Extend Sales Cloud with Oracle PaaS

How to Install, Extend and Deploy Doctor Patient application into Oracle PaaS

Contents

[Overview 3](#_Toc388634692)

[Assumptions 3](#_Toc388634693)

[Environment 3](#_Toc388634694)

[Architecture 4](#_Toc388634695)

[Use Cases Overview 5](#_Toc388634696)

[What has been developed 6](#_Toc388634697)

[Create Prescription (Doctor) 6](#_Toc388634698)

[View Patient History 7](#_Toc388634699)

[Data Sync Mapping UI 7](#_Toc388634700)

[Application setup 8](#_Toc388634701)

[Database scripts 8](#_Toc388634702)

[Opening JDeveloper Application and importing data 8](#_Toc388634703)

[Setting up WebLogic Server Datasource based on Oracle DBCS Datasource connection 14](#_Toc388634704)

[JDeveloper - Project Overview 18](#_Toc388634705)

[Packaging and Deploying HealthCare to Embedded WebLogic Server 34](#_Toc388634706)

[Packaging and Deploying HealthCare to Oracle Java Cloud Service 37](#_Toc388634707)

[Oracle Sales Cloud – Application Composer – JWT User Token 45](#_Toc388634708)

[Oracle Sales Cloud Custom Objects 49](#_Toc388634709)

[Custom Object: Pres\_c 49](#_Toc388634710)

[Custom Object: Pres\_Med\_c 50](#_Toc388634711)

[Custom Object: Med\_c 51](#_Toc388634712)

[Steps to create a custom object 53](#_Toc388634713)

# Overview

The purpose of this document is to guide you how to install, extend and deploy the DoctorPatient demo application. DoctorPatient demo application is a Fusion Web Application (ADF application) hosted in Oracle Java Cloud Service 13.2 (PaaS) which extends Oracle Sales Cloud providing a Custom UI leveraging the rich set of components provided by ADF Faces Rich Client Components (e.g. DVT graphs). A set of extensibility patterns was covered on the DoctorPatient demo application as well as PaaS Features:

* Reuse of JCS shared-library
* Simplified UI Mashup of PaaS Custom application : subtab for Prescription
* Simplified UI look-and-feel
* Sales Cloud Household: Patient Information
* Custom Object in PaaS: Prescription
* Uses validated JWT UserToken for calling Sales Cloud WS
* Integration with Third Party Service: Pharmacy (Data Sync)

## Assumptions

* It is assumed that the reader is competent with Oracle JDeveloper and general Java development.
* Reader is aware of creating jar,war,ear deployment packages
* It’s also assumed that reader is aware of WebLogic deployment

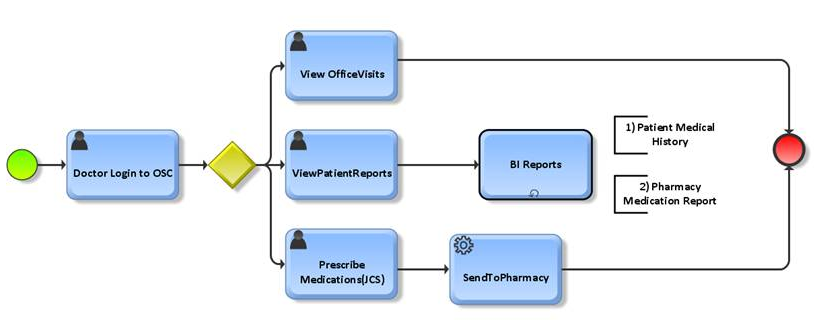
## Environment

* Oracle Sales Cloud Release 8
* Oracle Java Cloud Service 13.2
* Oracle Database Cloud Service
* Oracle JDeveloper 11.1.1.7.1
* Oracle Database 11g Expression Edition

## Architecture



## Use Cases Overview



* Prescribe Medications (Create Prescription)
* Send Medication to Pharmacy (Create Prescription)
* View Office Visits
* View Patient Reports

# What has been developed

## Create Prescription (Doctor)

List Prescriptions (Doctor)

## View Patient History



## Data Sync Mapping UI

# Application setup

## Database scripts

Go to your local database instance and if you don’t have created a database user for your project, you can run the script below. This script cannot be used in production; it’s just for testing purpose in a development environment. You can also find these scripts at the Database folder of your application.



Now you are able to create DoctorPatient model. In order to do that, run the script below with the user you just created above. Please note that there is a function which is only available in DBCS. You have to comment that code to run the ddl script in your local machine if not commented.



After running these scripts you can start working on JDeveloper. Please note that you are going to use a JDeveloper feature to load some data into your local database.

## Opening JDeveloper Application and importing data

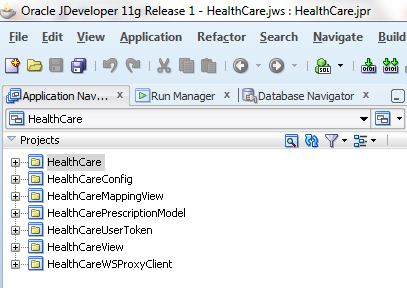
First step is to open your Application in JDeveloper. Our assumption is that you either have extracted the zip file into a folder in your machine or pulled out the source code from a source control repository (git, subversion, etc).

Start JDeveloper 11.1.1.7.1 and click on Application -> Open

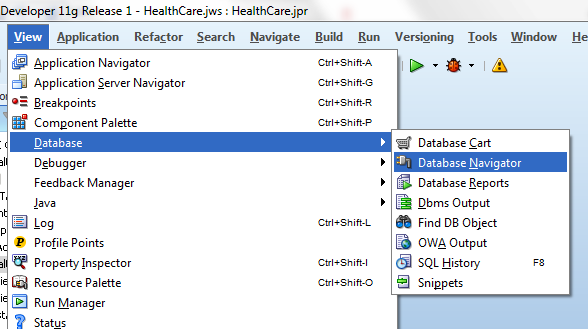
Locate the folder where all the source code is located (HealthCare). Select HealthCare.jws file and click on Open.



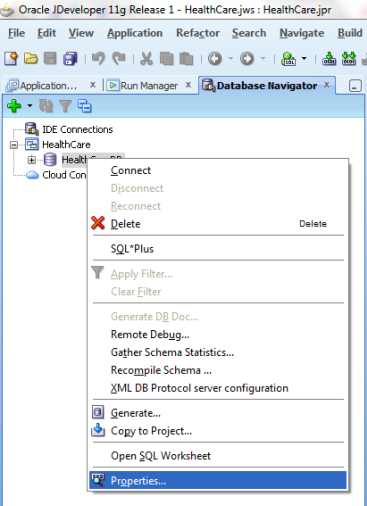
The following list of projects should be displayed in your Application:



Before proceeding, it’s good to load all the data which is required to run your application into the local database. Go to the main menu and select View->Database->Database Navigator



HalthCareDB connection should be available in your project. Just right click over HealthCareDB icon and select Properties… to change the connection settings.



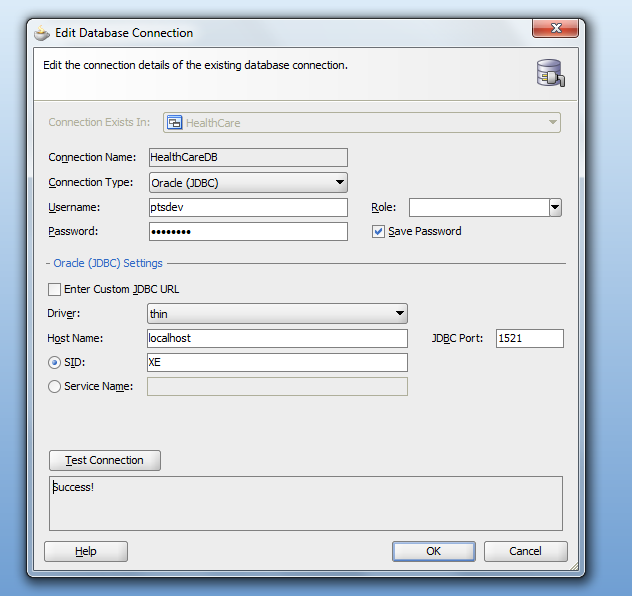
Set up the connection settings with username and password and other connections details which are required for your database. In your case, if you created your database user by using the sql script provided previously, you use the following username/password:

Username: ptsdev

Password: welcome1

SID: XE (this is because we are using Oracle Database 11g Expression Edition)

Then click on Test Connection button to validate the data you entered (make sure that Role field is empty):

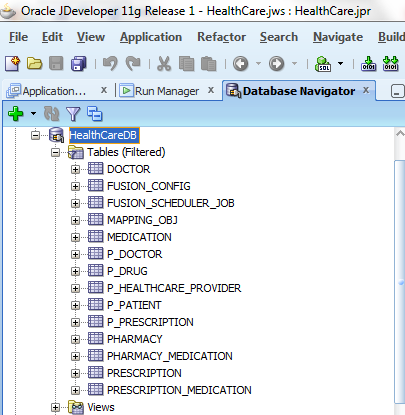


Next step is to load some data into database tables with the following .csv files. Save the files below into your project folder:

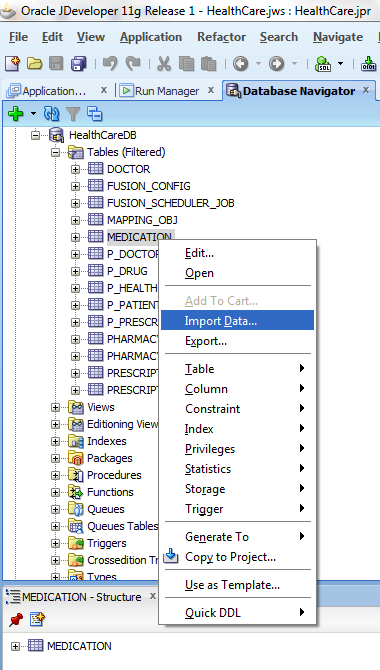
|  |  |
| --- | --- |
| Table | File |
| Medication |  |
| Pharmacy |  |
| Pharmacy\_Medication |  |
| P\_drug |  |
| P\_healthcare\_provider |  |

Before importing the data, make sure that the tool (e.g. Microsoft Excel) you are using to open those files, do not change the format of the numbers, e.g. 1.23567E18 instead of 1.2356789812318.

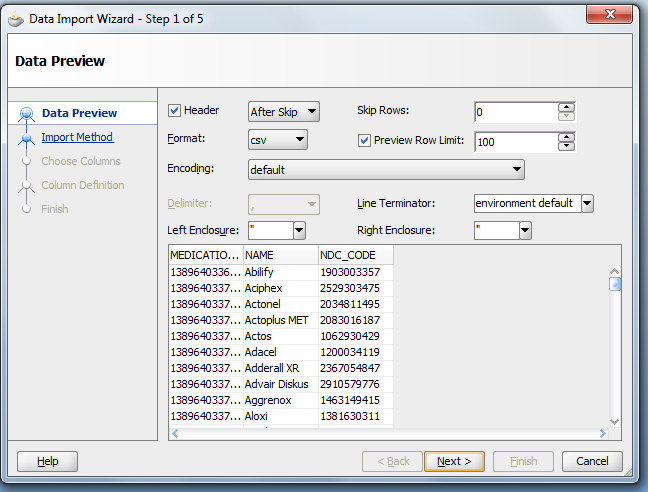
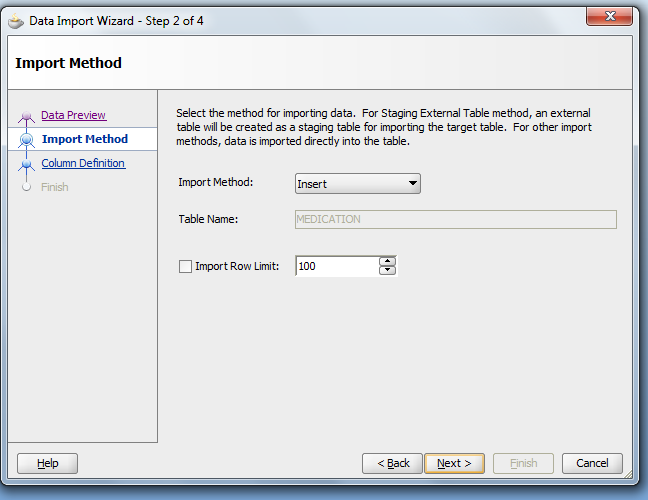
To import the data, double click on HealthCareDB icon to connect to the database instance, and then you should have access to some database objects. Click on the plus sign (+) of Tables icon to display all the tables you have created.

