# Extensibility Guide
# Oracle Financial Services Lending and Leasing
Release 14.3.0.1.0

Part No. E79024-01
[September] [2016]

# Table of Contents


**1. PREFACE ..... 1-1**

1.1 AUDIENCE ..... 1-1
1.2 CONVENTIONS USED ..... 1-1
1.3 PRE-REQUISITE ..... 1-1

**2. CUSTOMIZING AND EXTENDING THE APPLICATION ..... 2-2**

2.1 UNDERSTANDING CUSTOMIZING & EXTENDING THE APPLICATION ..... 2-2
2.2 UNDERSTANDING CUSTOMIZATION LAYERS ..... 2-2
2.3 INSTALLING CUSTOMIZATION TOOLS ..... 2-4

**3. USING JDEVELOPER FOR CUSTOMIZATIONS ..... 3-1**

3.1 ABOUT USING JDEVELOPER FOR CUSTOMIZATION ..... 3-1
3.2 ABOUT CUSTOMIZING ORACLE ADF ARTIFACTS ..... 3-1
3.3 CUSTOMIZING ORACLE ADF ARTIFACTS WITH JDEVELOPER ..... 3-3

**4. APPLICATION ARTIFACTS ..... 4-1**

4.1 ABOUT CUSTOMIZING ORACLE ADF APPLICATION ARTIFACTS ..... 4-1
4.2 CUSTOMIZABLE APPLICATION LIBRARIES ..... 4-1
4.3 ENABLE JDEVELOPER FOR CUSTOMIZATION ..... 4-2
4.4 CUSTOMIZING THE SKIN ..... 4-2
4.5 CUSTOMIZING OR ADDING RESOURCE BUNDLES ..... 4-3
4.6 EDITING EXISTING BUSINESS COMPONENTS ..... 4-7
4.7 EDITING PAGES ..... 4-12
4.8 EDITING TASK FLOWS ..... 4-12
4.9 CREATING CUSTOM BUSINESS COMPONENTS ..... 4-13
4.10 CREATING CUSTOM TASK FLOWS ..... 4-14
4.11 CREATING CUSTOM PAGES ..... 4-14
4.12 EDITING THE UI SHELL TEMPLATE ..... 4-15
4.13 DEPLOYING ADF CUSTOMIZATIONS AND EXTENSIONS ..... 4-15
4.14 DEPLOYMENT OPTIONS ..... 4-16

**5. CUSTOMIZING DATABASE OBJECTS ..... 5-1**

5.1 UI – PACKAGE INTERACTION LOGIC ..... 5-1
5.2 UI JAVA WRAPPER (U*JW) ..... 5-1
5.3 DATABASE SCHEMA ..... 5-2
5.4 WRAPPER ENGINE MODEL ..... 5-3
5.5 BATCH JOB (BJ) ..... 5-4
5.6 ENGINE WRAPPER (EW) ..... 5-4
5.7 MAIN ENGINE (EM) ..... 5-4
5.8 ENGINE FUNCTION (EN) ..... 5-4
5.9 ENGINE VIEW ..... 5-5
5.10 COMMON FEATURES ..... 5-5
5.11 SEED DATA ..... 5-5
5.12 DEVELOPER'S TIPS ..... 5-5

**6. CREATING NEW CUSTOM BI PUBLISHER REPORT/LETTER ..... 6-1**

6.1 CREATE REPORT LAYOUT ..... 6-8
6.2 CREATE XML DATA ..... 6-10
6.3 ADD DYNAMICS TO REPORT ..... 6-16
6.4 UPLOAD REPORT IN BIP ..... 6-20

**7. CUSTOMIZING EXISTING BASE BIP REPORTS ..... 7-1**

**8. CUSTOMIZING EXISTING BASE BIP LETTERS ..... 8-1**

**9. CREATE CUSTOM CORRESPONDENCE ..... 9-1**

**10. GENERATING CORRESPONDENCE ..... 10-1**

**11. SETTING UP THE OUTPUT FORMAT FOR BIP REPORTS ..... 11-1**

**12. NAMING CONVENTION FOR CUSTOMIZED OBJECTS ..... 12-1**

**13. RESTFUL WEBSERVICES EXTENSIBILITY ..... 13-1**

13.1 GENERIC POST TRANSACTION ..... 13-1

*13.1.1 Producer related transaction ..... 13-1*

*13.1.2 Other Transactions ..... 13-2*

13.2 ACCOUNT ON BOARDING ..... 13-5

# 1. Preface

This document provides an overview on extensibility capabilities supported by Oracle Financial Services Lending and Leasing Application.

## 1.1 Audience

This document is intended for administrators and developers who want to customize and extend the standard functionality provided by Oracle Financial Services Lending and Leasing Application. Administrators should have a basic understanding of Oracle Financial Services Lending and Leasing Application and Oracle Application Development Framework concepts. Developers should have a basic understanding of the Java programming language, web applications, Oracle JDeveloper, and Oracle Application Development Framework.

## 1.2 Conventions Used

| Term | Refers to |
|---|---|
| Application | Oracle Financial Services Lending and Leasing |
| Customization application workspace | *OracleFSLLEnterpriseApp/ OracleFSLLEnterpriseApp.jws* provided as part of installer under */cust_lib* folder |

## 1.3 Pre-requisite

You can find all the customizable libraries along with the necessary default projects as part of the product release installer bundle under /cust_lib folder.

You need to download and install JDeveloper 12.2.1.0.0.

# 2. Customizing and Extending the Application

This chapter provides an overview of how to customize and extend the application and, introduces the design time and runtime tools used in the process, such as Oracle JDeveloper, Oracle Business Intelligence (BI) Publisher and Oracle Enterprise Manager Fusion Middleware Control.

## 2.1 Understanding Customizing & Extending the Application

Oracle Financial Services Lending and Leasing application is based on Oracle Fusion Middleware. User interfaces are implemented using Oracle Application Development Framework (Oracle ADF) and standard Java technologies. Business intelligence frameworks provide a number of reporting capabilities. Each of these areas of the application can be customized and extended to suit your business needs.

Within this guide, the term customizing means to change a standard (existing) artifact. For example, you can add an attribute to a standard business object, or you can change what is displayed on a standard view page. The term extending means to create a completely new artifact, such as a custom business object or custom view page. For customizations and extensions of this application, there are two basic scenarios: personalization and design time customizations and extensions.

**Personalization**

Personalization refers to the changes that every end user of the application can make to certain artifacts in the user interface (UI) at runtime. These changes remain for that user each time that user logs into the application. Personalization includes changes based on user behavior (such as changing the width of a column in a table)

**Design time customizations and extensions**

Design time customizations and extensions include more complex changes, such as creating new business objects or creating new view pages, and they require deployment into the runtime environment. Design time customizations are done by Java developers using Oracle JDeveloper. The customizations are then uploaded or deployed to a running instance of the application.

Most customizations, whether a personalization an end user makes, or a change a developer makes using JDeveloper to create new source code, are stored in a business metadata repository. Because these customizations are kept separate from the base code, you can safely upgrade your application without overwriting or needing to redo your changes.

Customizations for the UI and for entity components are created in layers, meaning that you can create them for specific industry, or for specific region or sites, and the changes will be shown only when applicable. For more information about the metadata dictionary and customization layers, see section 2.2 - 'Understanding Customization Layers'.

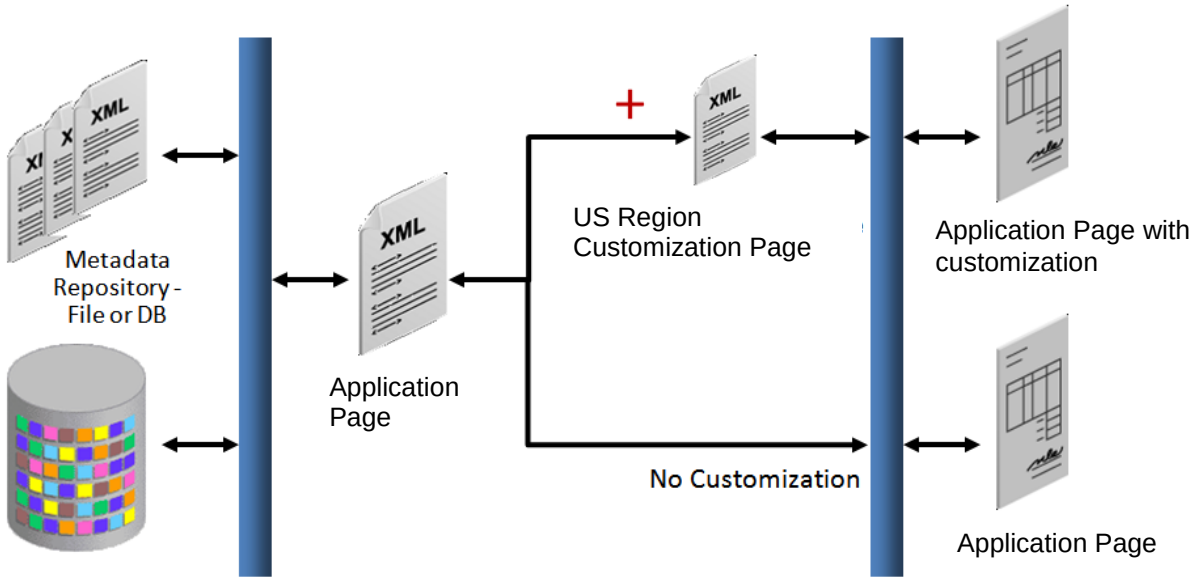
## 2.2 Understanding Customization Layers

The application contains customization layer that allows you to make customizations which affect only certain instances of an application. For example, the application has a layer for US region. When you customize an artifact, you can choose to make that customization available only for US region.

Customizations you make are not saved to the base standard artifact. Instead, they are saved to an XML file that is stored in an Oracle Metadata Services (MDS) repository. This XML file acts like a list of instructions that determines how the artifact looks or behaves in the application, based on the layer that is controlling the current context. The MDS Customization Engine manages this process.

For example, say you want to customize the Applicant fragment by adding a new Passport field, but only for US region. Before you make your customization, you first select the layer to make your customization in, in this case the region layer whose value is US. When you make your customization by adding the new Passport field in the Application fragment, an XML file is generated with the instructions to add the field, but only in the region layer, and only when the value is US. The original page file remains untouched. The MDS Customization Engine then stores the XML file in an MDS repository.

Now, whenever someone logs into the application and requests an artifact, the MDS Customization Engine checks the repository for XML files that match the requested artifact and the given context, and if there is a match, it layers the instructions on top of the base artifact. In this example, whenever the Application page is requested (the artifact) by someone where US region customization is applied, before the page is rendered, the MDS Customization Engine pulls the corresponding XML file from the repository and layers it on top of the standard Application page, thereby adding the new field.

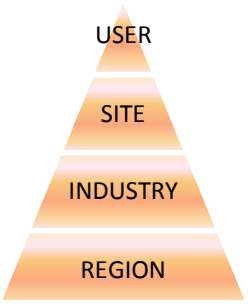



All users of the application can personalize the pages. Users can move elements around on a page, hide elements, and even add available elements to their page. When they do this personalization, the MDS Customization Engine creates an XML file specific to that user.

For example, say User 1 personalizes the Application page. There will then be an XML file stored in the repository, noting the changes that user made. When User 1 logs in, as in the previous example, the MDS Customization Engine pulls the XML file with the customizations from the repository and layers it on top of the standard Application page. In addition, the engine pulls the XML file with User 1's personalization's, allowing the user to see the personalization changes along with the US region changes. When other users log in, they do not see User 1's personalization changes.

The application has following customization layers:

- Region: When customizations are made in this layer, they affect users of the application for a specific region. This layer's XML files are added for everyone, whenever the artifact is requested.
- Industry: When customizations are made in this layer, they affect users of the application for a specific industry. This layer's XML files are added for everyone, whenever the artifact is requested.
- Site: Customizations made in the Site layer affect users at a particular location.
- User: This is where all personalization's are made. Users do not have to explicitly select this layer.




These layers are applied in a hierarchy, and the highest layer in that hierarchy in the current context is considered the tip. Within the default customization layers, the Region layer is the base layer, and the User layer is the tip. If customizations are done to the same object, but in different layers, at runtime, the tip layer customizations take precedence. For example, if you customize the label for a field in the site layer and customize the same label in the industry layer using JDeveloper, the site layer customization will be displayed at runtime.

Because customizations are saved in these XML files, when you patch or upgrade your application, the base artifacts can be updated without touching your changes. The base artifact is replaced, and when the application is run after the patch or upgrade, the XML files are simply layered on top of the new version. You do not need to redo your customizations.

Before you create customizations, you must select the layer to which you want your customizations to be applied.

## 2.3 Installing Customization Tools

For procedures for setting up JDeveloper for customizations, see chapter 3, Using JDeveloper for Customizations.

# 3. Using JDeveloper for Customizations

This chapter describes how to configure JDeveloper for implementing customizations in the application.

## 3.1 About Using JDeveloper for Customization

JDeveloper is used when it is needed to customize or create business objects or new pages. The procedures for each of these are different.

New custom objects created in JDeveloper are not saved into the MDS Repository, and so are done in a standard application workspace using the **Default** role. However, when you customize standard objects, those customizations are saved into the MDS Repository, and so must be done using the **Customization Developer** role. Doing customizations using the customization developer role ensures that the changes are saved to upgrade-safe MDS Repository, and not written directly to the standard object. In future, when patch or upgrade Application, the customizations held in these metadata files will not be touched, and so, it need not be re-done.

When customizing ADF artifacts, a special customization application workspace can be created; using the **Default** role, for this application a default customization application workspace (*/OracleFSLLEnterpriseApp/OracleFSLLEnterpriseApp.jws*) is provided. This workspace includes all the artifacts that can be customized. This customization workspace can be configured, so that when customizations are tested and deployed, they appear to be part of native Oracle Financial Services Lending and Leasing Application.

Using the default workspace, it is possible to switch roles to customization developer and customize the ADF artifacts required. After completion, the artifacts are packaged and deployed in the workspace to the Oracle Financial Services Lending and Leasing environment.

Often, there is a need to perform both customizations (customizing an existing standard object) and extensions (creating a new object). For example, suppose it is needed to create a new business object and expose that new object in an existing application module. First, because a new custom business object is being created, first a standard application workspace is created and then entity object is created. After completion, the workspace is packaged as an ADF Library, and placed into a directory. Next, using the default workspace provided, the new entity object library and the library that contains the application module to which we need to add the entity object is added. After both are imported, User should log in using the customization developer role and make the customizations to the application module. After customizations are complete, User would deploy the customizations to the test environment.

**Note**

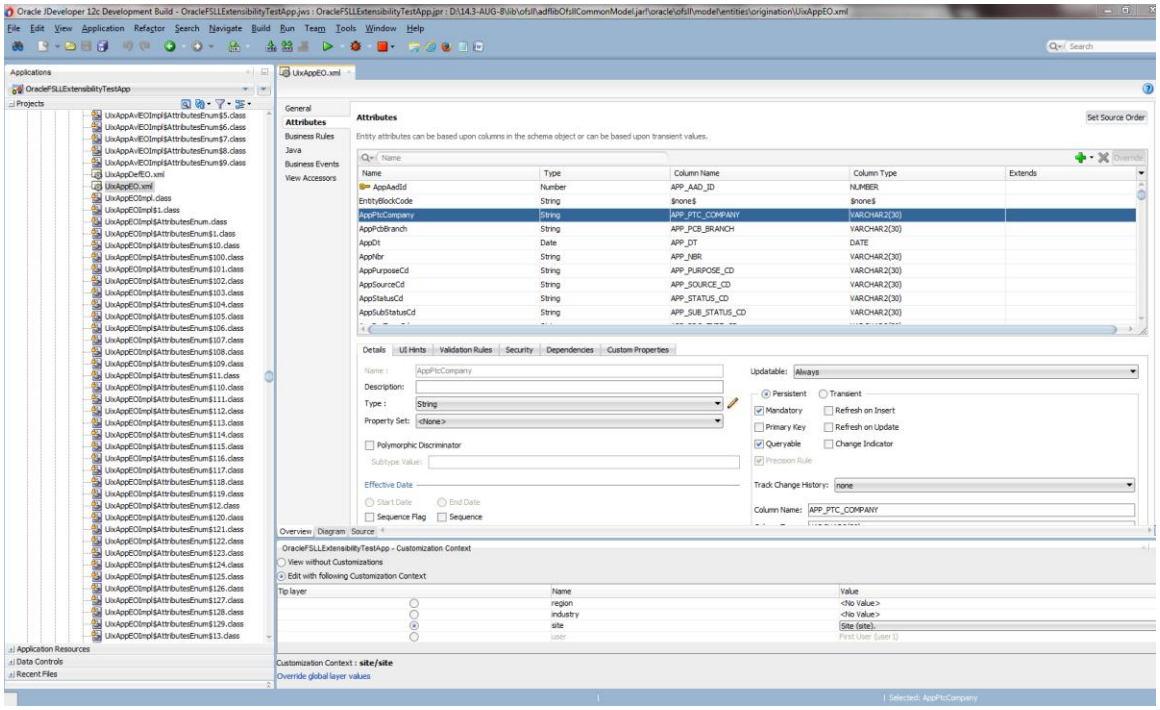
Before running JDeveloper in customization mode for the application, see Section 4.3 "Enable JDeveloper for Customization" for pre-configuration requirement.

