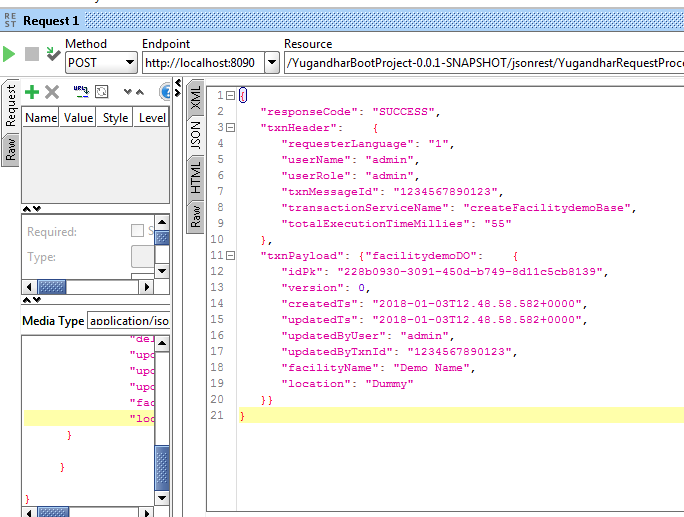
right click on SOAP project and click ‘new REST Service from URI’ and click ok



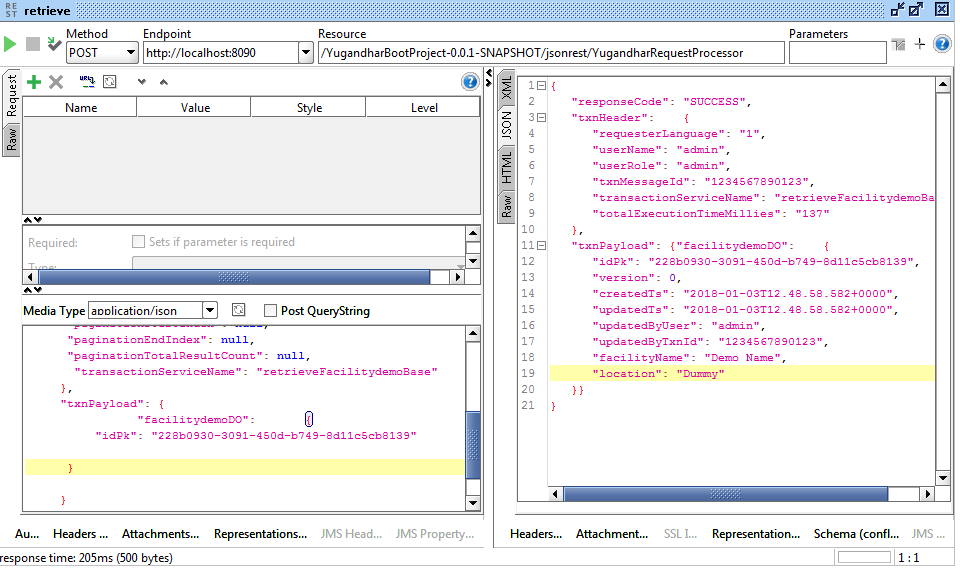
In the application/json, give the sample message from generated artifacts.txt file. Change the attributes e.g. facilityName and location and then execute the step.



You should see the response like below



Likewise execute the update and retrieve



And Update as below