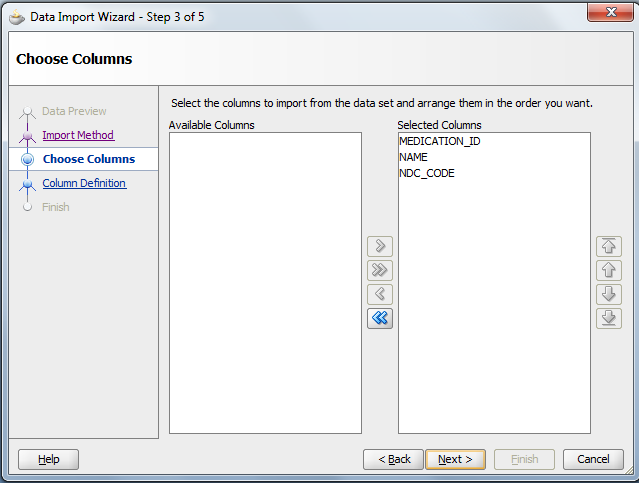
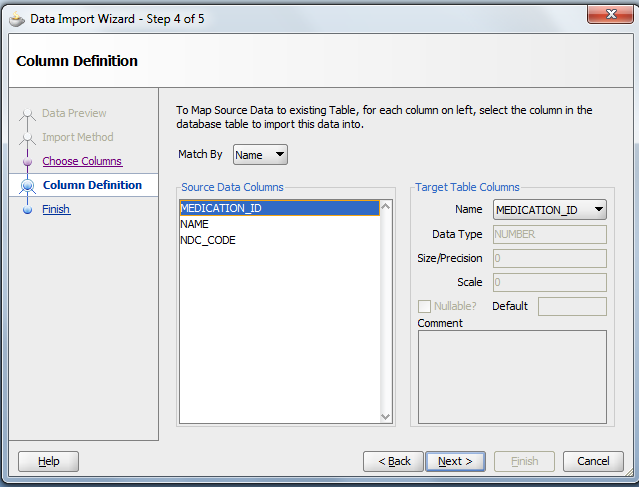


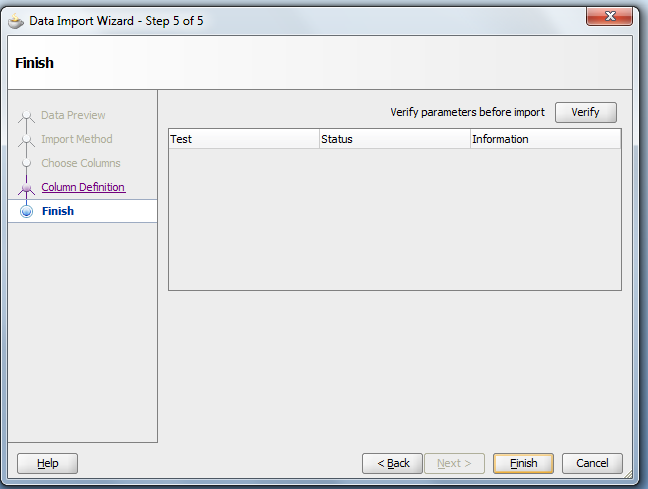
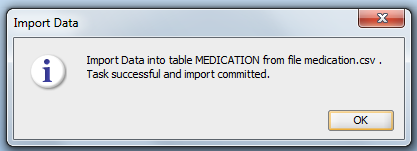
For each database table we mapped in the above table, you have to perform the following operation: Right-click on the table name <Medication> 🡪 Select Import Data…



Select the .csv file <medication.csv> and click on Open. Then, the Data Import Wizard window should come up. Click on Next up to the last window (Step 5), and finally you can click on Finish. Make sure you have imported all .csv files.

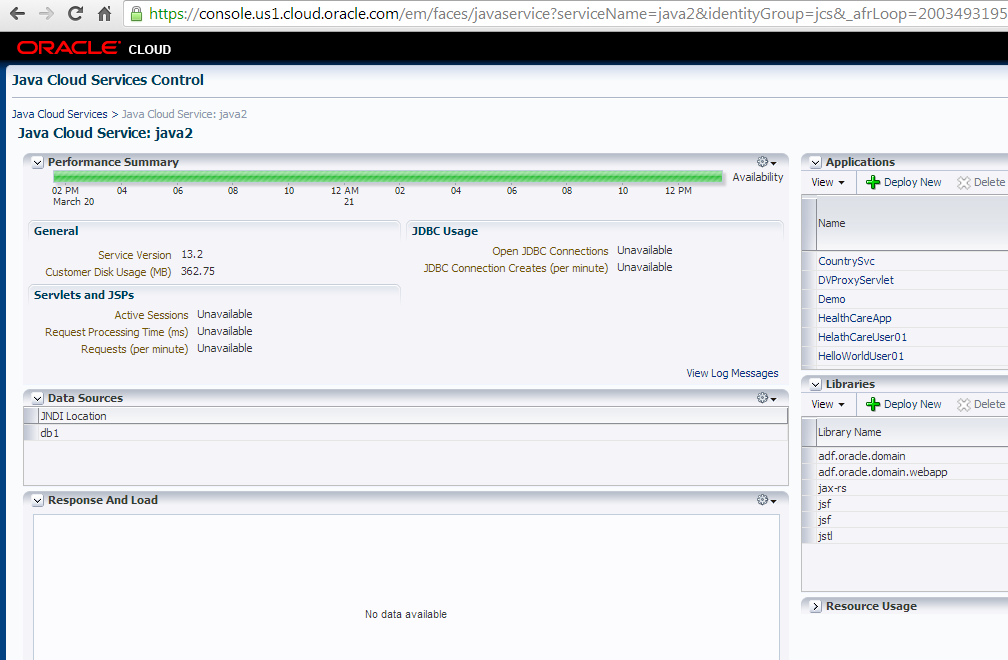
 

## Setting up WebLogic Server Datasource based on Oracle DBCS Datasource connection

Before moving on to the next step, it’s important to check the value of the Datasource name which was assigned for your Oracle Database Cloud Service instance.

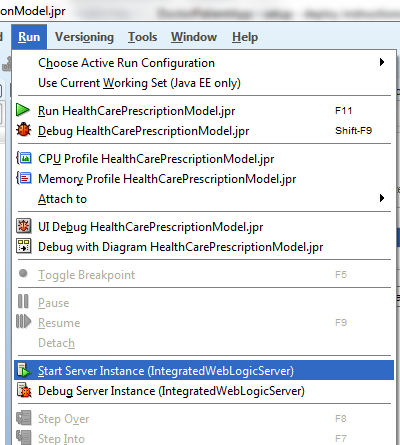
If you haven’t been provided this information, you can check the datasource name in the Java Cloud Service Control (JCS Console) as per the image below:



The JNDI Location should be set up in your Doctor Patient Application, at the projects which requires database connection.

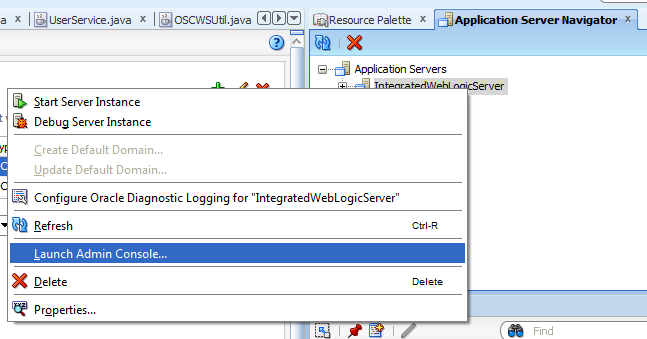
As a good practice, if you are planning to deploy your application first in your local machine in order to make some tests or while you are developing the application and don’t have access to the Oracle Java Cloud Service instance, you should create a Data Source Connection in your WebLogic Server (embedded to JDeveloper) and use the same as was set in DBCS.

As a first step to set up the Data Source Connection in embedded WLS, you should startup WLS from JDeveloper. Go to Run -> Start Server Instance. If it is the first time you are using Embedded WLS, you should be prompted by a username/password. Pick one and click on Ok.



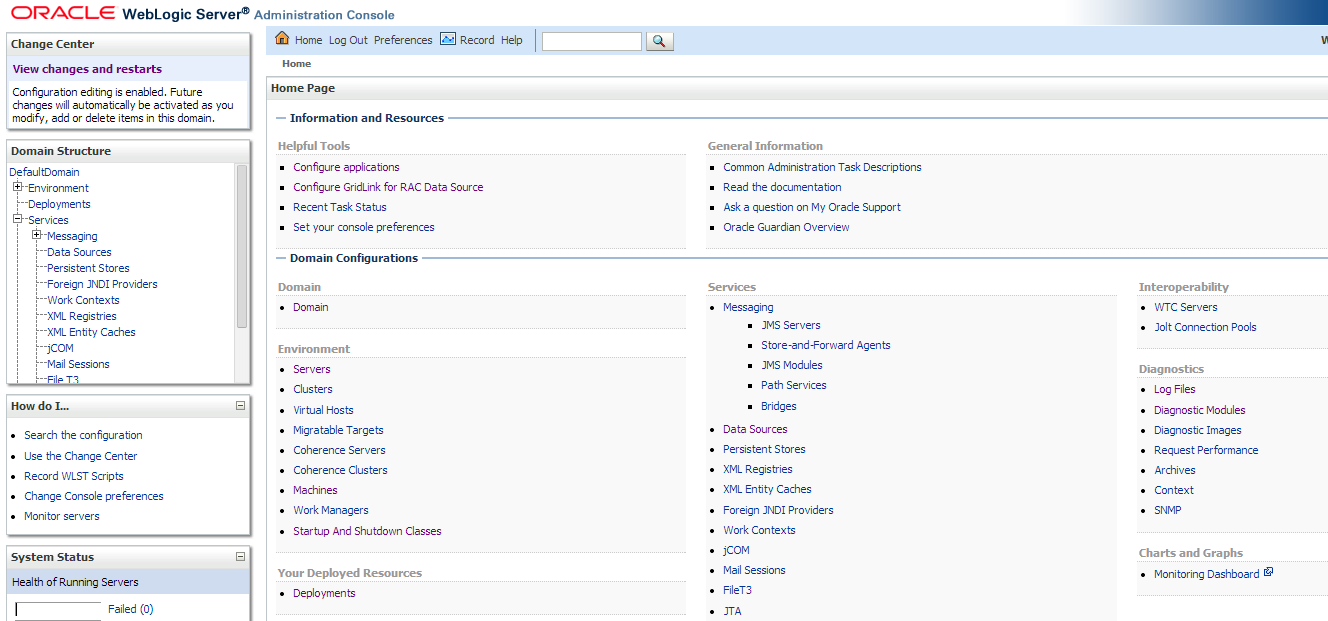
It would take awhile. Check the log message in JDeveloper console, until your server state goes to Running.

Go to the Application Server Navigator tab in JDeveloper. If it’s not displayed, click on View🡪 Application Server Navigator. Expand the Application Servers tree option and right click on IntegratedWebLogicServer, then select Launch Admin Console.



WebLogic Server 11g Administration Console window should be displayed in your browser window. Just fill out the form with the username and password you created and click on Login.

When the WebLogic Administration Console window comes up, click on the Data Sources link, under Home Page -> Domain Configurations -> Services, or under Domain Structure at the upper left corner of the window, under DefaultDomain, expand Services tree, and click on Data Sources Link.