## 3.2 About Customizing Oracle ADF Artifacts

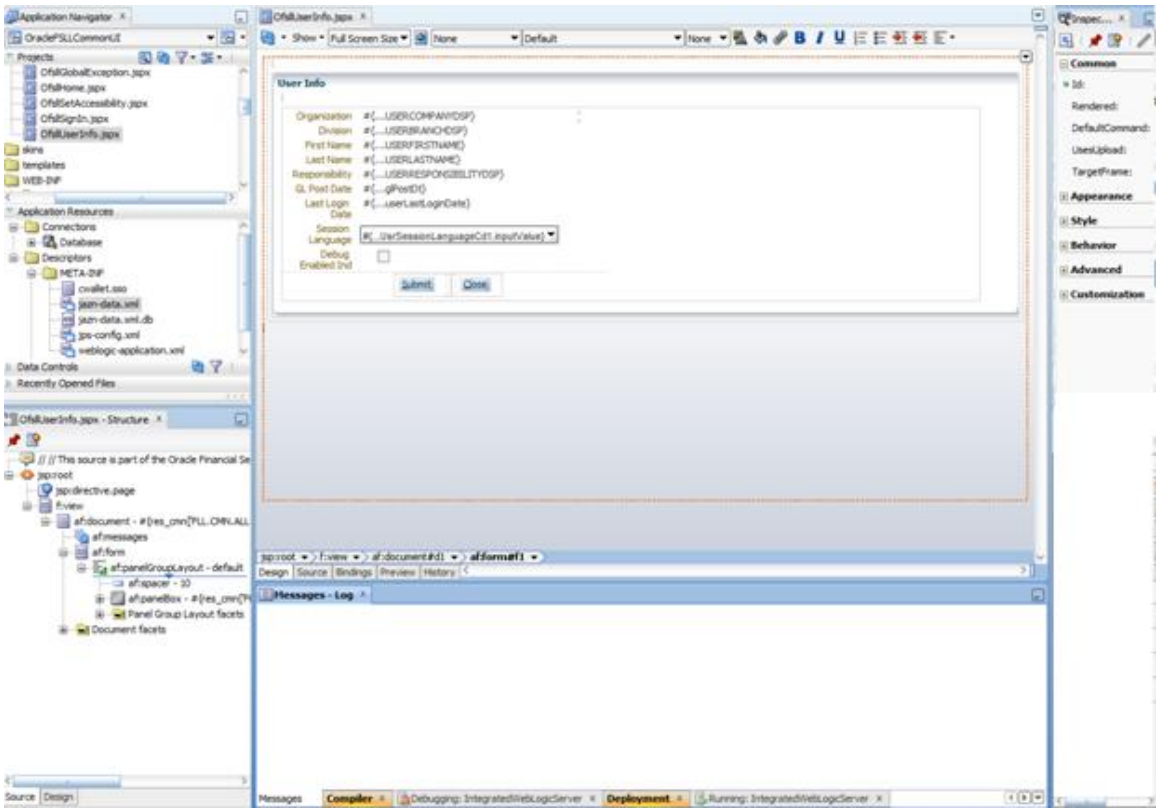
The application is built using Oracle ADF artifacts, including the following:

- Application modules: An application module is the transactional component that UI clients use to work with application data. It defines an updateable data model along with top-level procedures and functions (called service methods) related to a logical unit of work that is related to an end-user task.
- Entity objects: An entity object represents a row in a database table and simplifies modifying its data by handling all data manipulation language (DML) operations. It can encapsulate business logic to ensure that the required business rules are consistently enforced. An entity object can be associated with others to reflect relationships in the underlying database schema, to create a layer of business domain objects, and to reuse in multiple applications.
- View objects: A view object represents a SQL query and simplifies working with its results. The SQL language is used to join, filter, sort, and aggregate data into the shape required by the end-user task to be represented in the user interface. This includes the ability to link a view object with other view objects to create master-detail hierarchies of any complexity. When end users modify data in the user interface, view objects collaborate with entity objects to consistently validate and save the changes.
- Task flows: Task flows define the flow of control throughout an application. They also can be included in a page as a region, where users can navigate through a series of page fragments, without leaving the original page.
- JSPX pages and page fragments: The view layer of the application consists of a small number of pages per application. These pages then contain task flows, which in turn contain a number of page fragments.

When Oracle ADF artifacts are customized, it generally happens in an overview editor that allows making customizations declaratively. For example, below figure shows the editor for an entity object. Among other things, validation can be set or UI display changes can be done.



For JSPX pages, a WYSIWYG environment is displayed where changes can be made using the Design tab in the editor window or structure window.



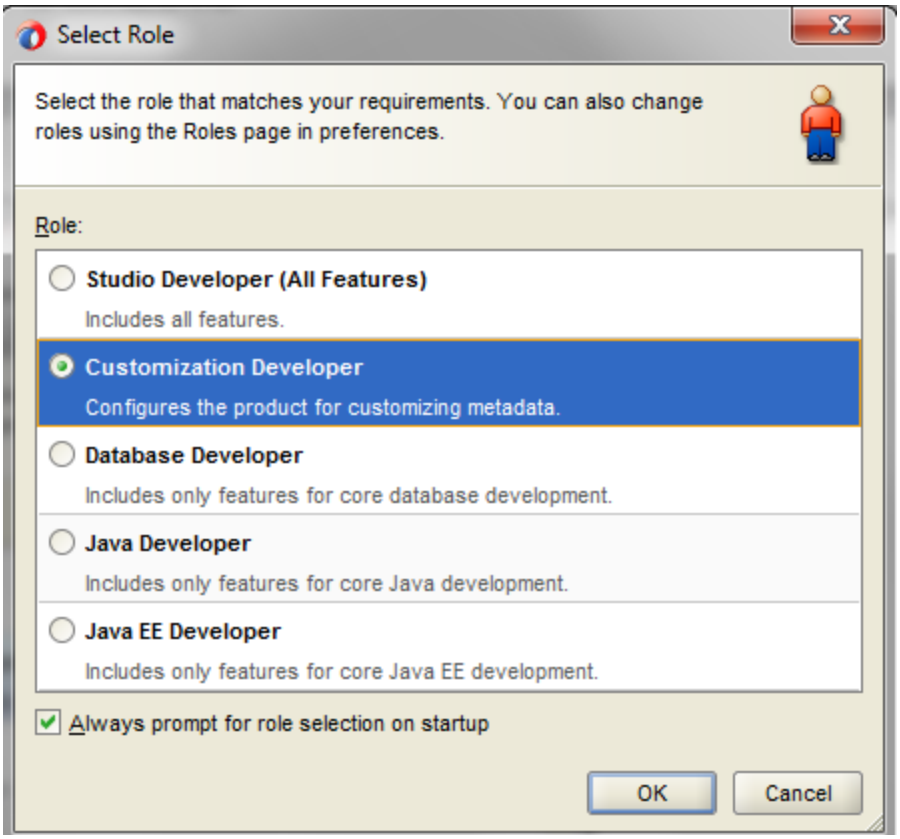

## 3.3 Customizing Oracle ADF Artifacts with JDeveloper

To customize ADF artifacts, open the default customization application workspace provided, using the **Customization Developer** role and customize the required artifacts.

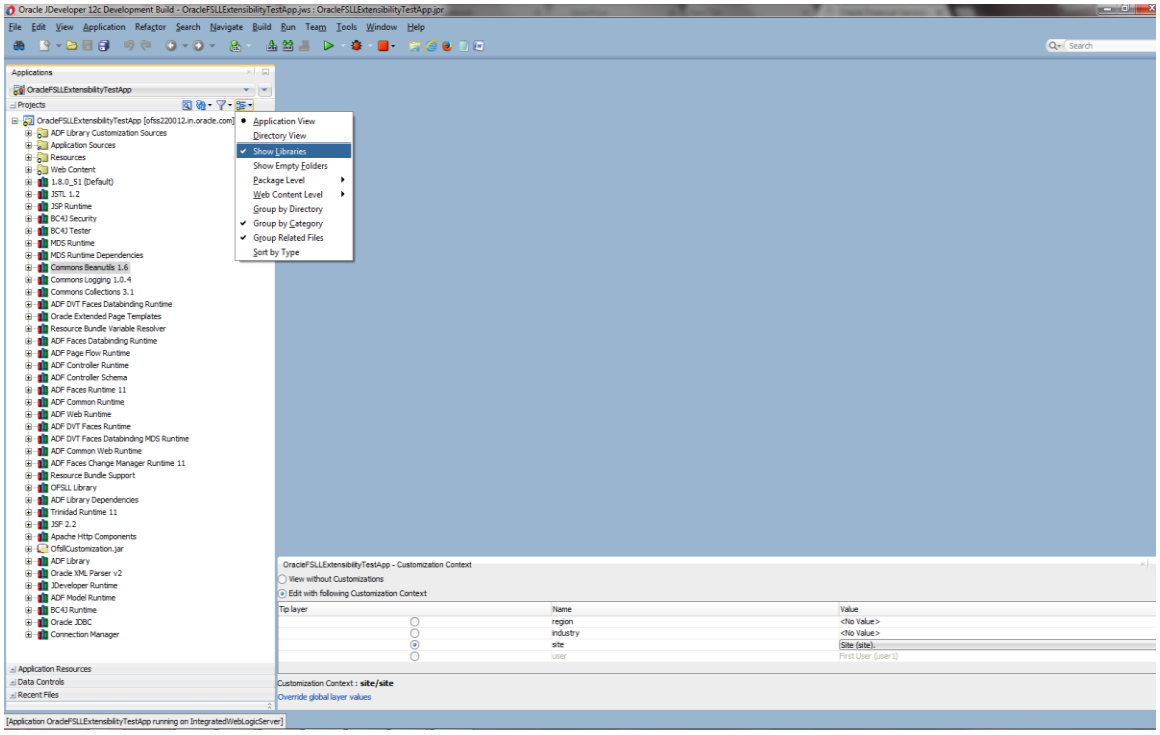
### Customizing the Artifacts

Users need to switch to the **Customization Developer** role before they can begin customizing.

1. Restart JDeveloper and select the **Customization Developer** role.




The artifacts from the imported library are displayed in the Application Navigator pane, under the ADF Library Customizations node, and the artifact selected to customize opens in the editor window.



2. In the Customization Context window (by default, displayed at the bottom of JDeveloper), select the layer that you want the customizations written to.

Note the following:

In case you want to change the value from customization.properties, you can follow the below steps

- Step 1: Extract the OfsllCustomization.jar using the following command.
  Jar –xvf OfsllCustomization.jar
- Step 2: Modify the value in customization.properties
- Step 3: Remove the old OfsllCustomization.jar, to build the jar again, please issue the following command
  Jar -cvf **OfsllCustomization.jar** customization.properties oracle META-INF adf-loc.jar

# 4. Application Artifacts

This chapter describes how to use Oracle JDeveloper to customize and extend application artifacts defined by Oracle Application Development Framework (Oracle ADF) in Oracle Financial Services Lending and Leasing Application.

## 4.1 About Customizing Oracle ADF Application Artifacts

With the customization features provided by Oracle Metadata Services (MDS), developers can customize the application using JDeveloper, making modifications to suit the needs of a particular group, such as a specific region or industry or site.

Using JDeveloper, you can implement customizations on existing artifacts that are provided. The application can also be extended with new custom artifacts that are packaged into a JAR file, and integrated using customizations on the existing application.

However customizations to the application require a lower level approach, for which JDeveloper needs to be used.

## 4.2 Customizable Application Libraries

All customization in the application would be done on the ADF Libraries. List of libraries that can be customized and set of default projects that can be used for building the projects are:

| Library Name | Description |
|---|---|
| adflibOfsllCommonModel.jar | Contains all the application Business Objects such as entity object, view object and application module. |
| adflibOfsllCommonUI.jar | Contains all the User Interface fragments (JSFF) and taskflows (TFs) and all re-useable templates. |

| Project Name | Description |
|---|---|
| OracleFSLLEnterpriseApp/ OracleFSLLEnterpriseApp.jws | Enterprise EAR Application deployment project. This is the default customization main project used to bundle all the libraries into an EAR. |
| OracleFSLLCommonSkin/ OracleFSLLCommonSkin.jws | Application Skin project, containing images and CSS file. The skin project changes can be handled through Oracle ADF Skin Editor. |
| OracleFSLLCustomization/ OracleFSLLCustomization.jws | Customization project containing the customization layer values i.e. region layer, Industry layer and site layer key value pair. |

**Note**

- Above projects are available as part of the application installer bundle under */cust_lib* folder.

- The customizable libraries can be extracted out of /core_as/*.ear file. Extract the *.war out of *.ear and the libraries are under /WEB-INF/lib folder.

- Currently existing menu items cannot be customized as well as new menu items cannot be added.

## 4.3 Enable JDeveloper for Customization

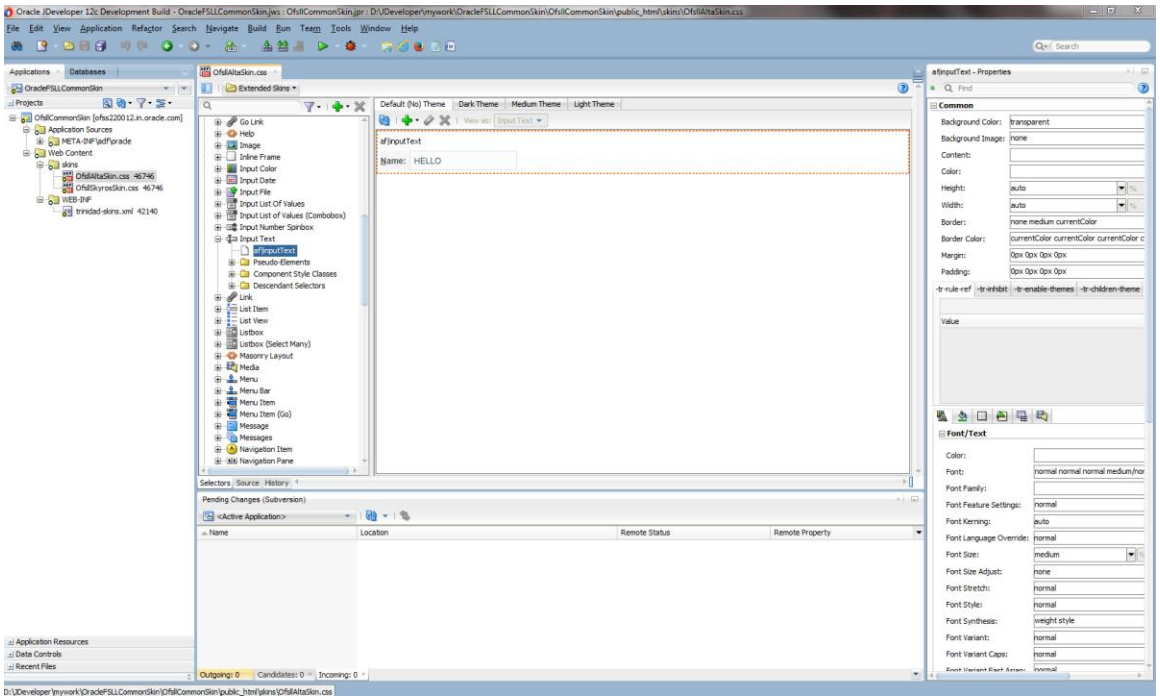
Before running the JDeveloper in Customization Developer role, JDeveloper needs to be configured with following settings:

3. Open JDeveloper in Default role and open the OracleFSLLCustomization/ OracleFSLLCustomization.jws project and edit the customization.properties file with appropriate values for Region key layer, Industry key layer and Site key layer.
4. Rebuild the OfsllCustomization.jar using the default deployment profile.
5. Edit the /cust_lib/CustomizationLayerValues.xml in Notepad and update the Region key layer, Industry key layer and Site key layer with the values added as per required customization.properties.
6. Copy the CustomizationLayerValues.xml onto JDeveloper installation location under $JDEV_HOME/jdeveloper/jdev.