At the Summary of JDBC Data Sources window, under Configuration tab, clicks on New -> Generic Data Source. Fill out the form with the required data, e.g.:

Name: db1

JNDI Name: db1

Database Type: Oracle

Then click on Next and use the default settings for that window. Next window you are required to set up the local database connection settings, e.g.:

Database Name: XE (SID of your database)

Host Name: localhost

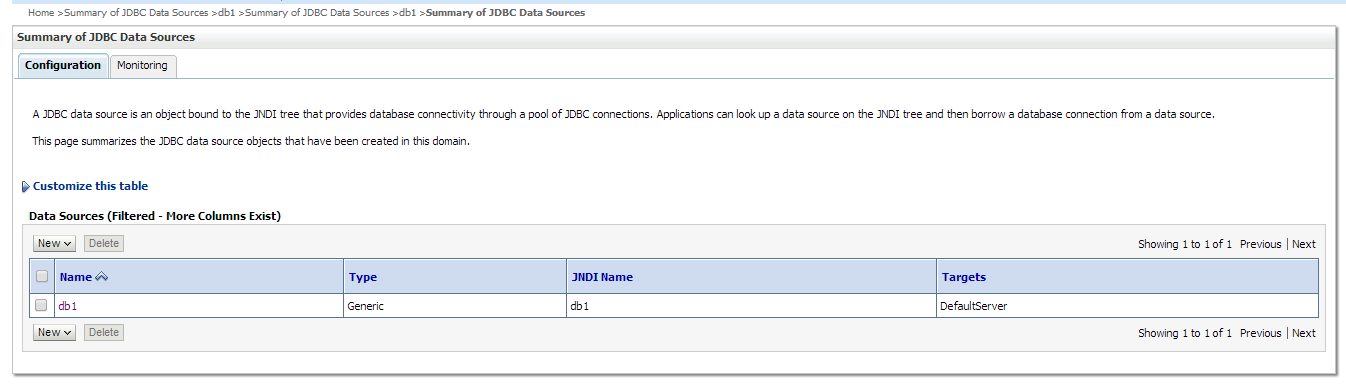
Port: 1521

Database User Name: ptsdev

Password/Confirm Password: welcome1

On next window, you should review your Datasource Connection setting. Click on Test Configuration button to check your connection. If it’s ok, click on Next.

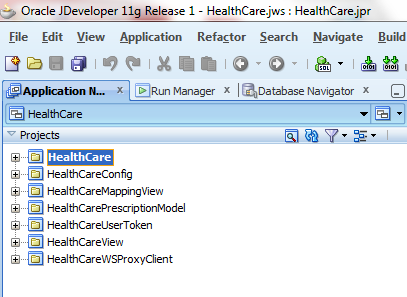
In the latest window you have to select a Managed Server which your Datasource connection should be available. Mark DefaultServer option and click on Finish. Your connection should be displayed in the Configuration table.



Ok, now you are ready to go back to JDeveloper and check the content of the demo application.

## JDeveloper - Project Overview

Doctor Patient Application (HealthCare) is composed by 7 Projects as displayed in the image below:



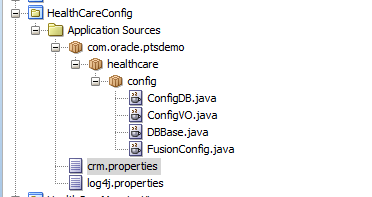
It’s important to follow all the instructions about how to prepare an application for Java Cloud Service Deployment. Check the documentation on this [link](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop002.htm#CSJSU7114).

Below, you can find a description of each project, what configuration files – either property files or deployment descriptors – need to be configured to deploy your application to a new environment. You also have to build the projects following the same order as described in the projects below, in case you have a problem to compile your application:

1. HealthCareConfig:

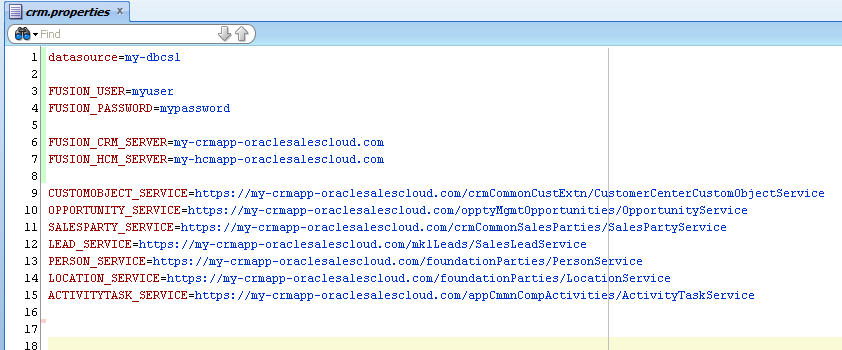
At this project you can find some resources related to Database configuration, Property Files/Resource bundle management. There are property files for [Apache log4j](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop002.htm#BABGFHHD) (click on the link), and for Oracle Sales Cloud web services. Basically, web service connection settings and logging are located on this project. Update the crm.properties file with the values related to your environment.

NOTE: The properties file is used here for convenience only. We recommend that, for production deployment, these properties should be stored in a database table in JCS/DBCS.



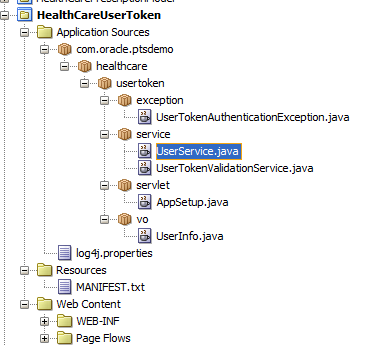
|  |  |  |
| --- | --- | --- |
| crm.properties | | |
| Property | Sample Value | Description |
| datasource | datasource=<dbcs schema name> | This property refers to the JNDI name of the data source required to connect to database. |
| FUSION\_USER | FUSION\_USER=<oracle sales cloud user> | This property refers to the username used to authenticate user against OSC in case where JWT token is not used. |
| FUSION\_PASSWORD | FUSION\_PASSWORD=<password> | This property refers to the password used to authenticate user against OSC. |
| FUSION\_CRM\_SERVER | FUSION\_CRM\_SERVER=<oracle-sales-cloud-instance-crm\_application> | Server address of Oracle Sales Cloud CRM application. |
| FUSION\_HCM\_SERVER | FUSION\_HCM\_SERVER=<oracle-sales-cloud-instance-hcm\_application> | Server address of Oracle Sales Cloud HCM application. |
| CUSTOMOBJECT\_SERVICE | CUSTOMOBJECT\_SERVICE=https://<my-oracle-salescloud-server >/crmCommonCustExtn/CustomerCenterCustomObjectService | OSC CustomerCenter Service endpoint |
| OPPORTUNITY\_SERVICE | OPPORTUNITY\_SERVICE=https:// <my-oracle-salescloud-server >/opptyMgmtOpportunities/OpportunityService | OSC Opportunity Service endpoint |
| SALESPARTY\_SERVICE | SALESPARTY\_SERVICE=https://<my-oracle-salescloud-server>/crmCommonSalesParties/SalesPartyService | OSC SalesParty Service endpoint |
| LEAD\_SERVICE | LEAD\_SERVICE=https:// <my-oracle-salescloud-server >/mklLeads/SalesLeadService | OSC Lead Service endpoint |
| PERSON\_SERVICE | PERSON\_SERVICE=https://<my-oracle-salescloud-server >/foundationParties/PersonService | OSC Person Service endpoint |
| LOCATION\_SERVICE | LOCATION\_SERVICE=https://<my-oracle-salescloud-server >/foundationParties/LocationService | OSC Location Service endpoint |
| ACTIVITYTASK\_SERVICE | ACTIVITYTASK\_SERVICE=https://<my-oracle-salescloud-server >/appCmmnCompActivities/ActivityTaskService | OSC Task Service endpoint |

See below an example:



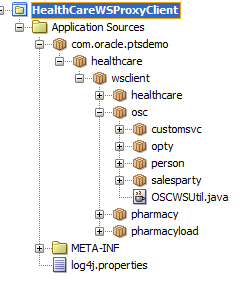
2. HealthCareUserToken:

At this project, you can find all the classes required to validate JWT token against Oracle Sales Cloud UserService WS. All the business logic required to validate JWT token can be found on UserService class.



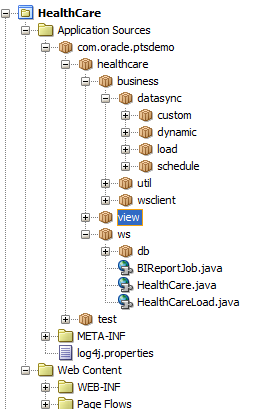
3. HealthCareWSProxyClient

At this project, you can find all the Java Proxy Client classes created to access some of the web services used by this project: Oracle Sales Cloud and Pharmacy (3rd Party). There is also a utility class – OSCWSUtil – which contains most of the methods responsible by set up HTTP Header / OWSM policies for web service calls.

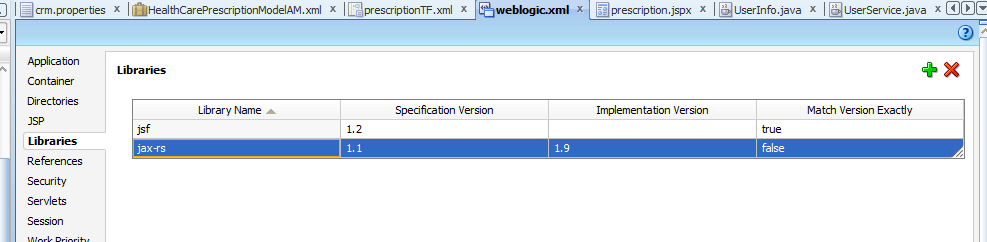


4. HealthCare

At this project, you can find all the logic used to manage and mapping custom fields from OSC to 3rd Party web service. JDBC and JAX-WS Handler to introspect SOAP payloads, JAX-RS to expose a Rest Service which is consumed by DBCS scheduler were used on this project.

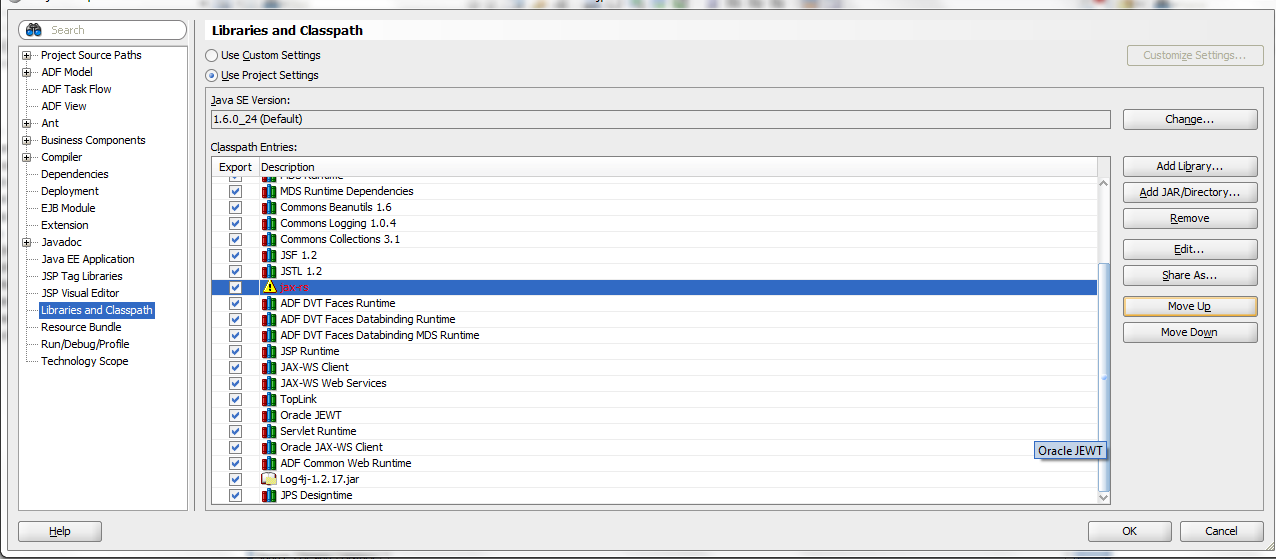


Note: Since JCS 13.2, it’s possible to deploy and re-use ADF shared libraries available in JCS. On HealthCare project we are reusing jstl and Jersey jax-rs libraries. It’s required to add an entry for each library we want to reuse into the weblogic.xml deployment descriptor as per the image below. More information about Oracle Java Cloud Service shared-libraries is available in the [JCS documentation](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop002.htm#BCEFFHEH).

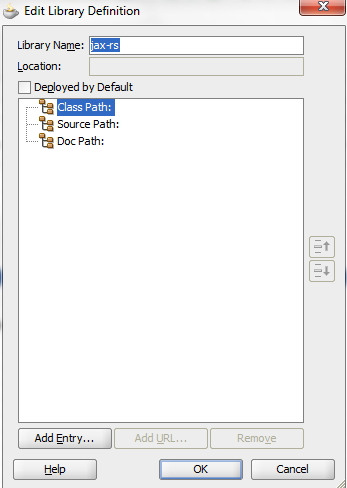


In addition, to compile your project you are required to add jax-rs library to the Classpath of HealthCare project. Oracle WebLogic Server ships with a set of pre-built shared libraries, packaged as Web Applications, however, in the embedded WebLogic Server, the jersey library is not available as the required version. Because of that, you can find a set of .jar file in the jax-rs folder in the application root folder. You can find more details about pre-built shared libraries available in WebLogic Server at the [documentation](http://docs.oracle.com/cd/E23943_01/web.1111/e13734/rest.htm).

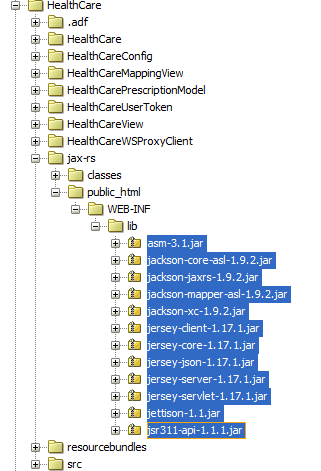
In order to setup the HealthCare project, you have to go to the Libraries and Classpath window. Right-click on the HealthCare project and select “Project Properties”. Then, select “Libraries and Classpath” node. At this window, you can find an entry to the jax-rs library as per the image below:



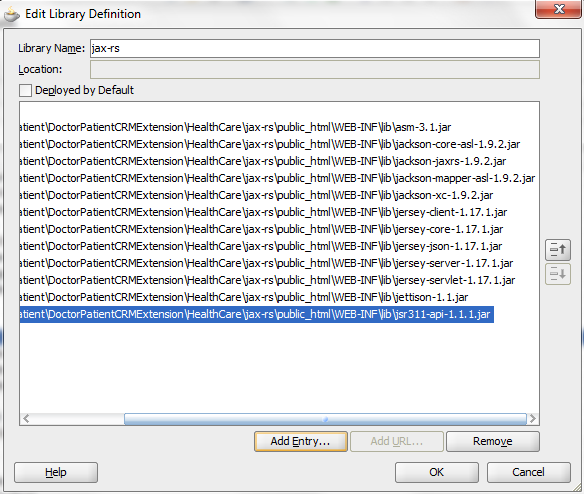
Select jax-rs entry and click on Edit button. The Edit Library Definition window displays.



Click on Add Entry… to select all files which should be used as a library. You should select all .jar files located at the “jax-rs\public\_html\WEB-INF\lib” folder, under the project root folder of your workspace (HealthCare).



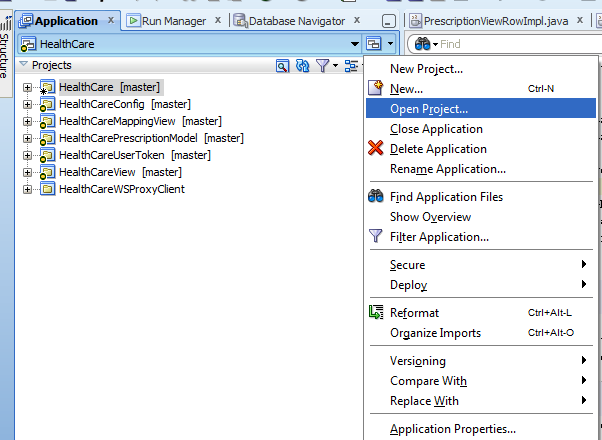
Then click on OK to confirm all files and OK again to close all the windows.



Rebuild your HealthCare project to confirm that all the compiling errors disappear.

Note: If you want to deploy your application to embedded WebLogic server using jax-rs as a shared library, instead of include all jersey files into your deployment package. You need to have Jersey jax-rs library deployed to your Oracle WebLogic server instance as a Shared Library. There is a JDeveloper project within the jax-rs folder of your project workspace named jax-rs.jpr.

You can add that project to your current application and after deploying the library remove from your application, or you can create a new application only to perform that deployment operation. To add the project to your current application, go to Application Menu at the Application Navigator window and click on Open Project.

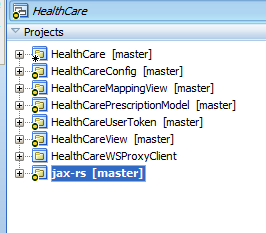


Click on Application Menu and Select Open Project

Then select the file jax-rs.jpr located on jax-rs folder, the same one which contains all jar files.



Now jax-rs project entry was added to the HealthCare application.



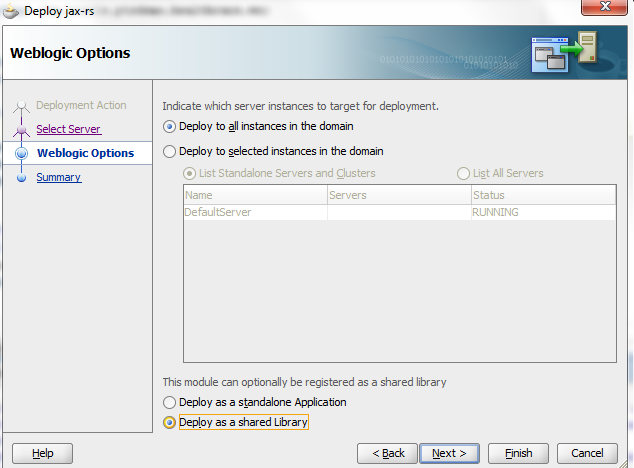
To deploy the shared library by using the JDeveloper wizard, right click on jax-rs project. Then, select Deploy 🡪 jax-rs.



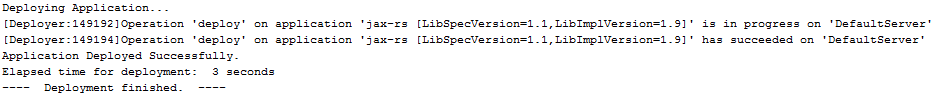
On the next window, select the option “Deploy to Application Server” and click on Next.



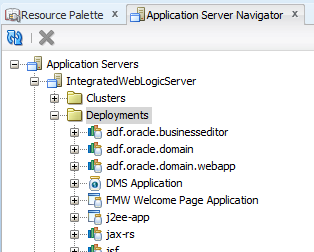
Then, select the Integrated WebLogic on the next Window and click on Next. Finally, at Weblogic Options window, mark the option “Deploy as a shared Library” and click on Finish.



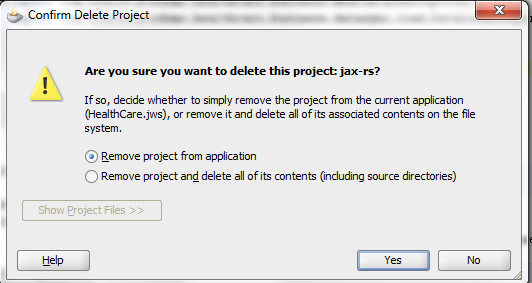
Check the deployment log confirming that the shared library was deployed to your embedded Weblogic Server.



You can use the Application Server Navigation window (Ctrl + Shift + G) to review all libraries and applications deployed in your WebLogic Server.

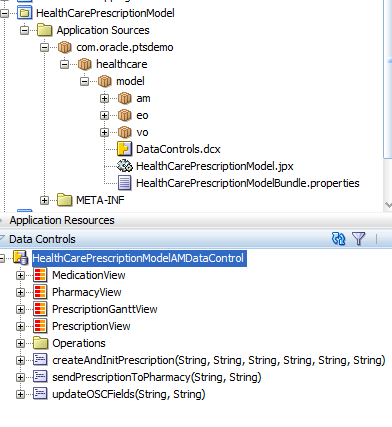


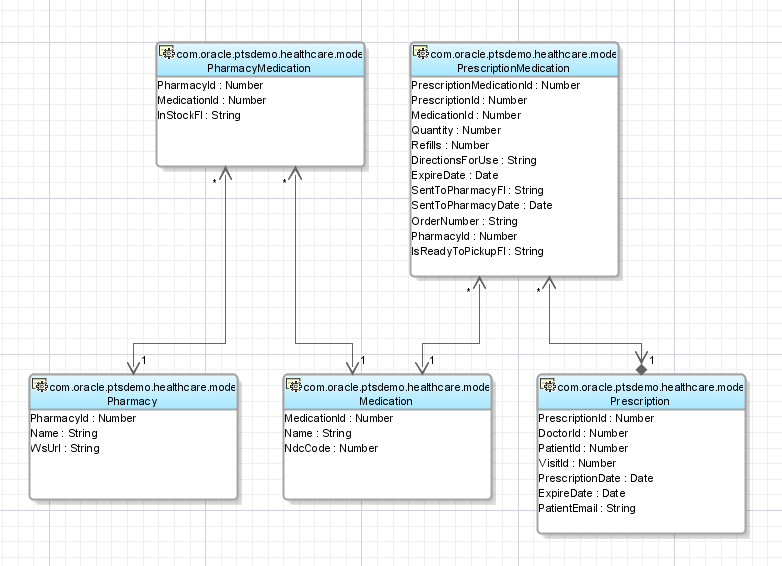
Finally, if you want to, you can remove jax-rs project from your Application, by right-clicking on the jax-rs project at the Application Navigator window and then select “Delete Project”. When prompted, just remove project from application so that it will become available for further usage.



5. HealthCarePrescriptionModel

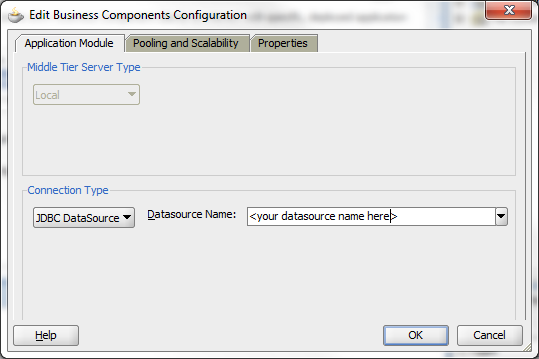
At this project you can find most of the business logic used to manage custom objects stored in Oracle DBCS. This one is an ADF Model/Business Service project which contains Entity Objects, View Objects and some business methods implemented in Application Model. There is also a Data Control which is consumed by the binding layer in HealthCareView project.





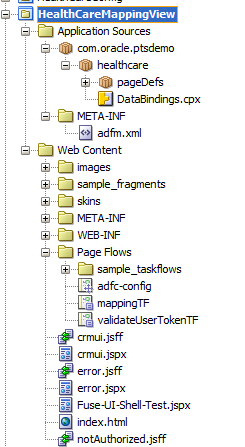
Please check the Datasource Name value assigned to the Application Module. Go to the HealthCarePrescriptionModelAM.xml file and, then select Configurations tab. Click on HealthCarePrescriptionModelAMShared under Configurations if this is selected as the Default Configuration, otherwise, change the value assigned to the HealthCarePrescriptionModelAMLocal entry.

The Edit Business Components Configuration screen should come up with the value assigned for the Datasource per the image below. Check if matches with the value you have in your local and cloud environments.



6. HealthCareMappingView

At this project, you can find the pages to display the mapping UI, which is based on classic UI of OSC (Skyros skin). Basically, there is a crmui.jspx page which contains a static region to display mappingTF which is an ADF Bounded Taskflow. This taskflow has an input parameter which is the JWT token variable, which is sent as an URL parameter from the Oracle Sales Cloud request (trough a UI Mashup).