## 4.4 Customizing the Skin

One method of customizing skin is opening the bundled *OracleFSLLCommonSkin/OracleFSLL CommonSkin.jws* project in Oracle ADF Skin Editor Application and customizes the skin details. Once the skin details are customized the same can be bundled as ADF library and deployed to the application server.

1. Open the OracleFSLLCommonSkin Project in Oracle ADF Skin Editor application.
2. Select the component through selectors structure which needs to be customized.
3. Go to Property Inspector and make necessary changes.



4. Make *ADF Library JAR* through deployment profile defined with this project.
5. Copy the JAR into *OracleFSLLEnterpriseApp* to build the EAR.

---

**Note**

Skin can be customized using Oracle ADF Skin Editor which can be downloaded from Oracle site. If the default skin family name is changed then *trinidad-config.xml* available in *OracleFSLLEnterpriseApp* needs to be changed with new skin family name.

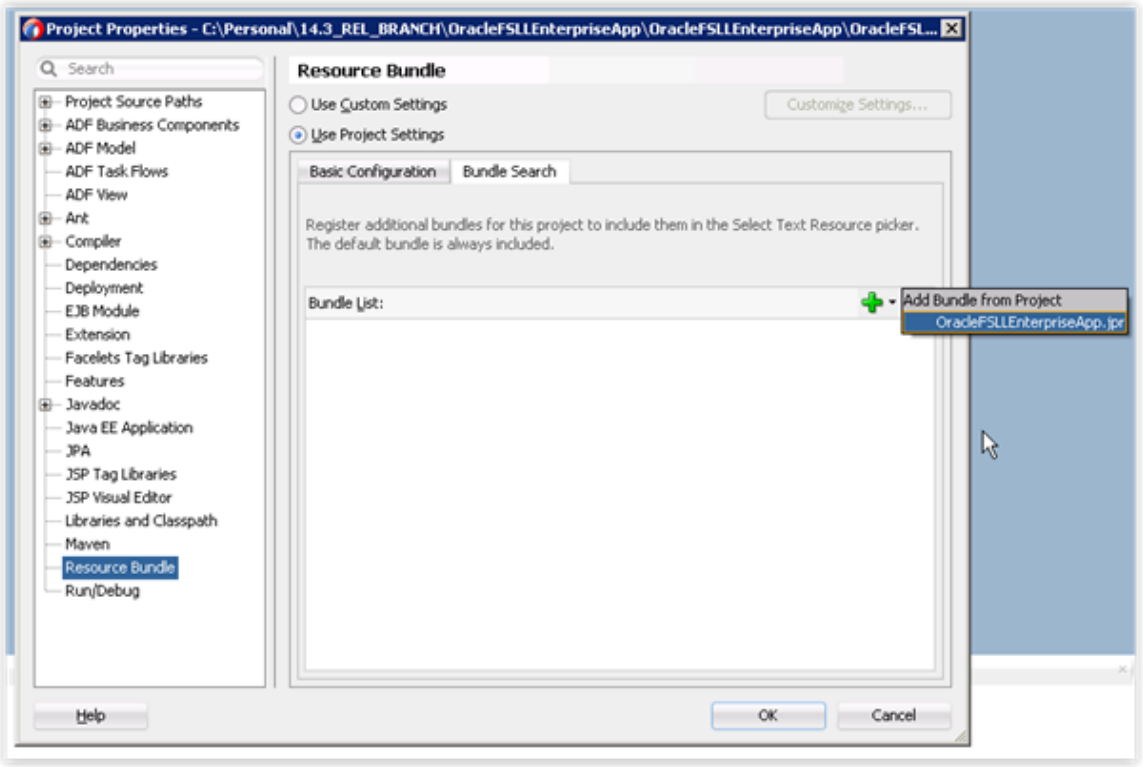
---

## 4.5 Customizing or Adding Resource Bundles

One method of customizing label is by overriding values for existing keys defined in the resource bundle, but new keys cannot be added.

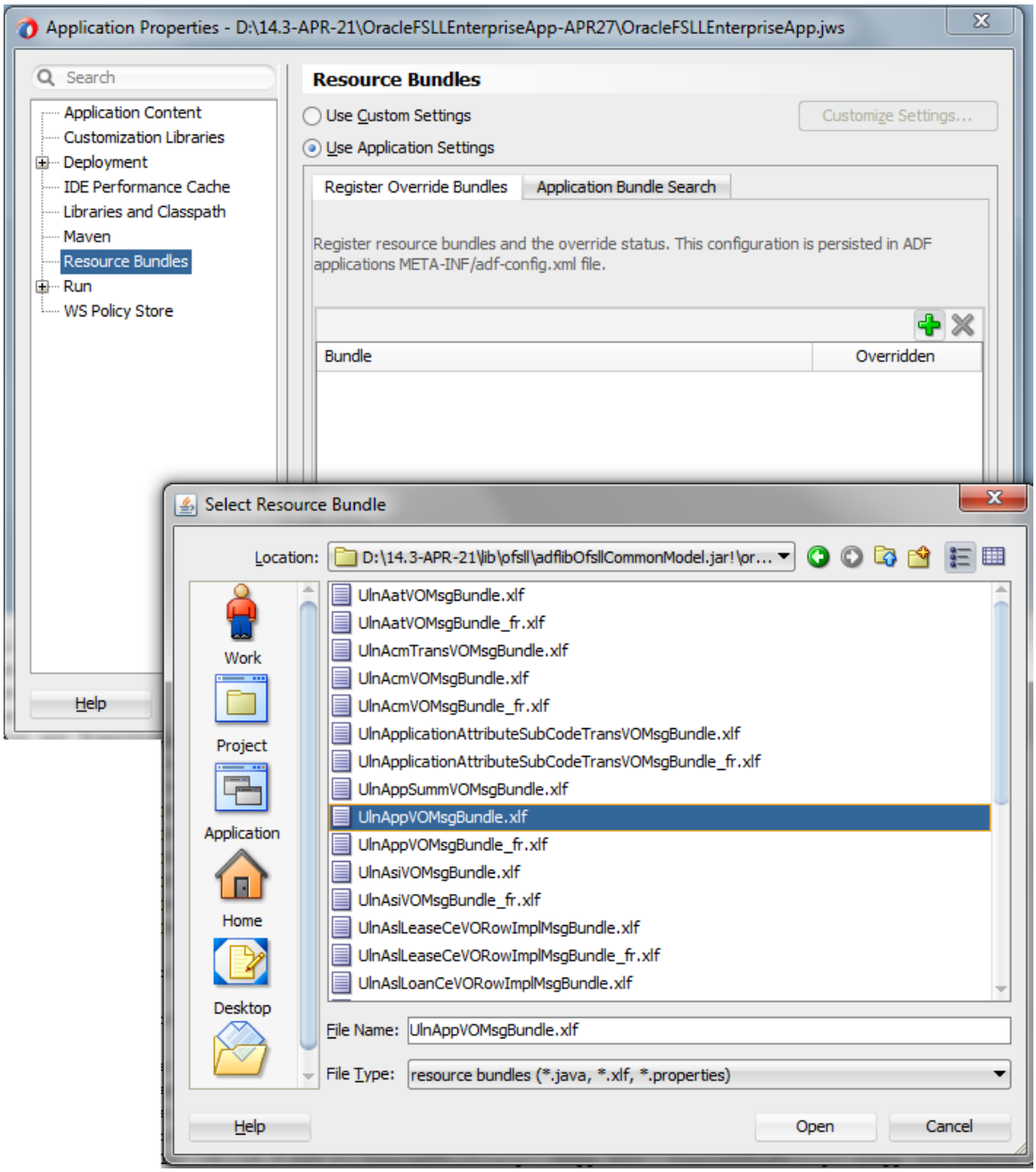
Because a new key cannot be defined in the shipped resource bundle, a new override bundle needs to be created. This can be accomplished in JDeveloper by creating an XLIFF file from the New Gallery. After the file is generated, new keys and their associated text in the XLIFF file can be entered.

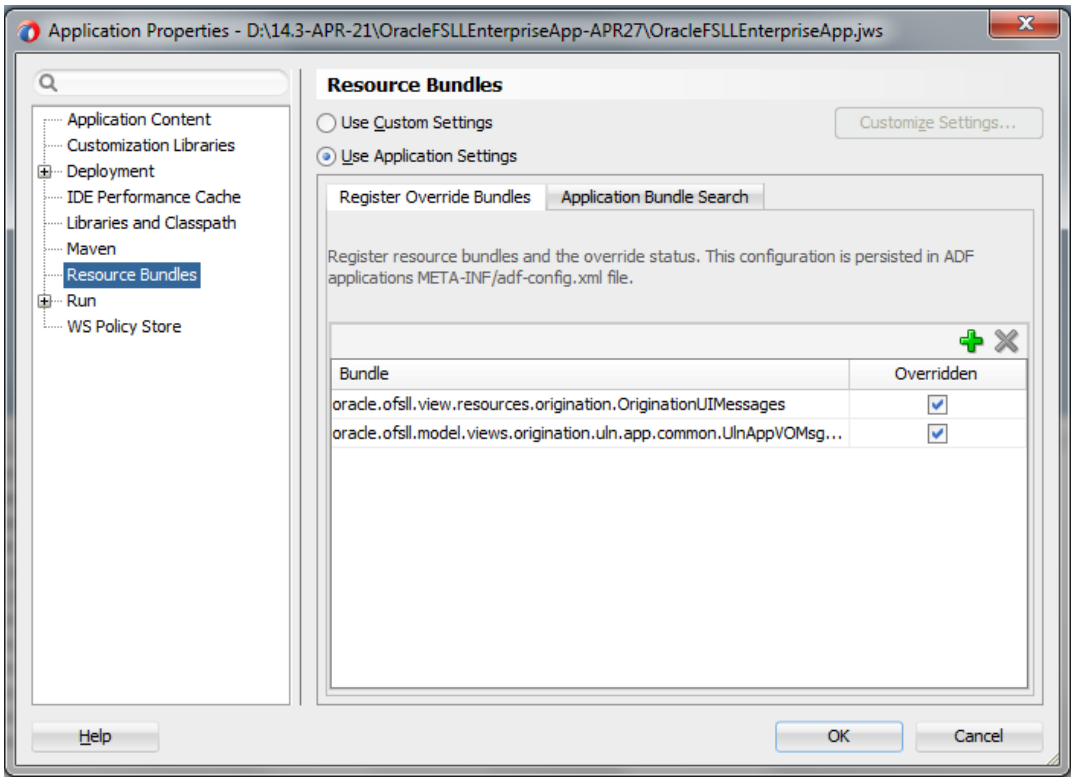
To make the newly created resource bundle available for customization, the resource bundle needs to be registered with the customization project. Newly created resource bundle can be present in customization project or as a separate project. To register the resource bundle with customization project, package it into an ADF Library JAR file, and import the JAR file into the customization project.



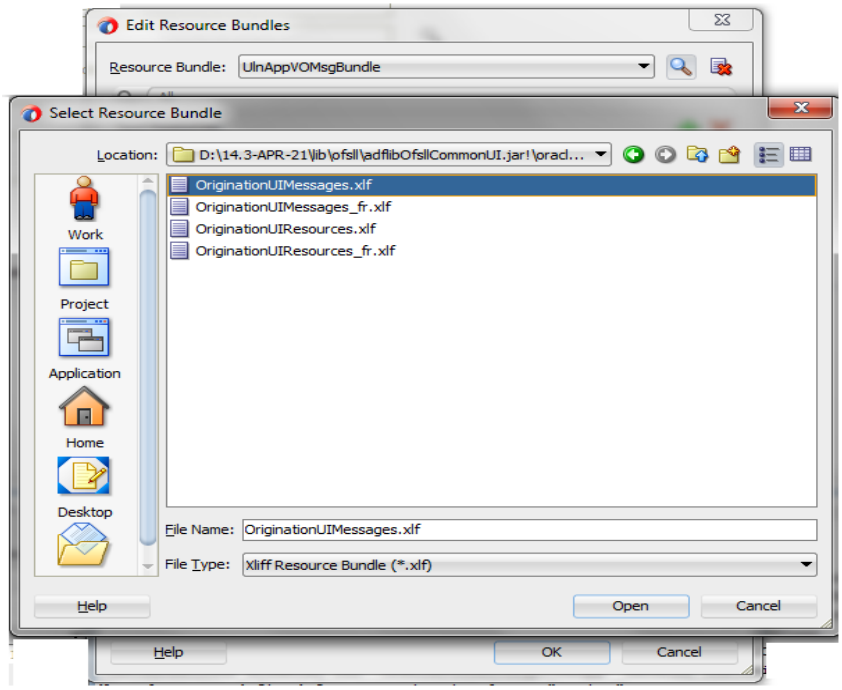

Step to override a message bundle which already exists in model or ui jar, is shown below:

1. Open the JDeveloper in Default role, select the application, click on Application (Menu)→ Application Properties. Select Resource Bundles, add the resource bundles here from jars present under lib folder and check “Overridden” check box. This will register the all selected bundles with adf-config.xml file.