7. HealthCareView

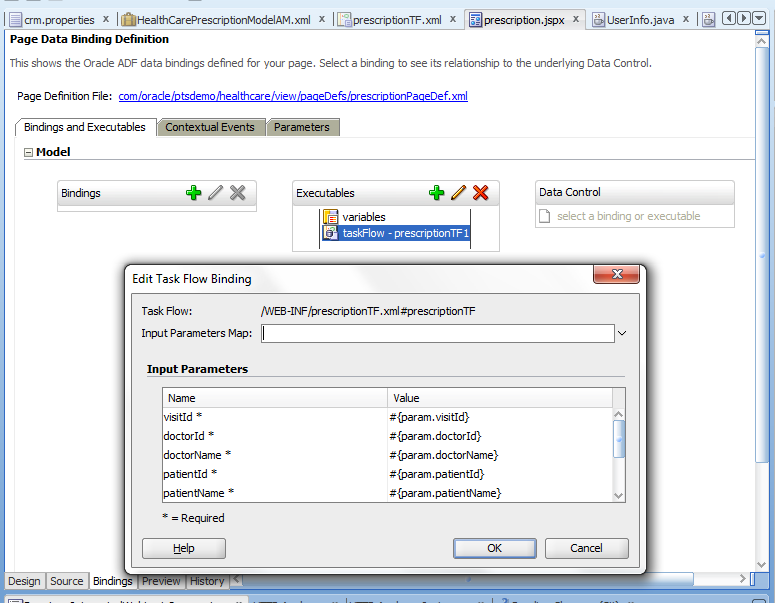
At this project, you can find the pages to display the Create/List Prescriptions and Prescription History. Most of the UIs created for the DoctorPatient application was designed on this project based on Simplified look and feel of OSC (Fuse skin). The start point for this project is the prescription.jspx page, which contains a static region to display prescriptionTF bounded taskflow. At this taskflow you can find most of the UI logic used to display one page or another, or to prepare some parameters to be used within a page, or even to call another action or flow. JWT token has also been used to validate OSC requests.

# 

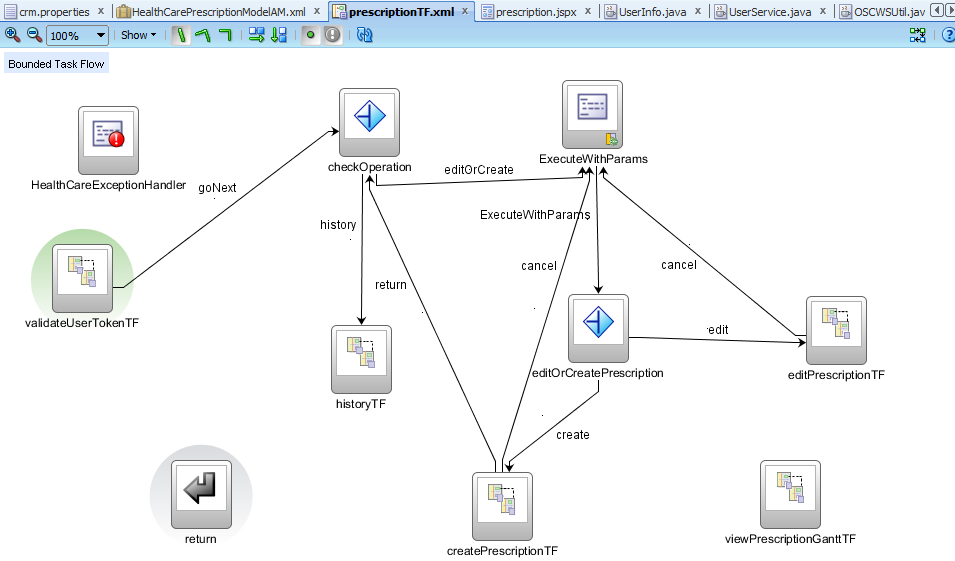
One of the integration flows between Oracle Sales Cloud and our demo application hosted in Oracle JCS is to have OSC calling the JCS application and passing some parameters through the URL. For instance, OSC pass the following parameters in the create prescription scenario:

* visitId (Opportunity.optyId)
* doctorId (Opportunity.OwnerResourcePartyId)
* doctorName (Opportunity.PartyName1)
* patientId (Opportunity.PrimaryContactPerson.PartyId)
* patientName (Opportunity.PrimaryContactPerson.PartyUniqueName).

One way to set up these parameters in the HealthCareView project is to open up the prescription.jspx file, which is the page called from OSC. Then, under Bindings tab, you have to double click taskFlow – prescriptionTF1 under Executables section of the Page Data Binding Definition. When the window comes up, you have to provide the values required to use that taskflow, as per the image below:



You can open prescriptionTF bounded taskflow and verify all the logic to display the pages:

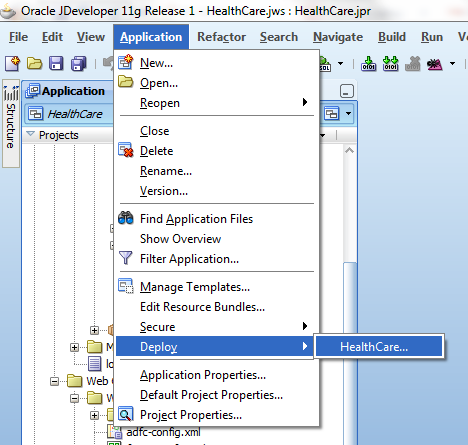


Note that the validateUserTokenTF is the default activity because that taskflow is the one responsible to validate the security token provided by OSC to avoid any invalid request call to OSC. The JWT Token is used as a security mechanism used as an alternative to HTTP Header/Basic authentication.

Building the application

## Packaging and Deploying HealthCare to Embedded WebLogic Server

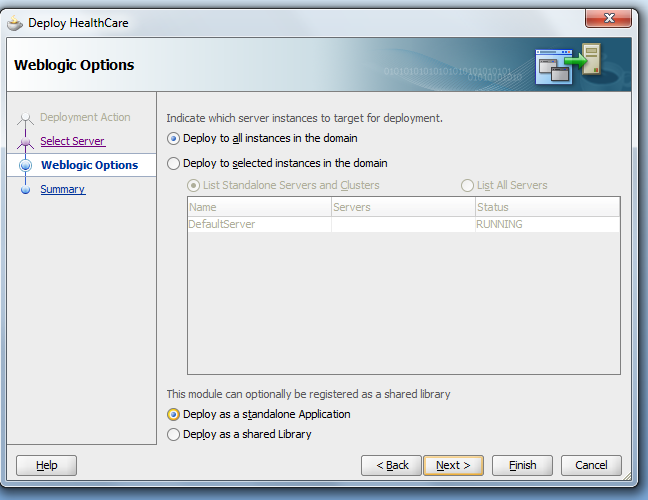
It’s time to package your application and deploy to embedded WebLogic server. The application was designed to be deployed as an “.ear” deployment package. In JDeveloper, go to Application 🡪 Deploy 🡪 HealthCare.



On Deployment Action screen, select Deploy to Application Server, and click on Next. Then, select IntegratedWeblogicServer.



On WebLogic Options screen, use Default options and click on Finish.



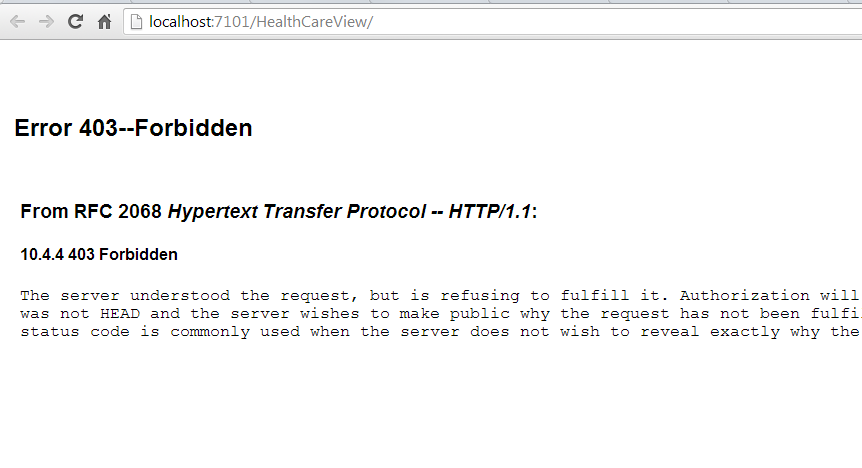
Check the status of the deployment in the Deployment – Log tab:

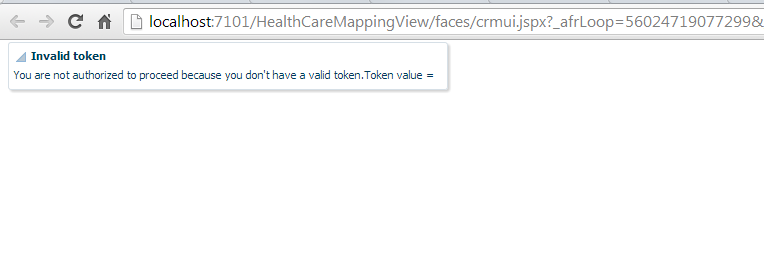


There are 3 URLs available as the result of your deployment, which can be used to access the application:

<http://localhost:7101/HealthCare>, <http://localhost:7101/HealthCareView> and <http://localhost:7101/HealthCareMappingView>

If you try to open those links, you should get an error on both pages as displayed below:





That happened mostly because of the input parameters which are required for each project were missing.

If you want to open Create / List Prescription pages in your local WLS, you should use the following URL:

[http://localhost:7101/HealthCareView/faces/prescription?visitId=<OptyId>&doctorId=<OwnerResourcePartyId>&doctorName=<PartyName1>&patientId=<PrimaryContactPerson.PartyId>&patientName=<PrimaryContactPerson.PartyUniqueName>&patientEmail=<PrimaryContactPerson.Email.EmailAddress>&jwt=<jwtToken>](http://localhost:7101/HealthCareView/faces/prescription?visitId=%3cOptyId%3e&doctorId=%3cOwnerResourcePartyId%3e&doctorName=%3cPartyName1%3e&patientId=%3cPrimaryContactPerson.PartyId%3e&patientName=%3cPrimaryContactPerson.PartyUniqueName%3e&patientEmail=%3cPrimaryContactPerson.Email.EmailAddress%3e&jwt=%3cjwtToken%3e)

Where we have

<OptyId> = OSC Opportunity Id

<OwnerResourcePartyId> = OSC User Id

<PartyName1> = OSC User Name

<PrimaryContactPerson.PartyId> = OSC Sales Account or Household Contact’s id, related to an Opportunity

<PrimaryContactPerson.PartyUniqueName> = OSC Sales Account or Household Contact’s name, related to an Opportunity

<PrimaryContactPerson.Email.EmailAddress> = OSC Sales Account or Household Contact’s e-mail address related to an Opportunity

<jwtToken> = OSC JWT Token

If you want to open Patient History page in your local WLS, you should use the following URL:

[http://localhost:7101/HealthCareView/faces/prescription?history=y&patientId=<PartyId>&jwt=<jwtToken>](http://localhost:7101/HealthCareView/faces/prescription?history=y&patientId=%3cPartyId%3e&jwt=%3cjwtToken%3e)

Where we have

<PartyId> = OSC Sales Account or Household Contact’s id

<jwtToken> = OSC JWT Token

You should pick all those parameters from OSC or you need to create a groovy script in OSC to create that URL with those parameters for you.

If you want to open Mapping UI page in your local WLS, you should use the following URL:

[http://localhost:7101/HealthCareViewMapping/faces/crmui?jwt=<jwtToken>](http://localhost:7101/HealthCareViewMapping/faces/crmui?jwt=%3cjwtToken%3e) where we have

<jwtToken> = OSC JWT Token

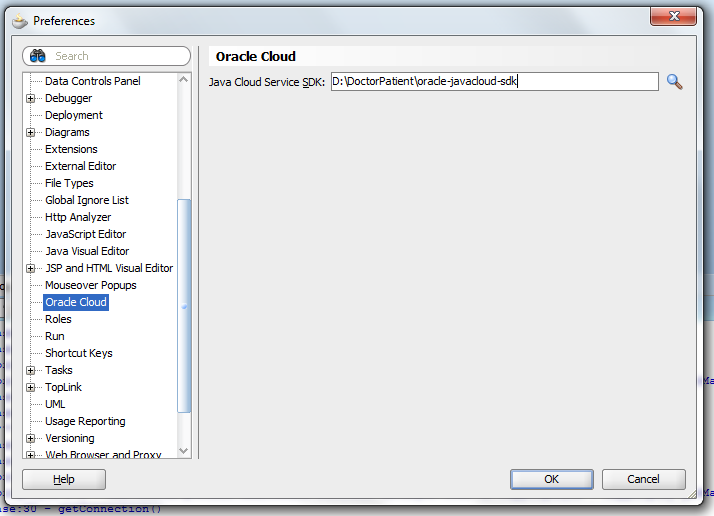
There is also a REST service which is consumed by DBCS Scheduler and can be accessed through the following URL:

<http://localhost:7101/HealthCare/jersey/BISchedulerJob>

## Packaging and Deploying HealthCare to Oracle Java Cloud Service

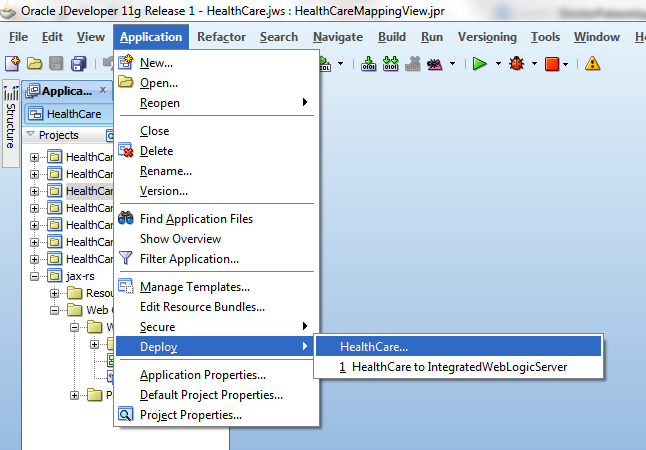
Since you developed and tested the application in your embedded WebLogic server, you are now almost ready to deploy your application to Oracle Java Cloud Service. Before deploying the application, it’s important to set up your JDeveloper to use JCS SDK. More information on how to download and install JCS SDK is available [here](http://docs.oracle.com/cloud/latest/javacs_common/CSJSU/java-develop001.htm#BCEFGHAF).