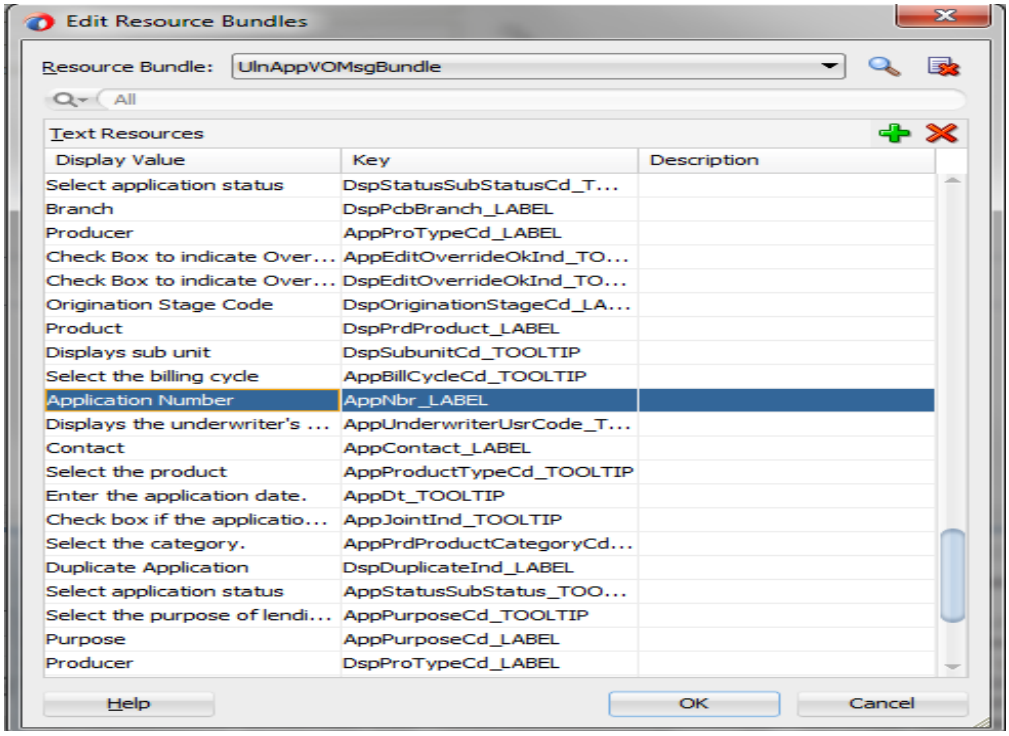

Application Properties - D:\14.3-APR-21\OracleFSLLEnterpriseApp-APR27\OracleFSLLEnterpriseApp.jws
Search
Application Content
Customization Libraries
Deployment
IDE Performance Cache
Libraries and Classpath
Maven
Resource Bundles
Run
WS Policy Store
Help
Resource Bundles
Use Custom Settings
Customize Settings...
Use Application Settings
Register Override Bundles
Application Bundle Search
Register resource bundles and the override status. This configuration is persisted in ADF applications META-INF/adf-config.xml file.
Bundle
Overridden
Select Resource Bundle
Location:
D:\14.3-APR-21\lib\ofsll\adflibOfsllCommonModel.jar!\or...
Work
Project
Application
Home
Desktop
UlnAatVOMsgBundle.xlf
UlnAatVOMsgBundle_fr.xlf
UlnAcmTransVOMsgBundle.xlf
UlnAcmVOMsgBundle.xlf
UlnAcmVOMsgBundle_fr.xlf
UlnApplicationAttributeSubCodeTransVOMsgBundle.xlf
UlnApplicationAttributeSubCodeTransVOMsgBundle_fr.xlf
UlnAppSummVOMsgBundle.xlf
UlnAppVOMsgBundle.xlf
UlnAppVOMsgBundle_fr.xlf
UlnAsiVOMsgBundle.xlf
UlnAsiVOMsgBundle_fr.xlf
UlnAslLeaseCeVORowImplMsgBundle.xlf
UlnAslLeaseCeVORowImplMsgBundle_fr.xlf
UlnAslLoanCeVORowImplMsgBundle.xlf
File Name:
UlnAppVOMsgBundle.xlf
File Type:
resource bundles (*.java, *.xlf, *.properties)
Help
Open
Cancel



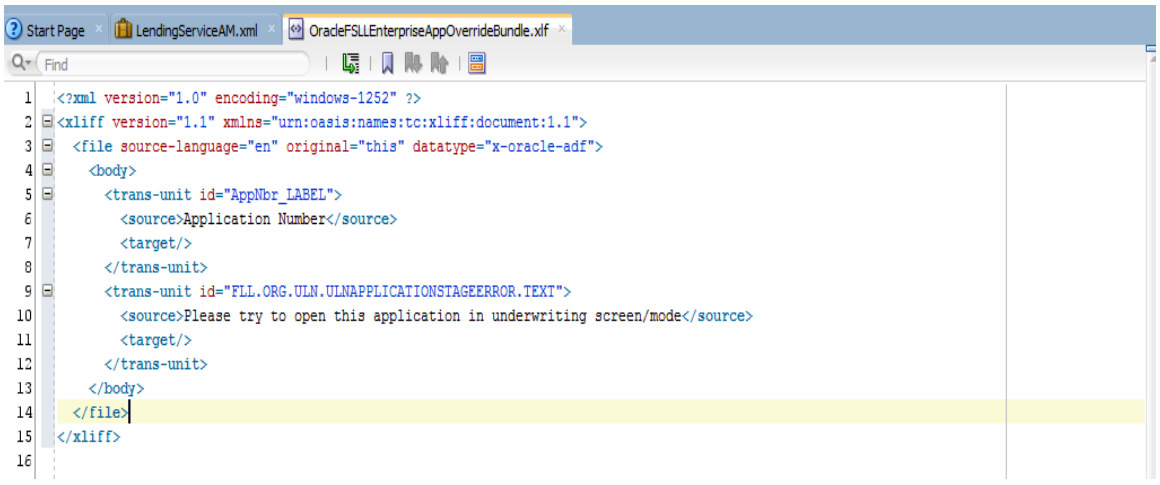

2. The Edit or Override Resource Bundles, Go to Customization Role, Select “Edit Resource Bundle” present under Application Menu. Navigate to the XLIFF Bundle that needs to overridden from the jar under lib folder.




3. Change Display Value for the Key as per requirement.




4. Once changes are submitted, the override resource bundle folder would be created with overridden values.




## 4.6 Editing Existing Business Components

Before you start customizing business objects, it has to be determined which business objects need customizing. Then when customizing ADF artifacts, JDeveloper has to be launched in the **Customization Developer** role, and the appropriate layer selected.

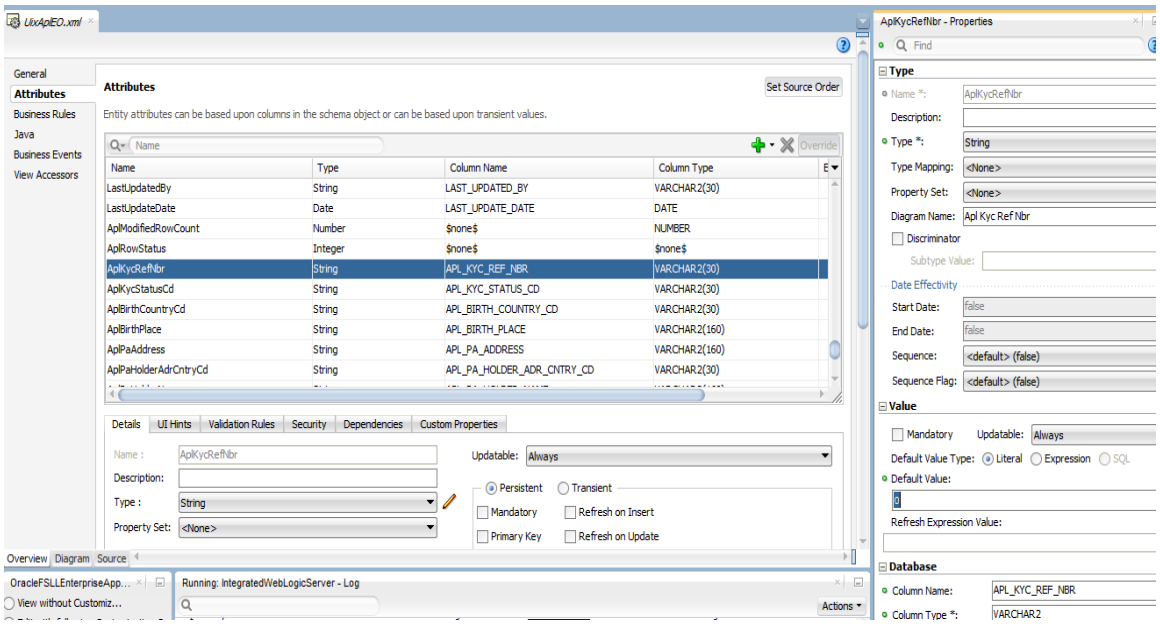
**Task: Edit Attributes**

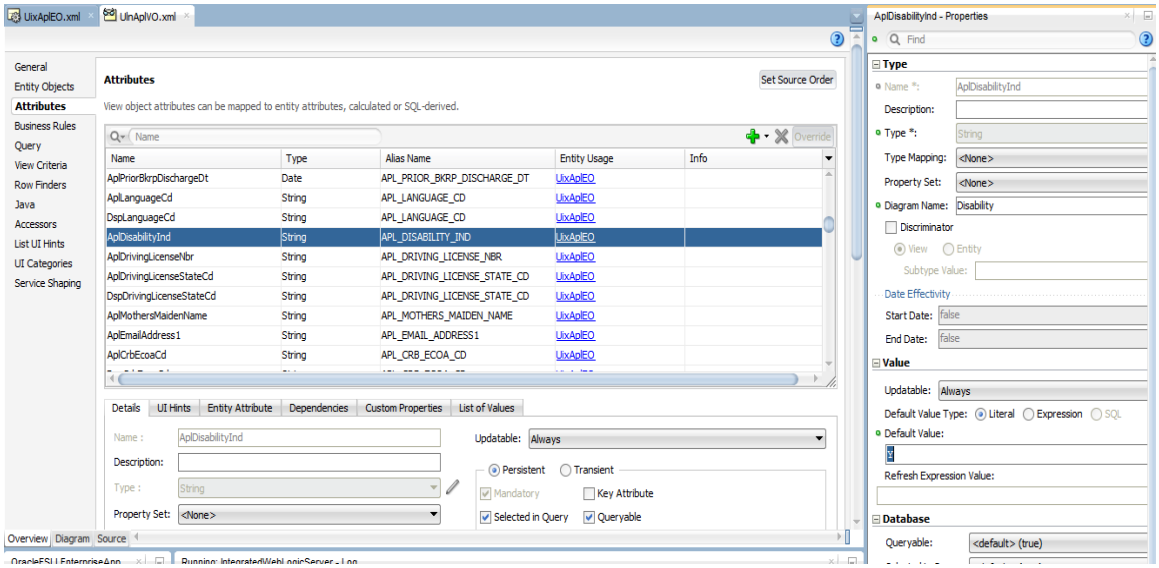
The properties of an attribute can be customized from an entity object or view object using JDeveloper. When an entity object opened or viewed in the overview editor, the attributes of the object can be seen on click in the Attributes tab. When an attribute is selected, its properties are displayed in the Property Inspector.

It is not necessary to modify the page after customizing the properties of an existing attribute. Customizations to existing attributes are automatically reflected on the pages that show them.

However, if an attribute is modified so that it requires a different UI component, it must also be updated in the page. For example, if a list of values (LOV) is added to an attribute, the page needs to be edited to hide the existing UI component that displays the attribute, and a new UI component added that can display the LOV.

Note that some attribute properties defined in the entity object can be overridden in the view object. For example, the label text for a field can be defined in an entity object and subsequently given a different label in the consuming view object. Then pages that use the view object display the label from the view object.



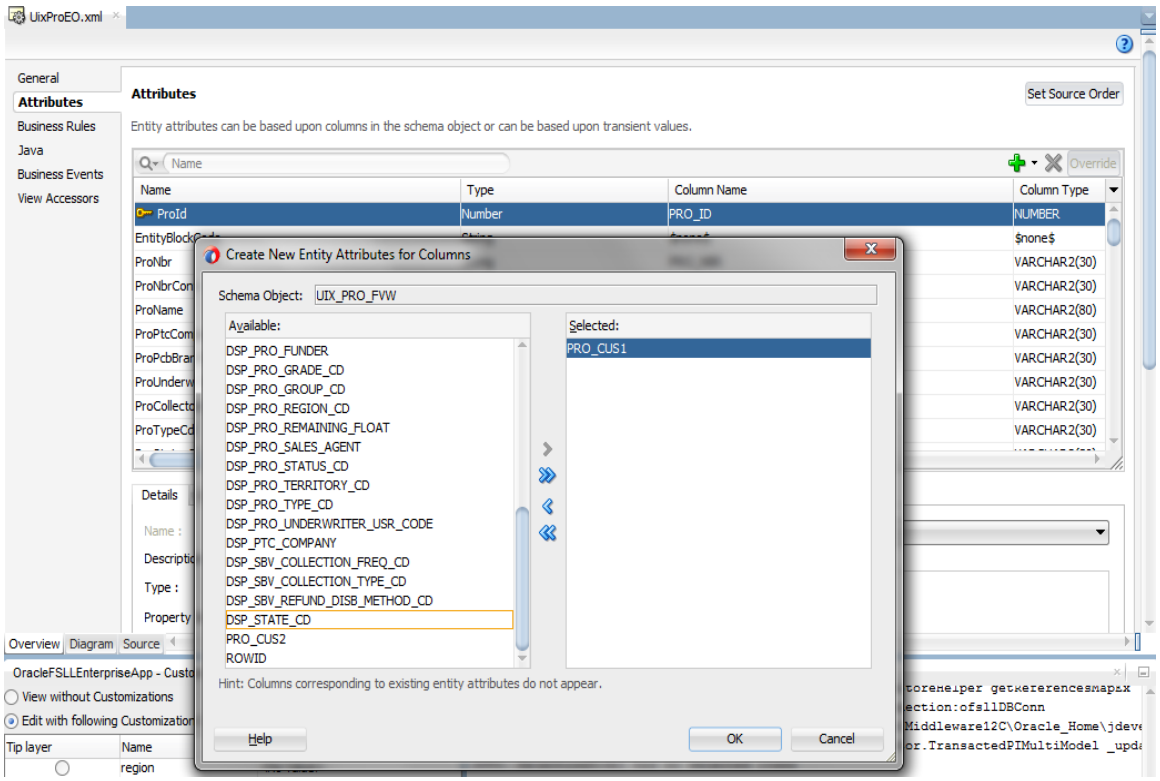


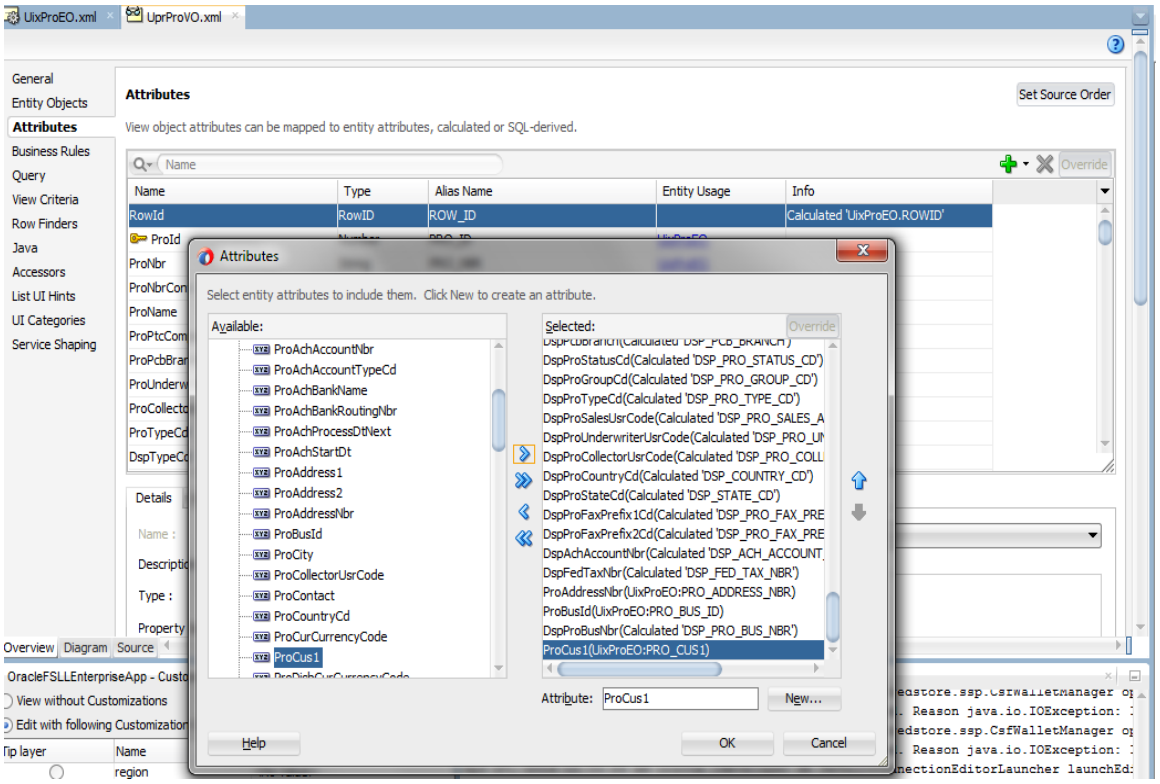

## Task: Add Attributes

Custom attributes can be added to an entity object or view object using JDeveloper. To do this, JDeveloper must be launched in the **Customization Developer** role, a layer selected. Open an entity object or view object in the overview editor, and click the Attributes tab to see the attributes of the object. To add a custom attribute, click the Add icon.

To store the custom attribute in the database, first create the column that will be used to store it.

To display the custom attributes in the application, the pages also needs to be customized to display them.


UixProEO.xml
General
Attributes
Business Rules
Java
Business Events
View Accessors
Attributes
Set Source Order
Entity attributes can be based upon columns in the schema object or can be based upon transient values.
Name
Override
Name
Type
Column Name
Column Type
ProId
Number
PRO_ID
NUMBER
$none$
VARCHAR2(30)
VARCHAR2(80)
Details
Name :
Type :
Property
Overview
Diagram
Source
Create New Entity Attributes for Columns
Schema Object:
UIX_PRO_FVW
Available:
Selected:
PRO_CUS1
DSP_PRO_FUNDER
DSP_PRO_GRADE_CD
DSP_PRO_GROUP_CD
DSP_PRO_REGION_CD
DSP_PRO_REMAINING_FLOAT
DSP_PRO_SALES_AGENT
DSP_PRO_STATUS_CD
DSP_PRO_TERRITORY_CD
DSP_PRO_TYPE_CD
DSP_PRO_UNDERWRITER_USR_CODE
DSP_PTC_COMPANY
DSP_SBV_COLLECTION_FREQ_CD
DSP_SBV_COLLECTION_TYPE_CD
DSP_SBV_REFUND_DISB_METHOD_CD
DSP_STATE_CD
PRO_CUS2
ROWID
Hint: Columns corresponding to existing entity attributes do not appear.
Help
OK
Cancel
OracleFSLLEnterpriseApp - Custo
View without Customizations
Edit with following Customizatio
Tip layer
Name
region


UixProEO.xml
UprProVO.xml
General
Entity Objects
Attributes
Business Rules
Query
View Criteria
Row Finders
Java
Accessors
List UI Hints
UI Categories
Service Shaping
Attributes
Set Source Order
View object attributes can be mapped to entity attributes, calculated or SQL-derived.
Name
Override
Name
Type
Alias Name
Entity Usage
Info
RowId
RowID
ROW_ID
Calculated 'UixProEO.ROWID'
ProId
Details
Name :
Type :
Property
Overview
Diagram
Source
Attributes
Select entity attributes to include them. Click New to create an attribute.
Available:
Selected:
Override
ProAchAccountNbr
ProAchAccountTypeCd
ProAchBankName
ProAchBankRoutingNbr
ProAchProcessDtNext
ProAchStartDt
ProAddress1
ProAddress2
ProAddressNbr
ProBusId
ProCity
ProCollectorUsrCode
ProContact
ProCountryCd
ProCurCurrencyCode
ProCus1
DspProStatusCd(Calculated 'DSP_PRO_STATUS_CD')
DspProGroupCd(Calculated 'DSP_PRO_GROUP_CD')
DspProTypeCd(Calculated 'DSP_PRO_TYPE_CD')
DspProSalesUsrCode(Calculated 'DSP_PRO_SALES_A
DspProUnderwriterUsrCode(Calculated 'DSP_PRO_UI
DspProCollectorUsrCode(Calculated 'DSP_PRO_COLL
DspProCountryCd(Calculated 'DSP_COUNTRY_CD')
DspProStateCd(Calculated 'DSP_STATE_CD')
DspProFaxPrefix1Cd(Calculated 'DSP_PRO_FAX_PRE
DspProFaxPrefix2Cd(Calculated 'DSP_PRO_FAX_PRE
DspAchAccountNbr(Calculated 'DSP_ACH_ACCOUNT
DspFedTaxNbr(Calculated 'DSP_FED_TAX_NBR')
ProAddressNbr(UixProEO:PRO_ADDRESS_NBR)
ProBusId(UixProEO:PRO_BUS_ID)
DspProBusNbr(Calculated 'DSP_PRO_BUS_NBR')
ProCus1(UixProEO:PRO_CUS1)
Attribute:
ProCus1
New...
Help
OK
Cancel
OracleFSLLEnterpriseApp - Custo
View without Customizations
Edit with following Customizatio
Tip layer
Name
region

### Task: Edit Entity Objects

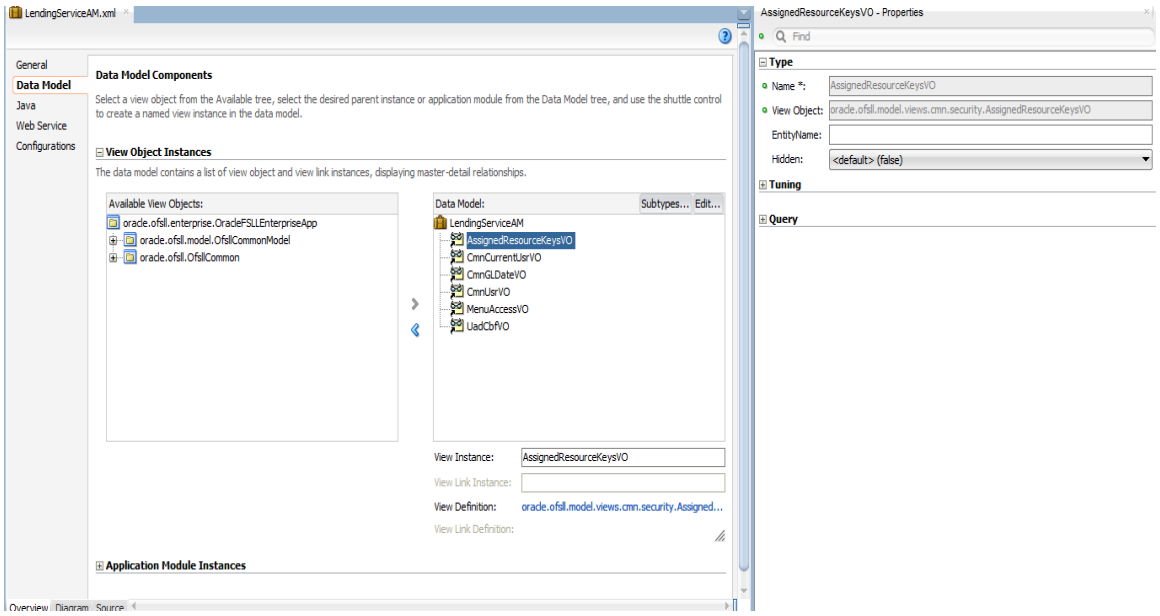
In JDeveloper, edit entity objects using the overview editor. In the Application Navigator, right-click an entity object, and choose **Open**. Then click on the navigator tabs to view and edit the various features of the entity object.

### Task: Edit View Objects

In JDeveloper, edit view objects using the overview editor. In the Application Navigator, right-click a view object, and choose **Open**. Then click on the navigator tabs to view and edit the various features of the view object

### Task: Edit Application Modules

In JDeveloper, edit application modules using the overview editor. In the Application Navigator, right-click an application module, and choose **Open**.




In JDeveloper, the following kinds of customizations can be made on an application module:

- Add new custom properties. This is done on the General page of the overview editor.
- Add new view object and application module instances. This is done on the Data Model page of the overview editor.
- Add newly created subtype view objects. This is done on the Data Model page of the overview editor.
- Add new application module configurations. This is done on the Configurations page of the overview editor.

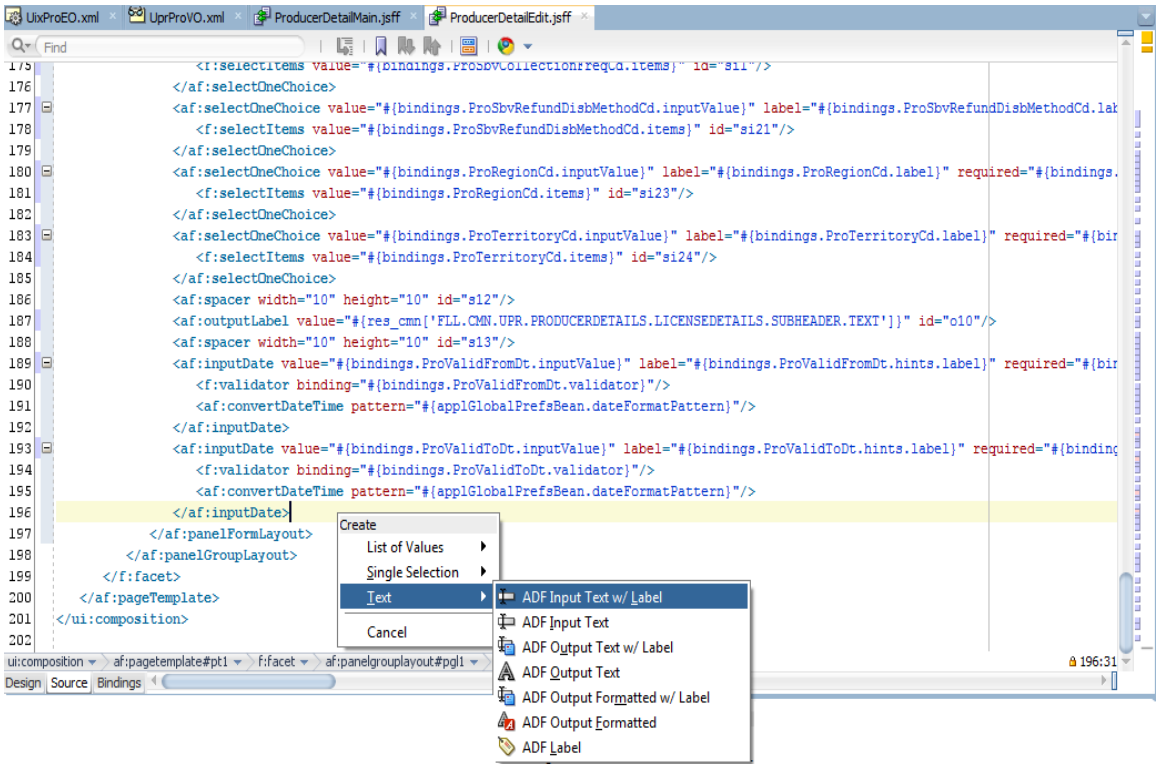
Once the changes are applied, the MDS file is created based on the customization layer value. shown below.

## 4.7 Editing Pages

JDeveloper can be used to implement customizations on the pages that are used in the application. When editing a page in JDeveloper, JDeveloper must be launched in the Customization role.

**Task: Edit Pages**

In the Application Navigator, right-click the page that has to be customized, and choose **Open**. Either new components can be added or existing components properties can be changed via property inspector.



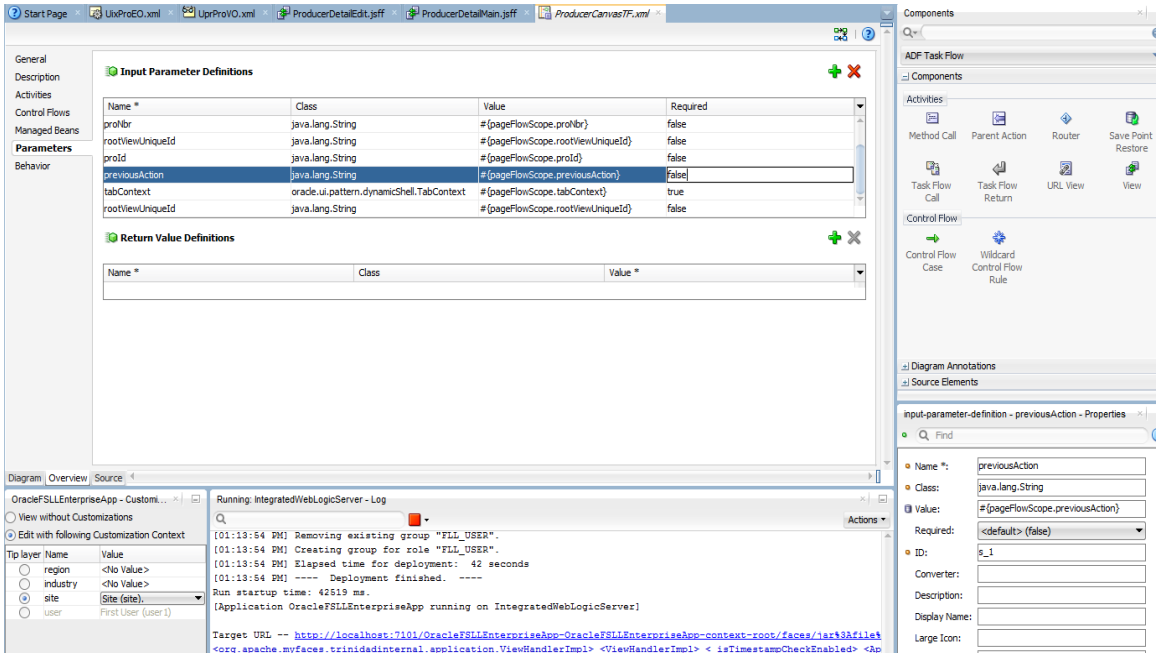

## 4.8 Editing Task Flows

JDeveloper can be used to implement customizations on the task flows that are used in the application. A task flow is a set of ADF Controller activities, control flow rules, and managed beans that interact to allow a user to complete a task. Although conceptually similar, a task flow is not the same as a human task, a task in the worklist, or a process flow.

A bounded task flow can be rendered in a JSF page or page fragment (.jsff) by using an ADF region. This is typically done to allow reuse of the task flow, as necessary, throughout the application. If a bounded task flow is modified, the changes apply to any ADF region that uses the task flow.

### Task: Edit Task Flows

In JDeveloper, the task flow diagram editor is used to implement customizations on existing task flows. In the Application Navigator, right-click the task flow that has to be customized and choose **Open**. The page is displayed in the diagram editor, where changes can be made to the existing activities and control flow cases, or create new custom ones. And in the Overview editor also changes can be made.




## 4.9 Creating Custom Business Components

JDeveloper can be used to extend the application by creating custom business components. When creating custom business components in JDeveloper, JDeveloper must be launched in the **Default** role. This role is used for creating new custom objects that needs to be added to the application. The same workspace that was created for customization can be used. After the custom business components are created, switch to the **Customization Developer** role, to make changes to existing artifacts to integrate the new custom artifacts into the application.

### Task: Create Custom Entity Objects

An entity object represents a row in a database table, and encapsulates the business logic and database storage details of business entities.

In JDeveloper, entity objects can be created using the Create Entity Object wizard, which can be launched from the New Gallery. In the Application Navigator, right-click the project that has to be added to the entity object, and choose New. Then in the New Gallery, expand Business Tier, click ADF Business Components, choose Entity Object, and click OK. Follow the prompts in the wizard to create an entity object.

### Task: Create Custom View Objects

A view object represents a SQL query and also collaborates with entity objects to consistently validate and save the changes when end users modify data in the UI.

In JDeveloper, view objects can be created using the Create View Object wizard, which can be launched from the New Gallery. In the Application Navigator, right-click the project that has to be added to the view object, and choose New. Then in the New Gallery, expand Business Tier, click ADF Business Components, choose View Object, and click OK. Follow the prompts in the wizard to create a view object.

**Task: Create Custom Application Modules**

An application module encapsulates an active data model and the business functions for a logical unit of work related to an end-user task.

In JDeveloper, application modules can be created using the Create Application Module wizard, which can be launched from the New Gallery. In the Application Navigator, right-click the project that has to be added to the application module, and choose New. Then in the New Gallery, expand Business Tier, click ADF Business Components, choose Application Module, and click OK. Follow the prompts in the wizard to create an application module.

**Task: Add Validation**

In JDeveloper, declarative validation rules can be created for entity objects and view objects to help ensure the integrity of the data. To do this, open the entity object or view object in the overview editor, and click the Business Rules navigation tab. Then select the attribute for which validation needs to be provided, click the Create new validator icon, and use the Add Validation Rule dialog to configure the rule.

## 4.10 Creating Custom Task Flows

JDeveloper can be used to create custom task flows that can be included in the application. A task flow is a set of ADF Controller activities, control flow rules, and managed beans that interact to allow a user to complete a task. Although conceptually similar, a task flow is not the same as a human task, a task in the worklist, or a process flow.