For now, after downloading and installing JCS SDK in your local machine, you need to go to JDeveloper preferences and update Oracle Cloud settings with the location of SDK Classpath.

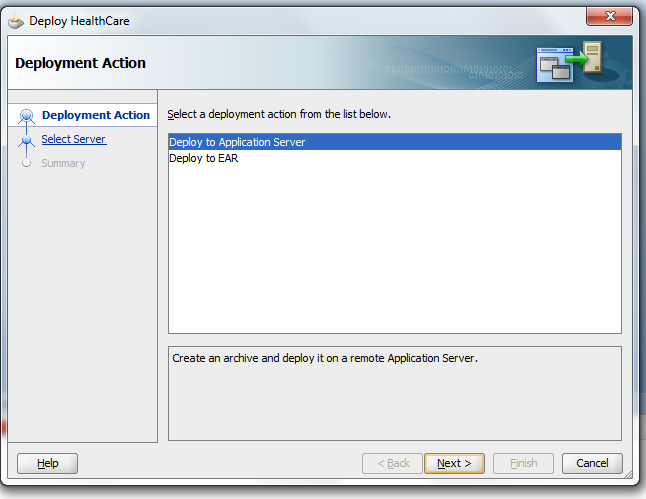


Ok, let’s start working with JCS. Since the application is working on local environment, we should promote the application to JCS. The same way we created a connection to embedded WebLogic Server in JDeveloper, we should follow the same instructions to create a connection to JCS.

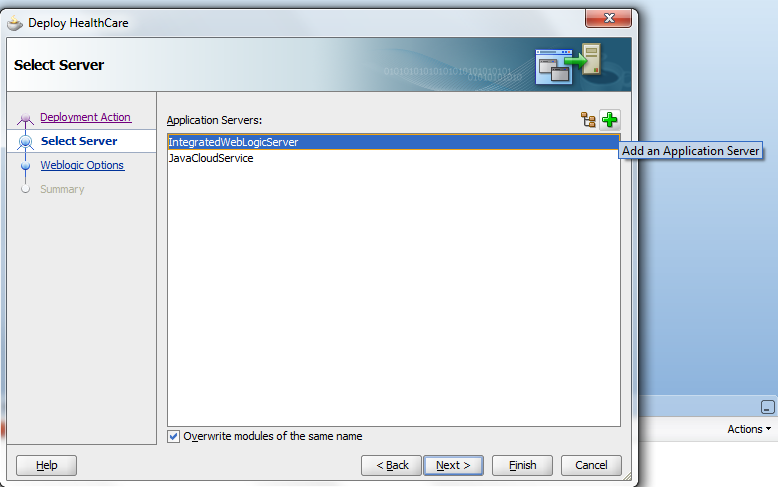
Go to Application 🡪 Deploy 🡪 HealthCare…



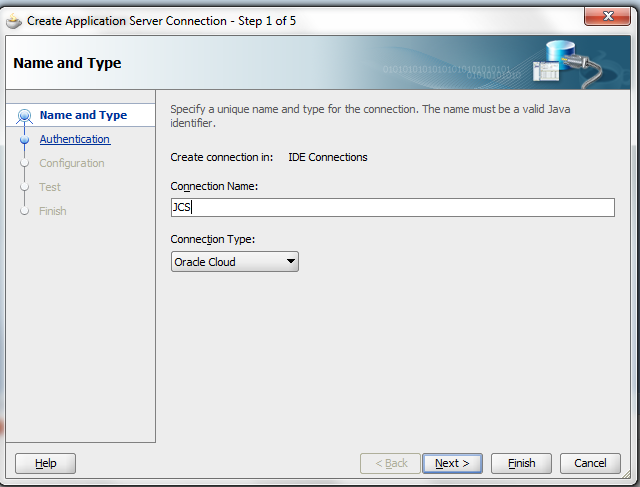
Select Deploy to Application Server:



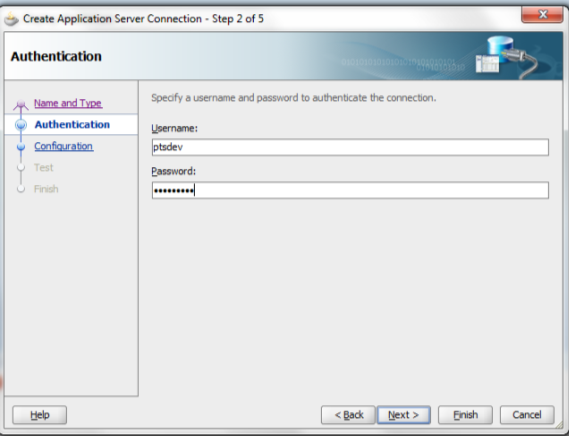
Click on the plus icon (+) to create a new connection



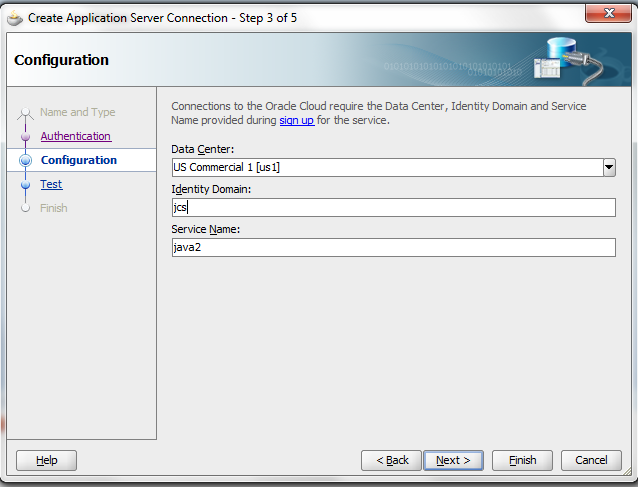
Type in a name for your connection and choose Oracle Cloud as a Connection type.



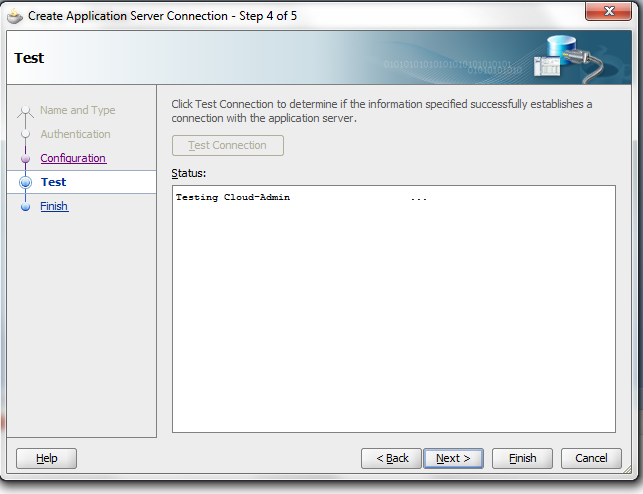
Fill out the form with the Username and Password for your JCS connection:



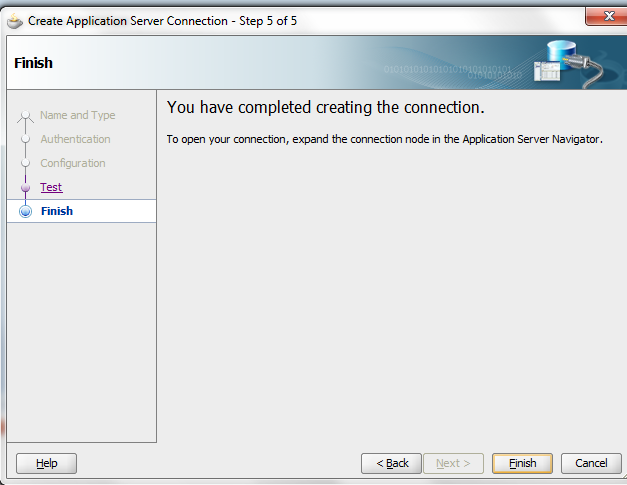
Provide the connection details related to the Data Center, Identity Domain and Service Name:



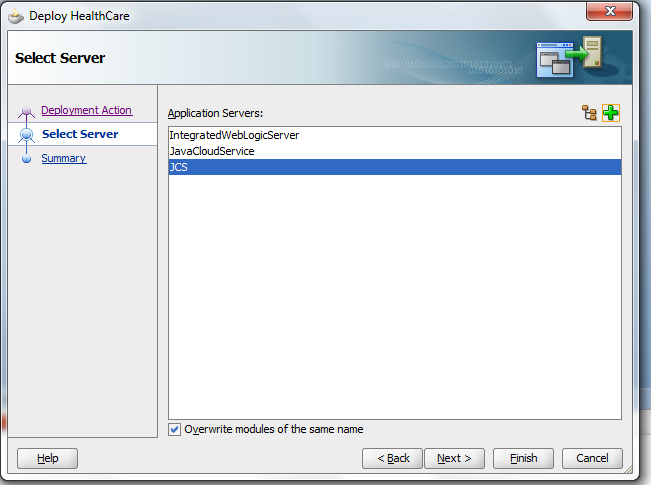
Test your connection:



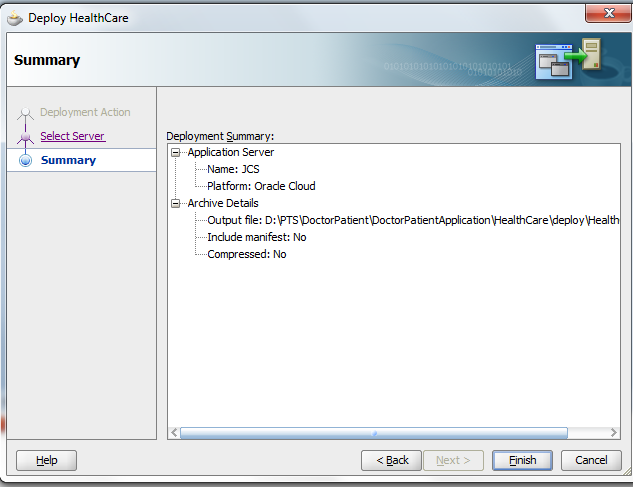
If you successfully connected to JCS, then click on Next and finally on Finish button complete your configuration to the service.



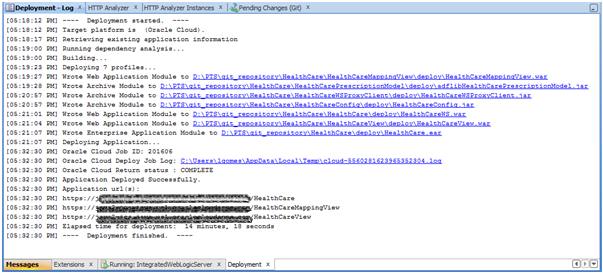
Now that your connection is set up, you just have to select the Application Server connection you created and click on Next.



Check the Deployment Summary and click on Finish to start deploying the application to Oracle Java Cloud Service.



Check the deployment log.



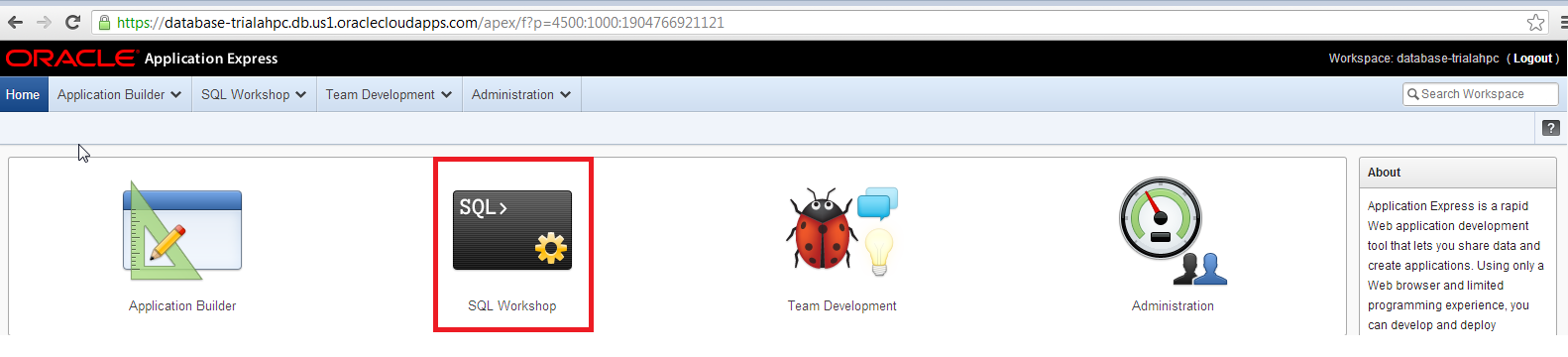
Now you are ready to setup your database structure in DBCS. It’s important to note that some applications will begin with a clean state – with data structures defined, but without any data populating those structures. The data is added through user interaction.

However, a more common scenario is an application where some amount of data already exists, although not in an Oracle Database Cloud Service. This scenario will require data to be loaded into the Oracle Database Cloud Service.

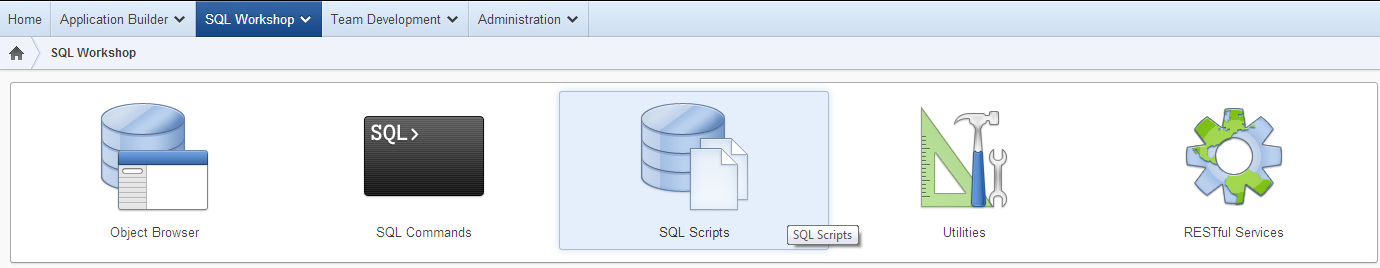
You can load data into your Oracle Database Cloud Service with two different tools – Application Express or SQL Developer. You can find more information [here](http://www.oracle.com/webfolder/technetwork/tutorials/obe/cloud/13_2/dbservice/dataload/dataload.html?cid=8710&ssid=0).

In this tutorial, we will create the data structure using Oracle Database Cloud Service Application Express.

Go to the DBCS service Console and select SQL Workshop.



Then Click on SQL Scripts.



On the upcoming screens, you can copy the content of the database scripts which has been provided at the beginning of this tutorial to create the database structure. Please note that there is a Stored Procedure within that script which needs to be updated with the values of the REST service which you will deploy or was deployed:

create or replace procedure HealthCareBIScheduler

is

l\_clob clob;

l\_buffer varchar2(32767);

begin

l\_clob := apex\_web\_service.make\_rest\_request(

p\_url => '<REST\_SERVICE\_URL>',

p\_http\_method => 'GET',

p\_username => '<USERNAME>',

p\_password => '<PASSWORD>'

);

l\_buffer := dbms\_lob.substr(l\_clob, 32000, 1 );

dbms\_output.put\_line(l\_buffer);

end;

Where:

<REST\_SERVICE\_URL> = https://<JCS\_SERVER>/HealthCare/jersey/BISchedulerJob

<USERNAME>= JCS username

<PASSWORD>= Password of JCS User

You also have to uncomment the create Scheduler job as per the script below:

BEGIN

CLOUD\_SCHEDULER.CREATE\_JOB (

job\_name => 'HealthCareBISchedulerJob',

job\_type => 'STORED\_PROCEDURE',

job\_action => 'HealthCareBIScheduler',

start\_date => sysdate,

repeat\_interval => 'FREQ=MINUTELY;INTERVAL=5',

end\_date => sysdate+365,

enabled => TRUE,

comments => 'Push custom object data back to OSC for BI reports');

END;

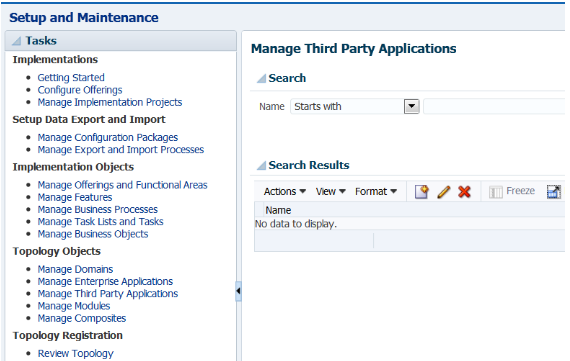
/

## Oracle Sales Cloud – Application Composer – JWT User Token

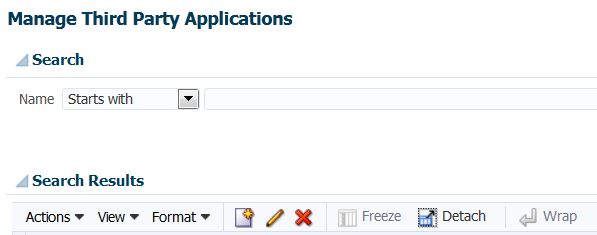
In order to invoke Web Services with end user context in a UI mashup use case, you should be required to use JWT User Token (since OSC R8). The Application Composer is the tool used to create UI Mashups or External Links, as well as to create Custom Objects or Custom Fields.

The first step is to register the service endpoint (JCS application URL) on the Topology Manager.

1. Go to Navigator 🡪 Set up and Maintenance, and choose option "Manage Third Party Applications". Make sure you have Application Implementation Consultant credentials to perform the rest of this task.



(b) Click on the "Create new" icon - this is in the Search Results pane with the icon looking like a "rectangle with a star on it" (or on the menu bar of the Search Results pane Actions 🡪Create)

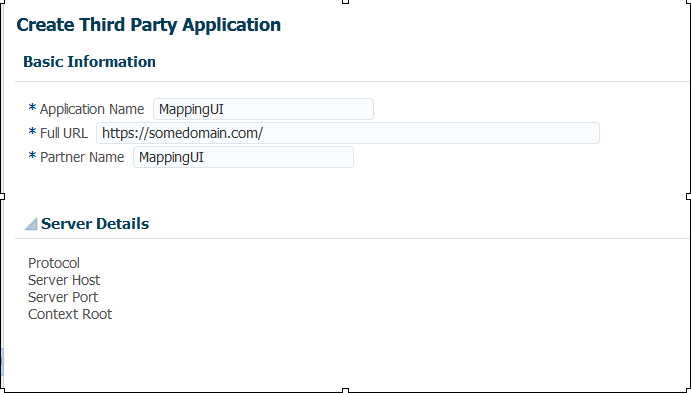


(c) Put in entry here corresponding with your application

Application Name: HealthCareWebApp

Full URL = https://<oracle-jcs-server>/HealthCareView/faces/prescription

Partner Name: HeathCareWebApp



(d) Click "Apply" and then "Save and Close"

Now you need to create the UI Mashup.

(e) Go to Application Composer 🡪 Common 🡪 Contact 🡪 Simplified UI 🡪 Pages 🡪 Edit Page 🡪 Create a new subtab 🡪 Web Page Component.

The script below allows you to insert the History page as a new subtab in Edit Page of Simplified UI.

def urlParam = "?history=y&patientId="+PartyId

//def serverAddress = "<oracle-jcs-server>"

def serverAddress = oracle.topologyManager.client.deployedInfo.DeployedInfoProvider.getEndPoint("HealthCareWebApp")

def jwt = (new oracle.apps.fnd.applcore.common.SecuredTokenBean().getTrustToken())

def finalURL = "https://"+serverAddress+"/HealthCareView/faces/prescription"+urlParam+"&jwt=" +jwt

return(finalURL)

To add the Create/View Prescription page into the Edit Opportunity screen in Simplified UI, you should go to Application Composer 🡪 Sales 🡪 Standard Objects 🡪Opportunity 🡪Pages 🡪Simplified Pages tab 🡪Details Page Layout 🡪Edit Layout 🡪 Web Page Component. And then create a new subtab, adding the following Groovy script:

//Subtab Groovy script in Opportunity

// //Add a new subtab (green "plus" button subtab)

//code below requires creating an endpoint on Topology Manager

def crmkey= (new oracle.apps.fnd.applcore.common.SecuredTokenBean().getTrustToken())

def urljwt = "&jwt="+crmkey

def patient = PrimaryContactPerson

def patientEmail = ""

def patientName = ""

def patientId = ""

if (patient != null) {

patientId = patient.PartyId

patientName = patient.PartyUniqueName

def patientEmailObj = patient.Email?.first()

patientEmail = patientEmailObj?.EmailAddress

}

else {

throw new oracle.jbo.ValidationException('There is no Patient associated with the Office Visit.')

}

def urlParam = "?visitId="+OptyId+"&doctorId="+OwnerResourcePartyId+"&doctorName="+PartyName1+"&patientId="+patientId+"&patientName="+patientName+"&patientEmail="+patientEmail+urljwt

def serverAddress = oracle.topologyManager.client.deployedInfo.DeployedInfoProvider.getEndPoint("HealthCareWebApp")

return serverAddress+urlParam;

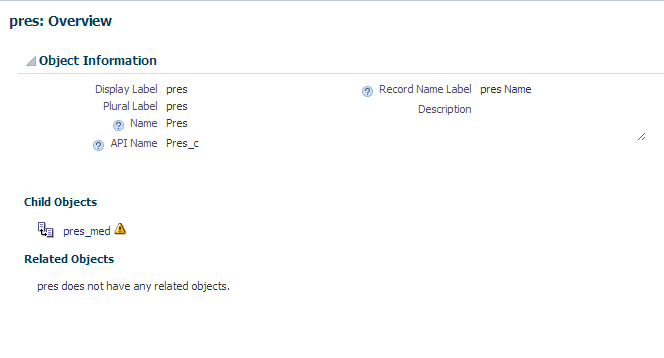
# 

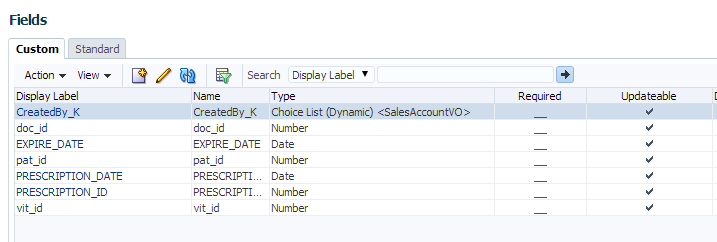
# Oracle Sales Cloud Custom Objects

There are some Custom Objects which need to be created under Customer Center Application by using Application Composer. Those objects are used by BI to generate some reports.

## Custom Object: Pres\_c

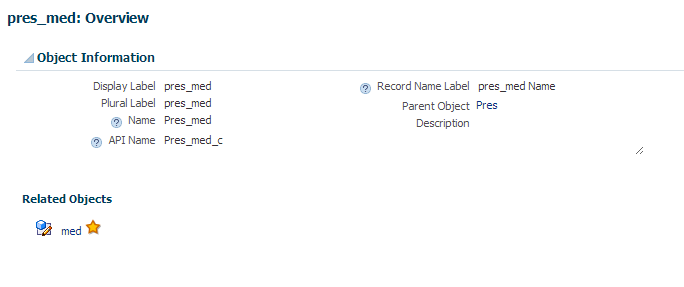
This object represents a Prescription.