**Task: Create a Custom Task Flow**

A custom task flow can be created in JDeveloper using the New Gallery, and then its activities defined using the task flow diagram editor. In the Application Navigator, right-click the project that has to be added to the task flow, and choose **New**. Then in the New Gallery, expand Web Tier, and click JSF/Facelets. Then select ADF Task Flow, and click OK. In the Create Task Flow dialog, specify the details about the type of task flow that needs to be created. Click **OK** and the task flow is created and displayed in the diagram editor.

## 4.11 Creating Custom Pages

JDeveloper can be used to create custom pages that can be included in the application. When creating custom pages in JDeveloper, JDeveloper must be launched in the **Default** role.

When creating the page (or dropping a view activity onto a task flow), it can be created either as a JSF JSP or as a JSF JSP fragment. JSF fragments provide a simple way to create reusable page content in a project, and are used for task flows as regions on a page. When a JSF page fragment is modified, the JSF pages that consume the page fragment are automatically updated.

After extending the application with custom pages, it is required to make sure that security for the new pages are implemented appropriately and that the new pages are deployed so that they are accessible from the application.

**Task: Create a Custom Page**

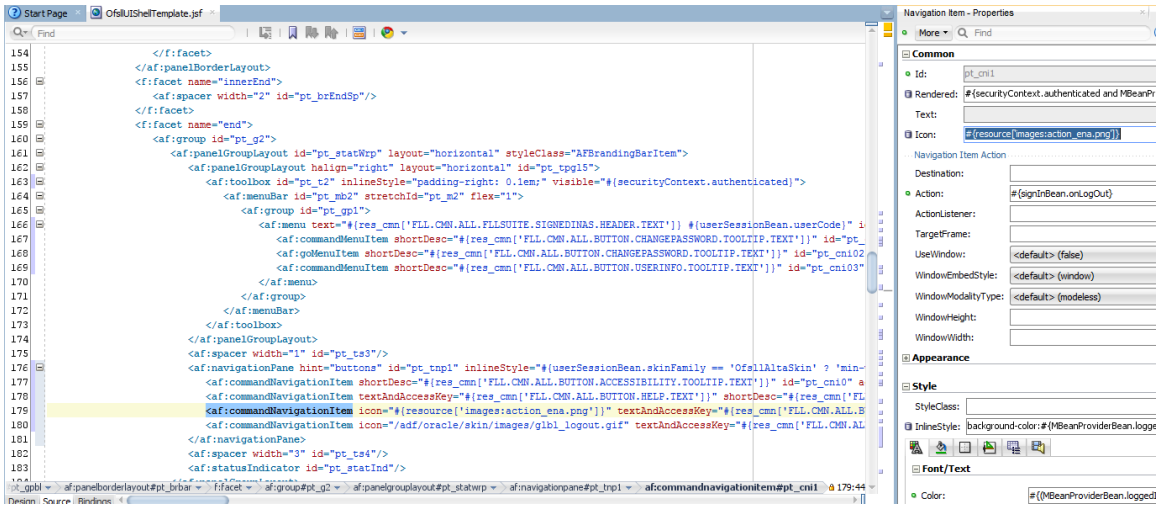
In JDeveloper, pages can be created either by double-clicking a view activity in a task flow or by using the New Gallery. In the Application Navigator, right-click the project to which the page has to be added to, and choose **New**. Then in the New Gallery, expand Web Tier, and click JSF/Facelets. Then select either Page or ADF Page Fragment, and click **OK**.

**Task: Add a Custom Page to a Task Flow**

If the page is created by double-clicking a view activity in a task flow, it is already added to the task flow. If it is created using the New Gallery, it can be added to a task flow by dragging the page from the Application Navigator and dropping it in the task flow diagram editor. Then connect the page using a control flow.

## 4.12 Editing the UI Shell Template

To edit the UI Shell template in JDeveloper, in the **Customization Developer** role, select the *OfsllUiShellTemplate.jsf* file and open and changes can be made as necessary.




## 4.13 Deploying ADF Customizations and Extensions

After customizing existing artifacts, JDeveloper can be used to deploy the customizations to Oracle Weblogic Server.

The default customization workspace as described in Section 3.1, 'About Using JDeveloper for Customization', contains a MAR profile. By default, the name of the MAR profile is application_name_customizations. It will automatically include the customizations that are implemented. This profile can be used to package the customizations for deployment.

When customizations are packaged from the customization workspace, the MAR file should include only library customizations. Do not include the User Metadata or HTML Root Dir for Project in the MAR profile, unless explicitly directed to do so by product documentation.

If the application is extended with new custom artifacts, JDeveloper can be used to package them into an ADF Library JAR and place them into the proper location within the application directory structure.

**Task: Deploy the Customizations**

JDeveloper can be used to deploy the customizations directly or to create a MAR, and then load the MAR using WLST commands or the WebLogic Server Administration Console.

When customizations are deployed on ADF Business Component objects (such as entity objects and view objects), the server must be restarted for the customizations to be picked up.

**Task: Package New Artifacts into ADF Library**

If the application is extended with new custom artifacts (or new artifacts are supplied with), these artifacts must be packaged into an ADF library JAR and place the JAR files in the proper location within the application.

The ADF library JAR for the new model artifacts (such as entity objects and view objects) should be placed into the /APP-INF/lib directory. The ADF Library JAR for the new user interface artifacts (such as pages) should be placed in the /WEB-INF/lib directory

## 4.14 Deployment Options

The Deployment or EAR creation of the application would be done through OracleFSLLEnterpriseApp project. In this Project, JPR has the necessary deployment profiles available. Deployment of the application on to Weblogic Server is defined as per “Install UI Components to Application Server” document.

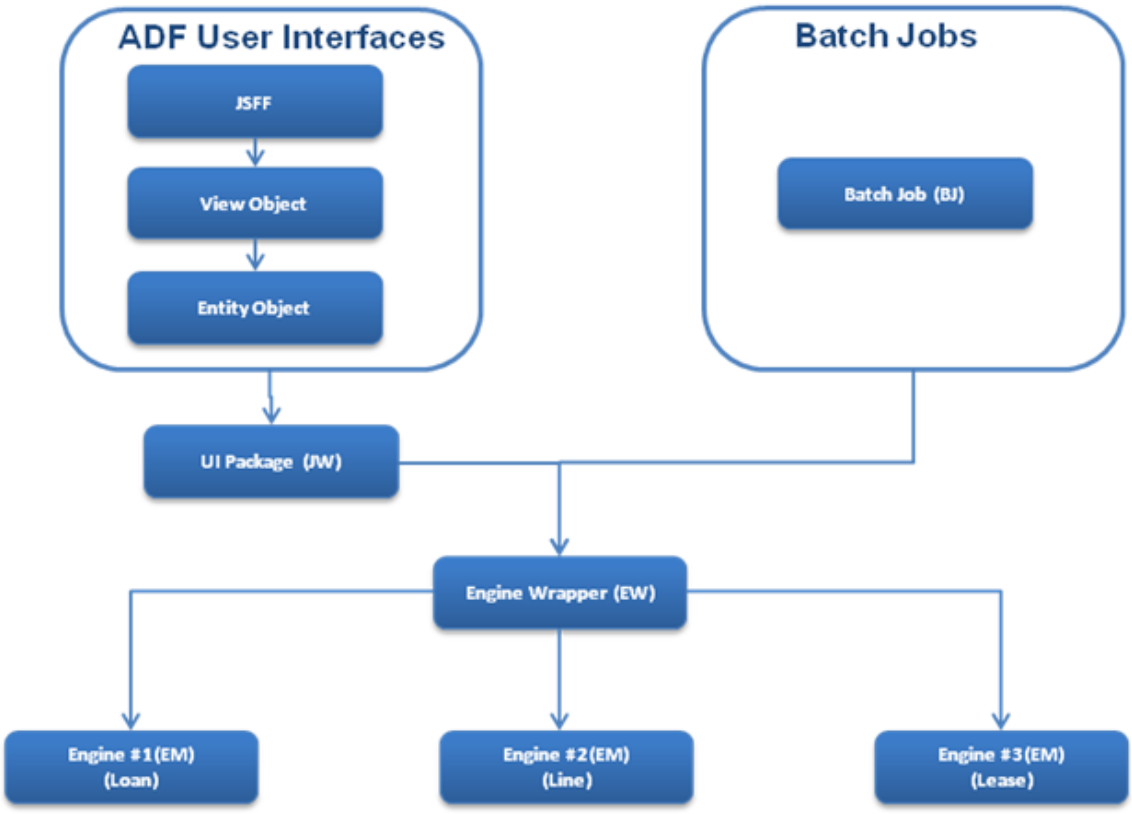
**Note**

- In *Customization Developer* role, the project creates the MAR deployment profile for customization deployment.
- MAR deployment is same as EAR deployment.

# 5. Customizing Database Objects

## 5.1 UI – Package Interaction Logic

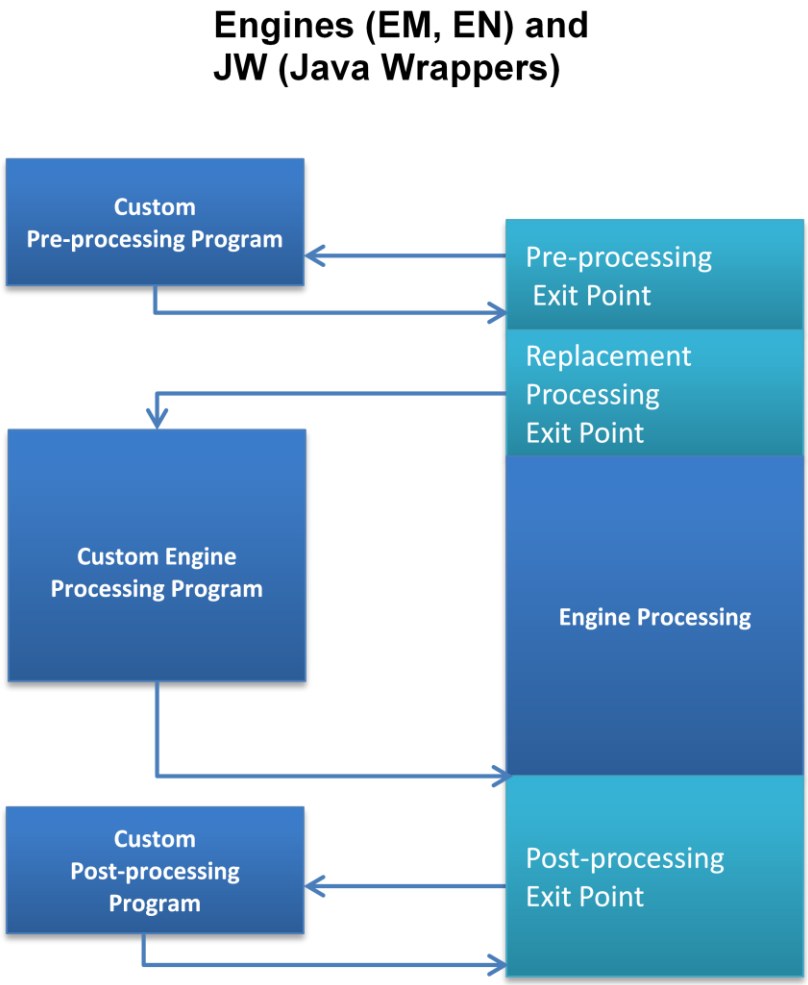
OFSLL uses the Oracle Fusion Middleware based ADF user interface. Below mentioned image show how OFSLL user interfaces interacts with the Java wrapper.




## 5.2 UI Java Wrapper (U*JW)

If the java wrapper engine needs to be customized, follow the steps given below:

1. Select the Exit point for the customization.
2. Rename the exit point package file name with _xyz. Do not change Package name.
3. Change the variable CV from NON_CUSTOMIZED to CUSTOMIZED depending upon the exit point before, replace or after.
4. Write the required customized engine library and call it in the java wrapper Exit Points Package (EX).



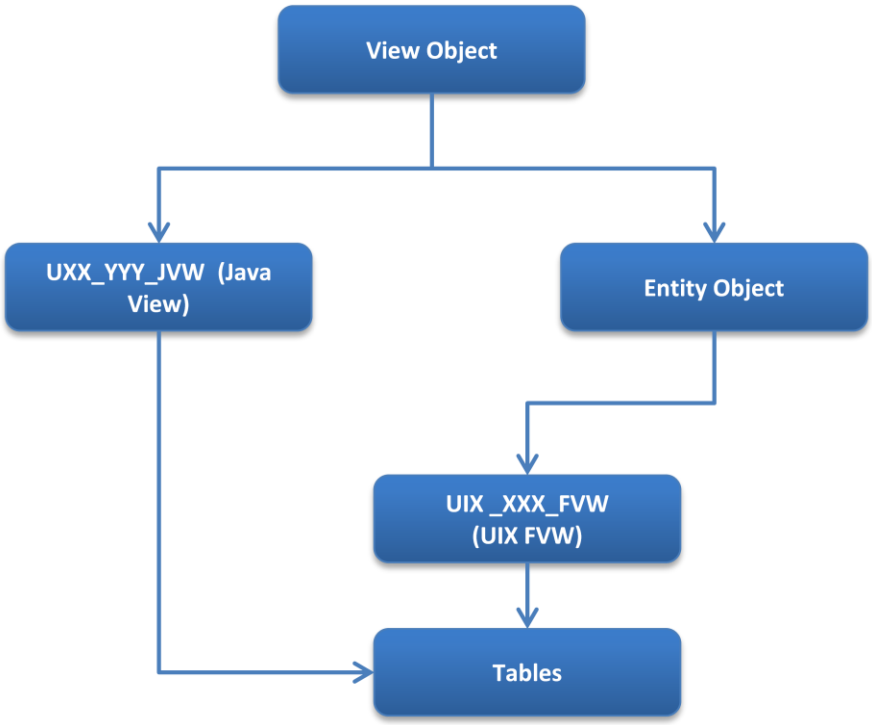

**Business Logic Engine (Process)**

## 5.3 Database Schema

Oracle Financial Services Lending and Leasing has the below mentioned Database Objects

- Table
- Table Column
- Sequence
- Index
- View

- FVW – User Interface Views
- JVW – Java Interface Views
- EVW – Engine/Wrapper Signature Views
- PL/SQL Programs




## 5.4 Wrapper Engine model

Below mentioned is the naming convention for Wrapper Engine model used in Oracle Financial Services Lending and Leasing.

| **XXXYYY**_ZZ_ABC_99 | | XXXYYY_ZZ_**ABC_99** | |
|---|---|---|---|
| XXX | Module or Engine | A | System |
| YYY | Function | | 0-Common |
| ZZ | Program Type | | 1-Consumer |
| EM | Engine Main | | 2-Commercial |
| EN | Engine Function | | (Always 0 for Wrapper) |
| EW | Engine Wrapper | B | Product Type |
| EL | Engine Library | | 0-Common |
| EX | Engine User Exits | | 1-Loan |
| JW | Java Wrapper | | 2-Lease |

| **XXXYYY**_ZZ_ABC_99 | | XXXYYY_ZZ_**ABC_99** | |
|---|---|---|---|
| BJ | Batch Job | | 3-WFP |
| BL | Batch Job Library | | (Always 0 for Wrapper) |
| CL | Common Library | C | Product Sub Type |
| | | | 0-Common |
| | | | 1-Closed Ended |
| | | | 2-Open Ended |
| | | | (Always 0 for Wrapper) |
| | | 99 | Running Sequence Number |
| | | | Starting 01 to 99 |

## 5.5 Batch Job (BJ)

Batch Job **cannot** be customized, it has to be developed as a new job.

## 5.6 Engine Wrapper (EW)

Engine Wrapper **cannot** be customized.

## 5.7 Main Engine (EM)

To customize the main engine, follow the steps given below:

- Select the Exit point for the customization.
- Rename the exit point package file name with _xyz. Do not change Package name.
- Change the variable CV from NON_CUSTOMIZED to CUSTOMIZED depending upon the exit point before, replace or after.
- Write your customized engine and call it in the Engine Exit Points Package (EX).

## 5.8 Engine Function (EN)

To customize an engine function, follow the steps given below:

- Select the Exit point for the customization.
- Rename the exit point package file name with _xyz. Do not change Package name.
- Change the variable CV….from NON_CUSTOMIZED to CUSTOMIZED depending upon the exit point before, replace or after.
- Write your customized engine function and call it in the Engine Exit Points Package (EX).