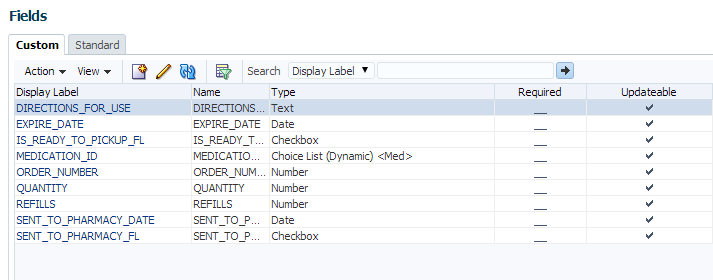




## Custom Object: Pres\_Med\_c

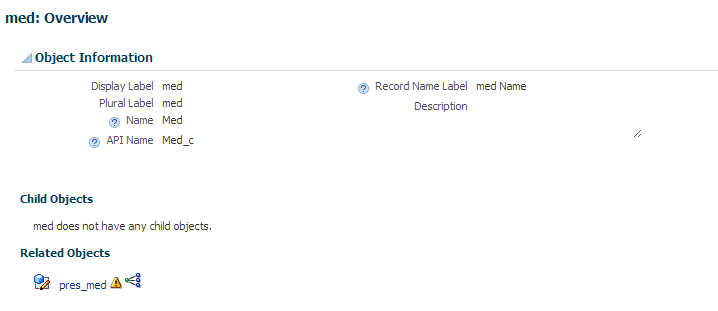
This object represents a relationship between Prescription and Medication.

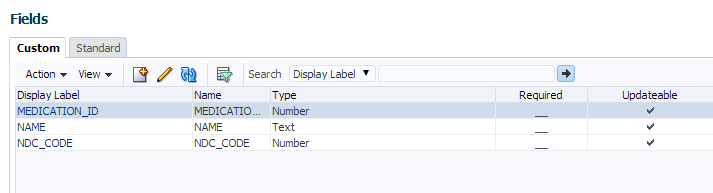




## Custom Object: Med\_c

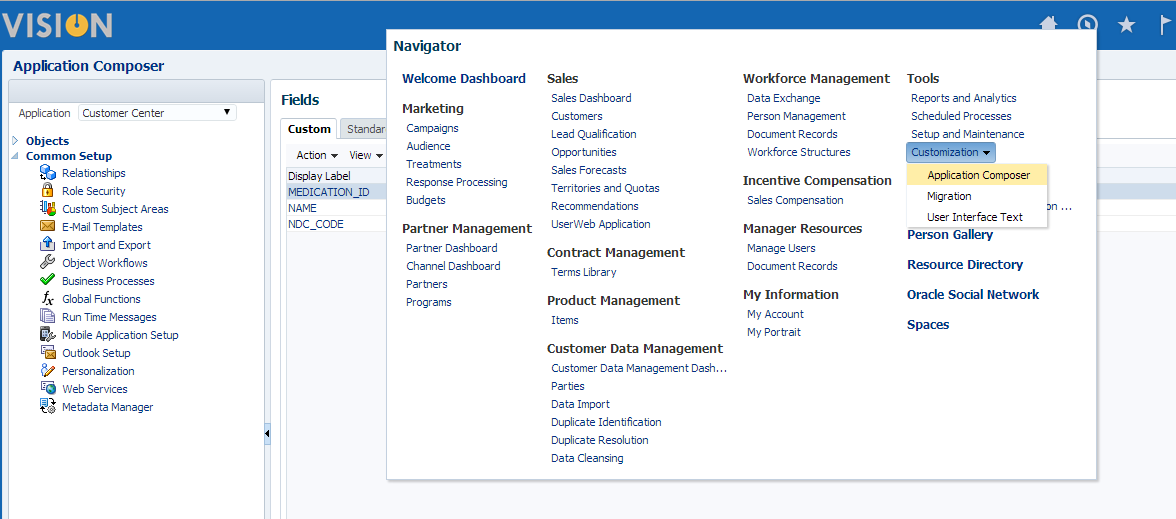
This object represents a Medication.



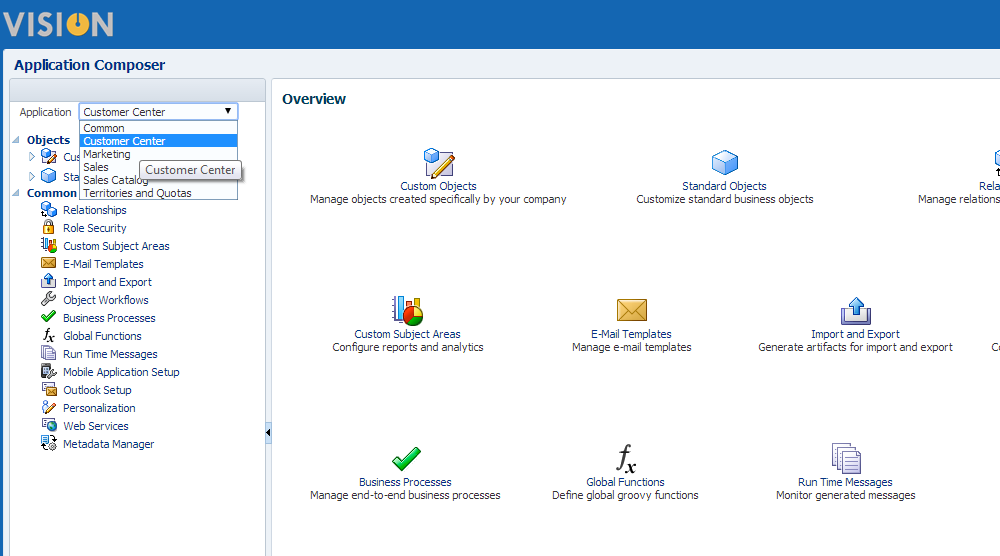


## Steps to create a custom object

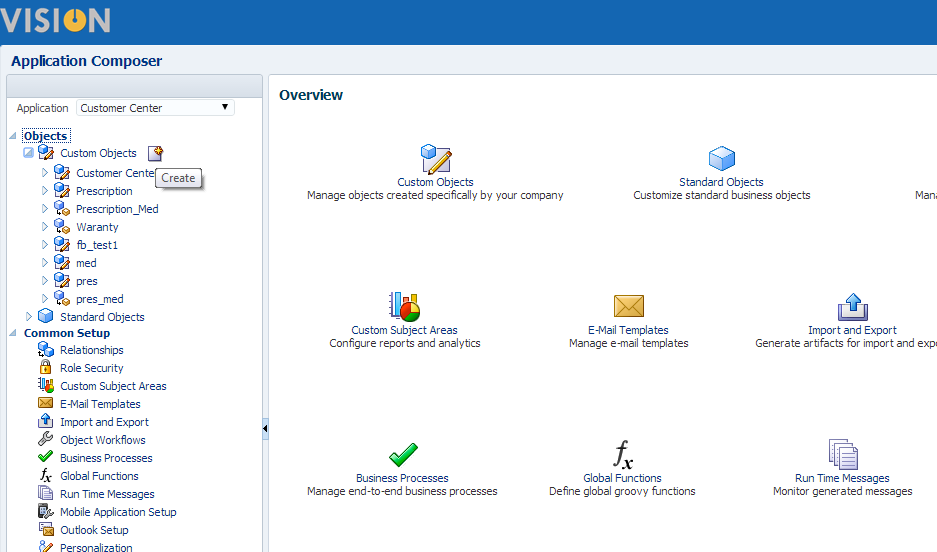
1. Click on Navigator -> Customization -> Application Composer



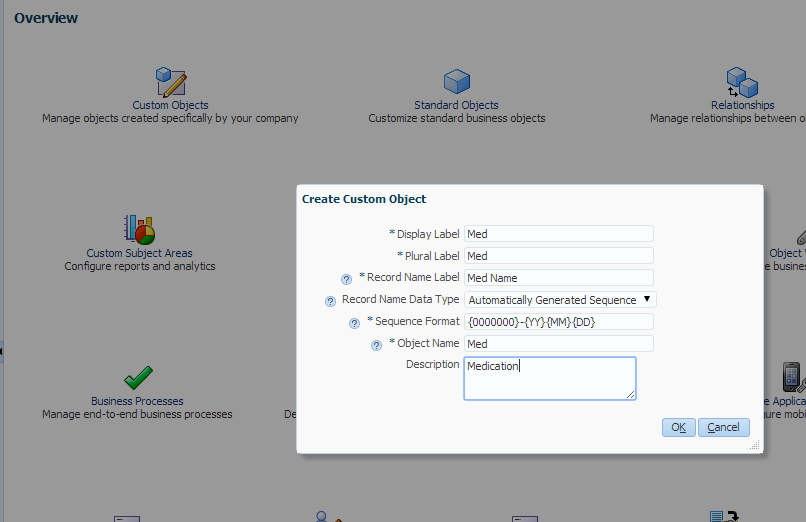
1. Choose Customer Center



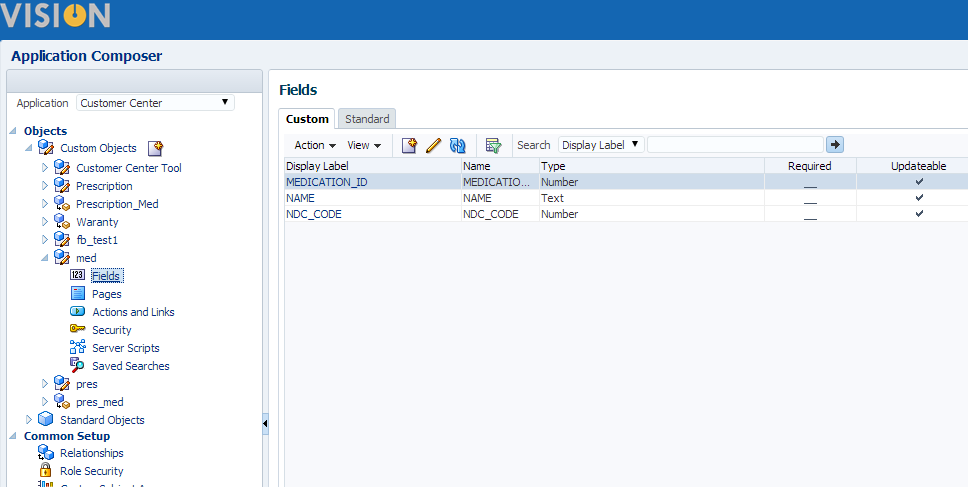
1. Click a ‘+’ icon to create a custom object



1. Fill up the required fields and click OK



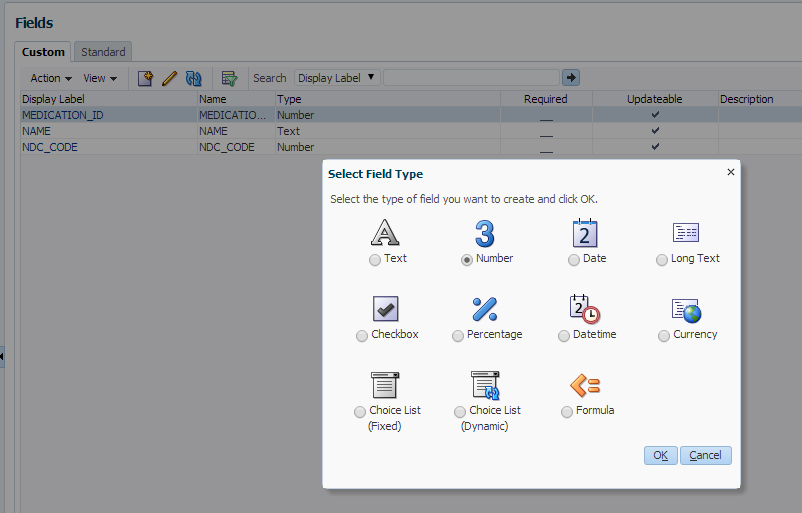
1. Click on Fields on the left side



1. Click ‘+’ icon to create a custom field



1. Choose number



1. Fill up the required field and click save