## 5.9 Engine View

To customize an Engine View (EVW), follow the steps given below:

- Do Not modify the OFSLL Base Engine View Script
- Create a copy of the OFSLL Base Engine View Script, rename and modify that Engine View Script.

Do **not** modify the OFSLL Base Engine View Name.

## 5.10 Common Features

- Error Logging
  - Alert Log
- Debugging
  - Debug Log
- Version Control Header in each code unit

## 5.11 Seed Data

Oracle Financial Services Lending and Leasing Seed data tables are classified in following three categories

System

- Only Oracle Financial Software Services Ltd. can change/update this data

Combination

- Oracle Financial Software Services Ltd. or customer can change/update this data. It is recommended to identify all the new customized seed data records with a customer identifier in the primary key.

Demo

- Oracle Financial Software Services Ltd. provides the demo data as sample demo configurations. Customer can change/update/delete this data, this data should **not** be used for production configurations.

All seed data tables have two Primary Keys - one is user defined codes and the other is a system generated sequence number.

All seed data tables have a system defined indicator to indicate whether a record is system defined.

All seed data are stored is in files and checked in the version control systems and sent as merged statements in patch for changed (added or modified) data.

## 5.12 Developer's Tips

Suppose the account number generation needs to be customized different from what OFSLL generates; Requirement is to replace the baseline format with its own format (like ACC-NNNNNNN).

Locate the procedure that generates the account number.

- Procedure “set_acc_nbr” from program “aaiacc_en_111_01.pkb” generates account number in “YYYYMMNNNNNNND” format.

Identify the exit point package having the set_acc_nbr_xxx procedures where xxx is bfr – before, afr - after and rep – replace.

- aaiprc_ex_111_01.pks and aaiprc_ex_111_01.pkb
- Create new package with name as **xyz**aaiacc_en_111_01.pkb. Add procedure to create account number in new format.
- Copy “aaiprc_ex_111_01.pks” to “xyzaaiprc_ex_111_01.pks”
- Modify “**xyz**aaiprc_ex_111_01.pks”, change constant

  From

  *CV_SET_ACC_NBR_REP CONSTANT VARCHAR2(30) := cmncon_cl_000_01.NOT_CUSTOMIZED;*

  To

  *CV_SET_ACC_NBR_REP CONSTANT VARCHAR2(30) := cmncon_cl_000_01.CUSTOMIZED;*
- Copy “aaiprc_ex_111_01.pkb” to “xyz aaiprc_ex_111_01.pkb”
- Call new procedure from **xyz**aaiprc_ex_111_01

```
PROCEDURE set_acc_nbr_rep(
                          iv_con_rec IN aai_con_evw%ROWTYPE
                         ,iv_acc_aad_id IN OUT aai_con_evw.con_aad_id%TYPE
                         ,iv_acc_nbr    IN OUT aai_con_evw.con_acc_nbr%TYPE) IS
BEGIN
      xyzaaiprc_en_111_01.set_acc_nbr(iv_con_rec,iv_acc_aad_id, iv_acc_nbr );
END set_acc_nbr_rep;
```

The above example shows the usage with replacement exit point. Similar way “before” and “after” exit points can be used to extend the business logic functions.

- Where ‘**xyz**’ is Customer Unique Id

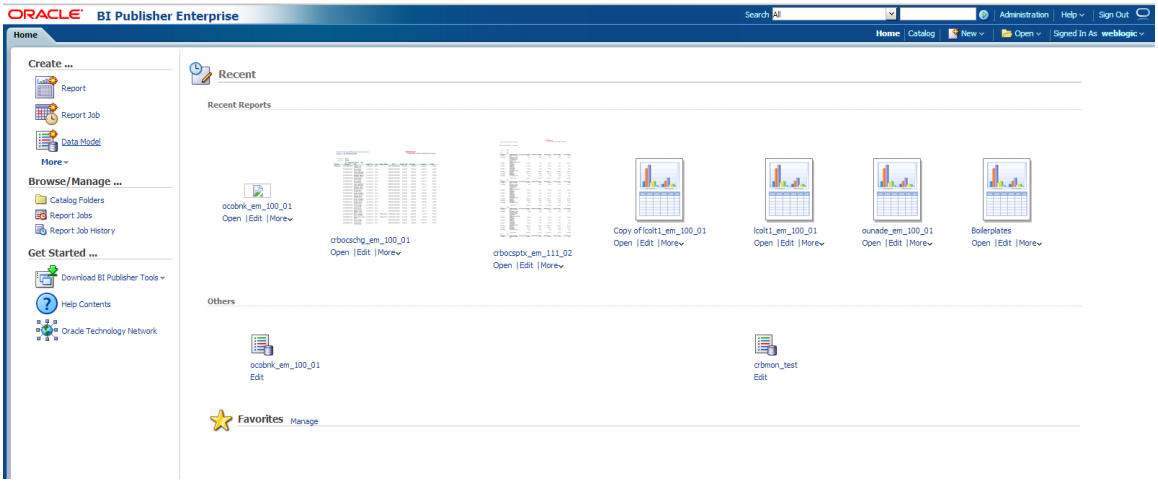
# 6. Creating New Custom BI Publisher Report/Letter

## Pre-Requisites

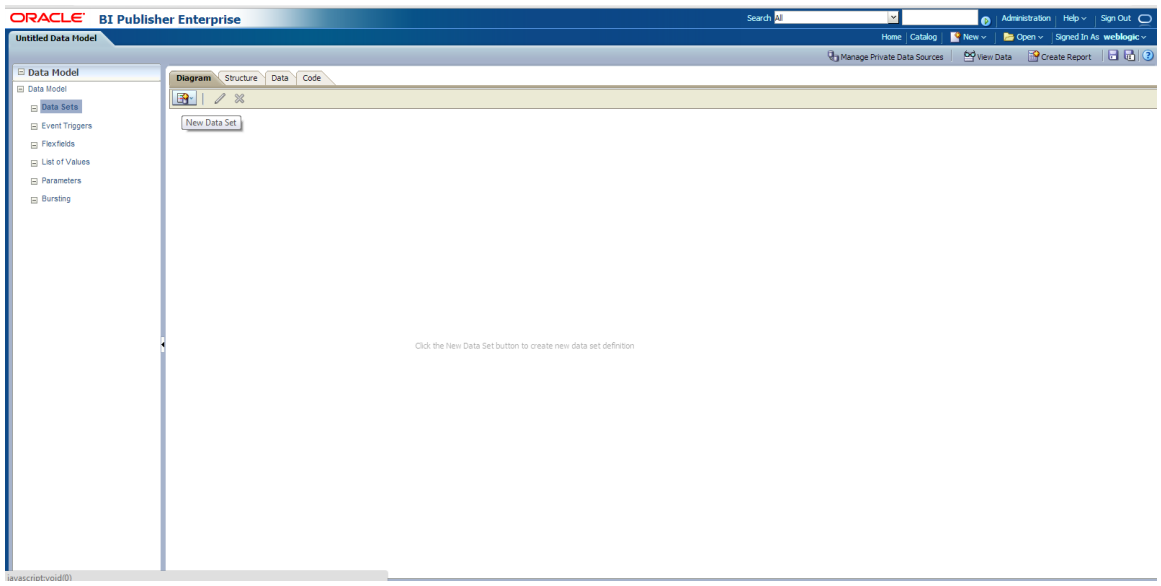
1. Changes to base reports not allowed
2. Basic knowledge on BIP and BIP client.
3. The reports should be placed into the same folder structure
   - i.e. For Reports → Shared Folders/oracle/fll/xmlp/reports
   - For Letters → Shared Folders/oracle/fll/xmlp/letters
   - For Correspondences → Shared Folders/oracle/fll/xmlp/correspondence

## Creating a New Report

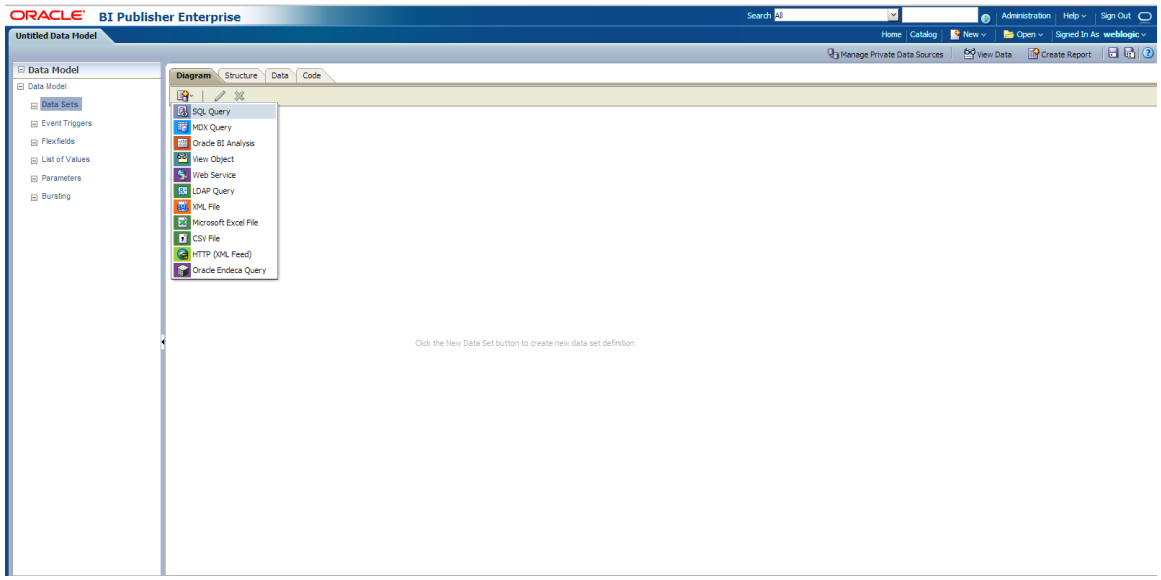
1. Login into BIP console.
2. To Create a new report first create the data model , Click on Data Model on left.



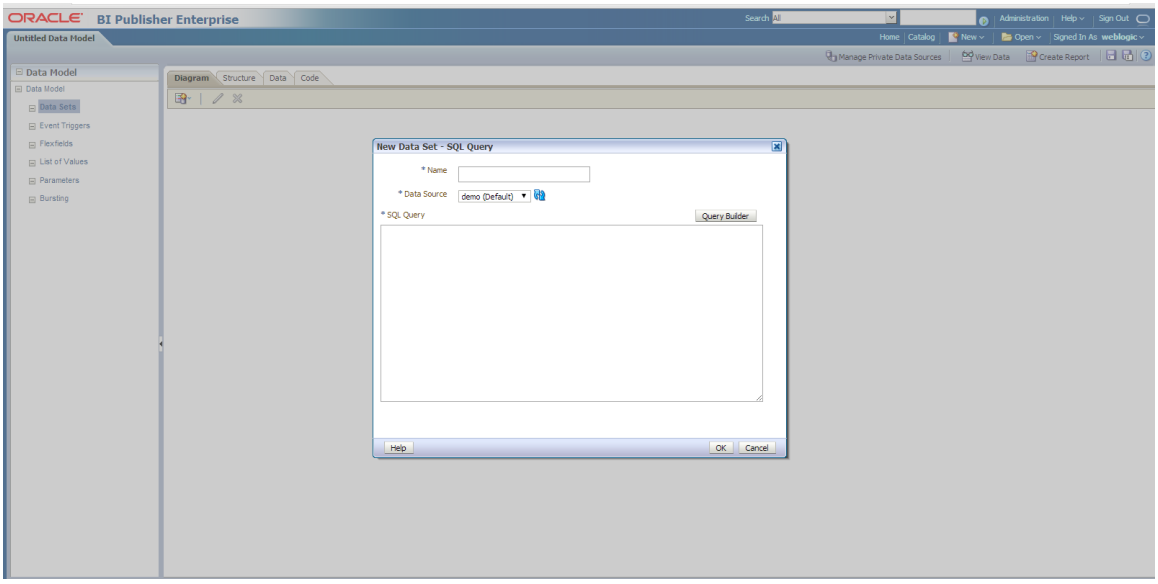

3. On click on Data Model it will open the following Screen.



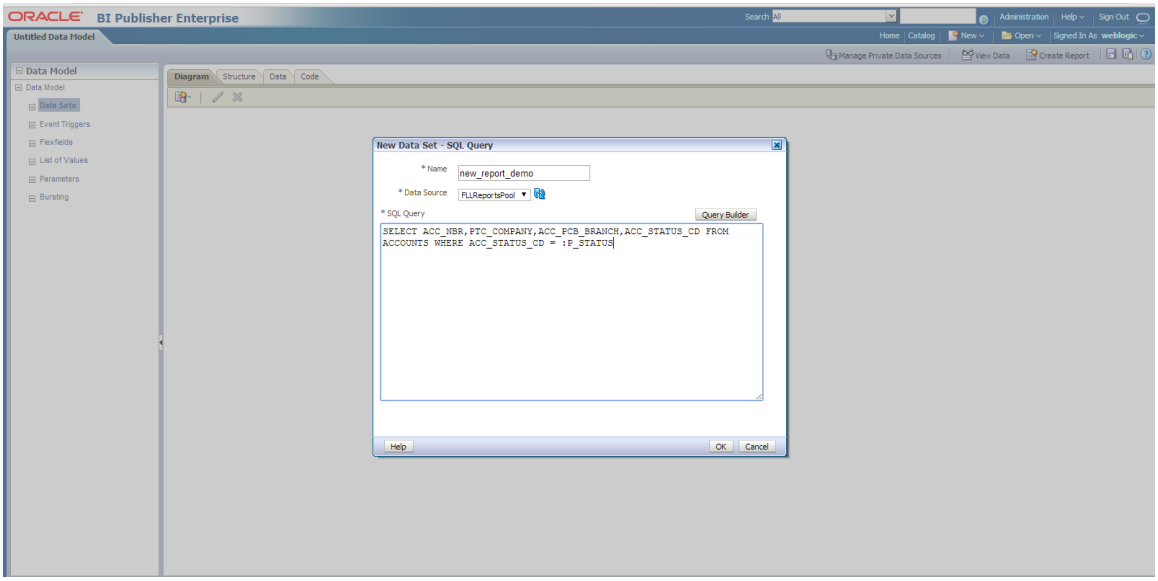

4. To create a New Data Set, Click on New Data Set and select the type “SQL Query”



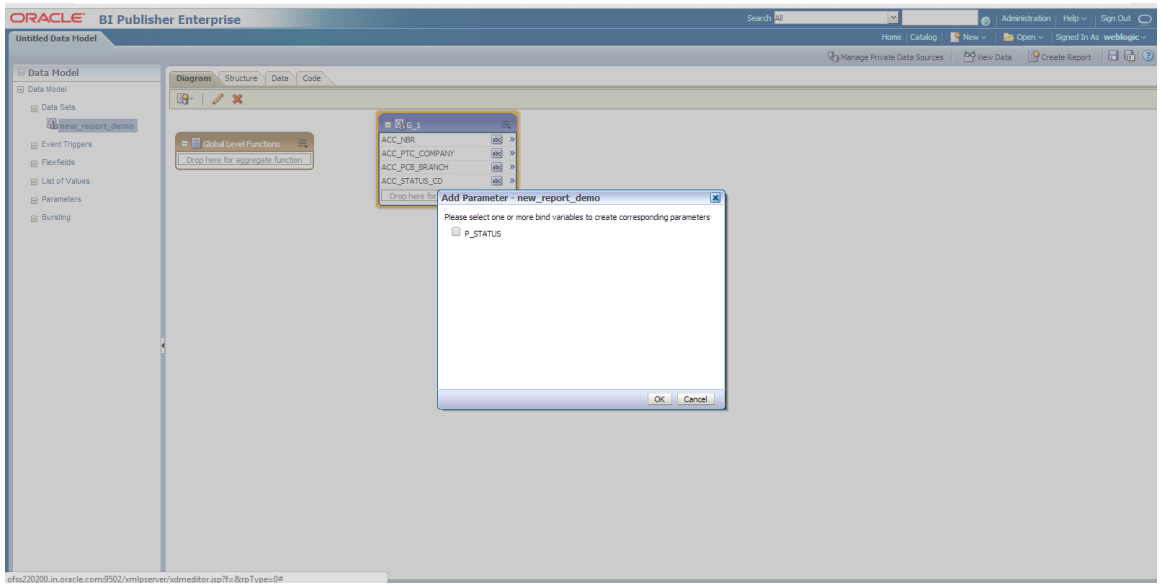

The following screen is displayed.



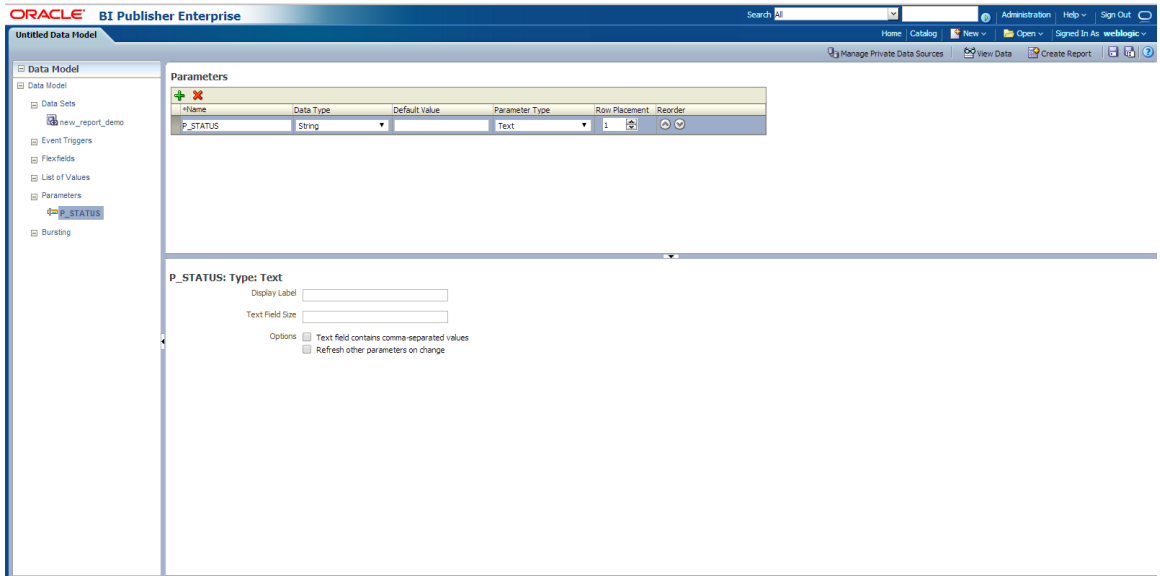

5. Enter the new data model name (Use the same name while creating the report layout)
6. Select the Data Source.
7. Add the sql query. It depends on what data needed on the report and what should be the parameters.




8. Click OK. A confirmation dialog is displayed to create the parameter. Select the parameter and press OK.

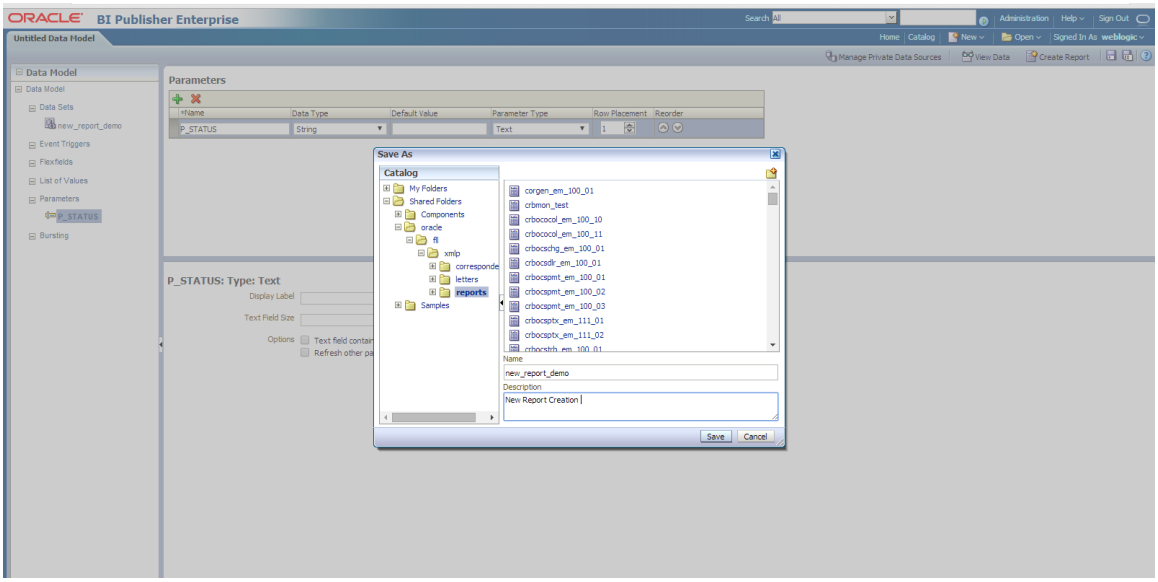


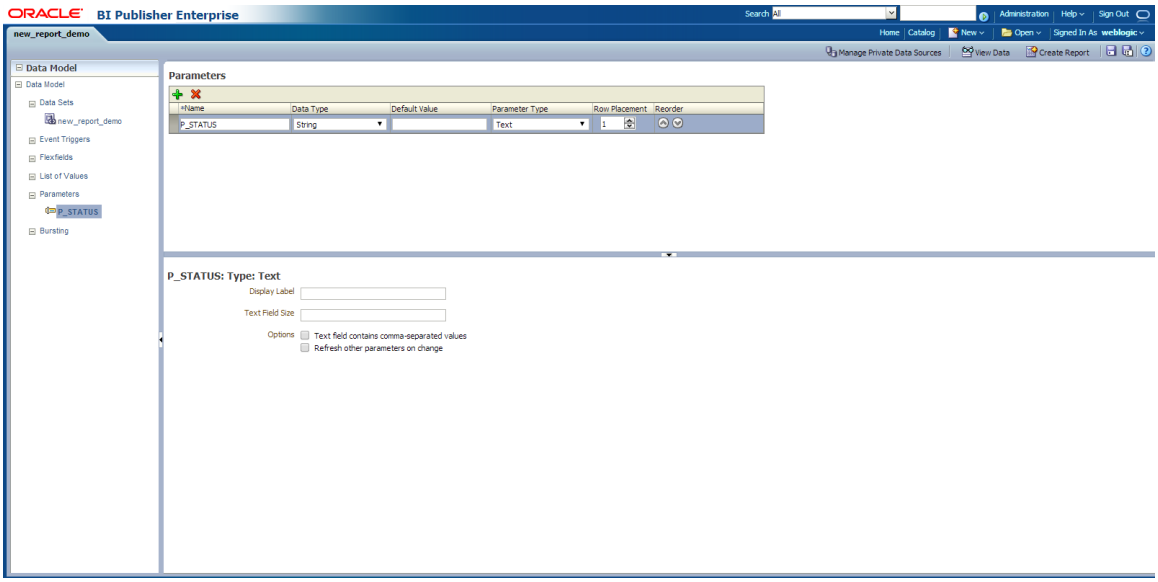

9. The parameter is created as indicated below.



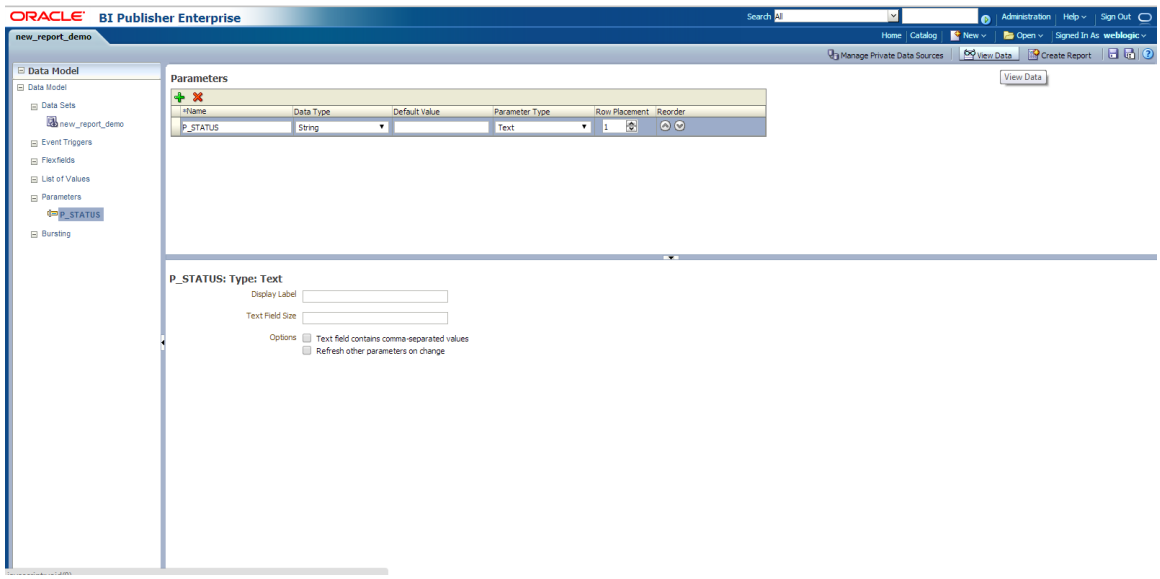

10. Save the data model. Select the directory in which you would need to save the details. For Reports save in reports directory and for Letters save it in Letters directory.

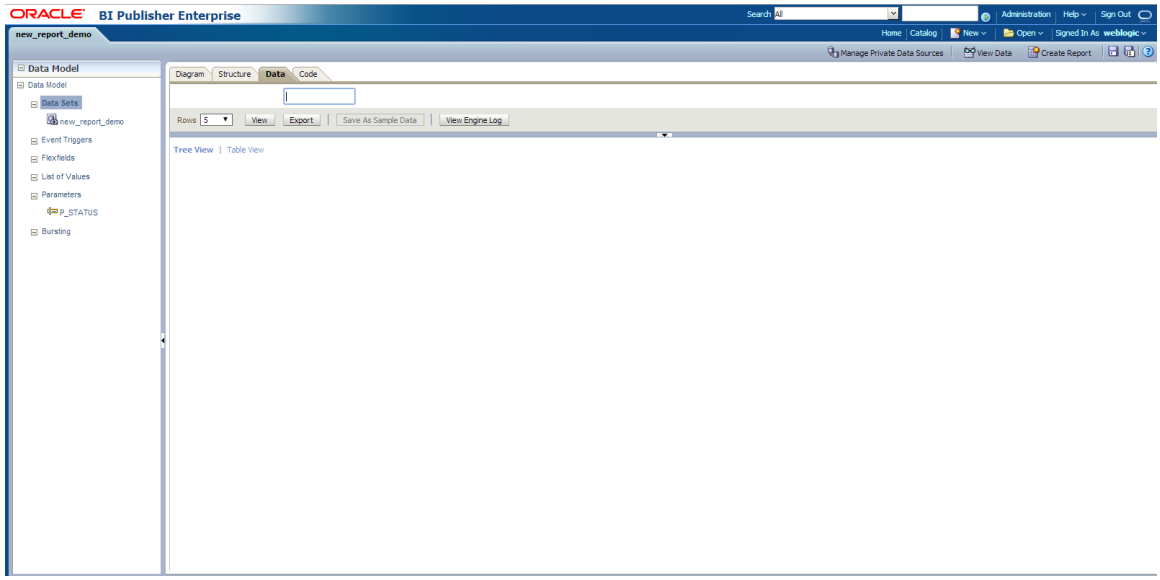
11. You can also save with the same name and specify the description of the report.



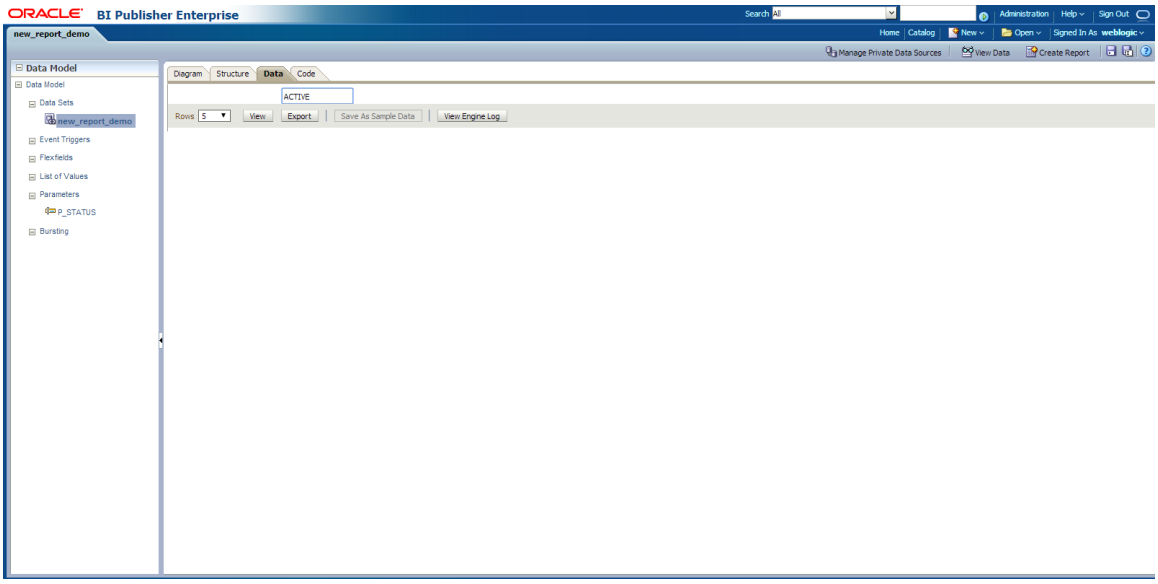




12. Once the Data Model is created, create the sample data as indicated below.
13. Click on View Data on right side of the page.

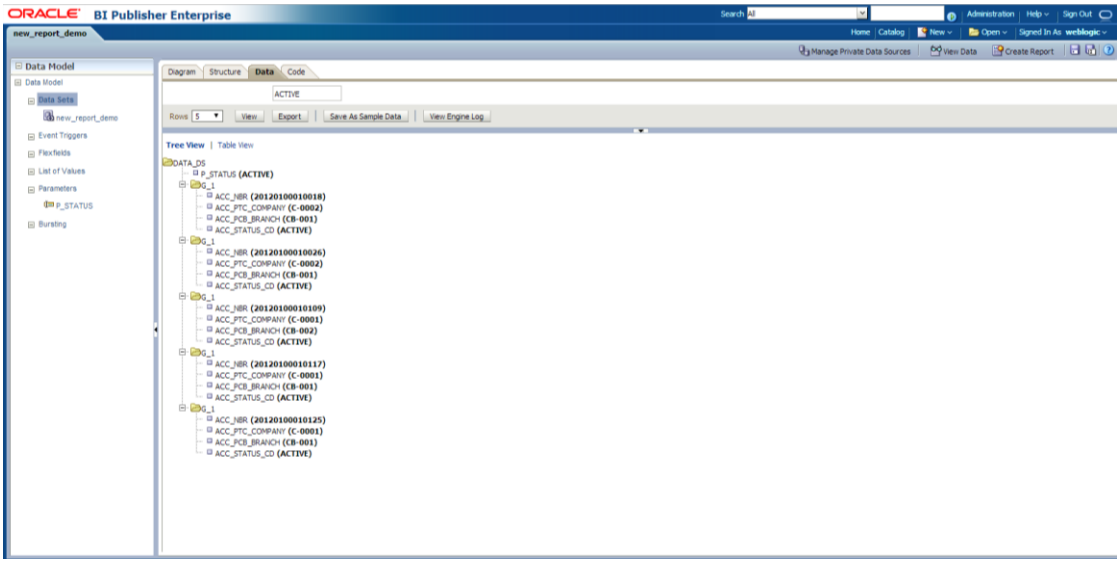


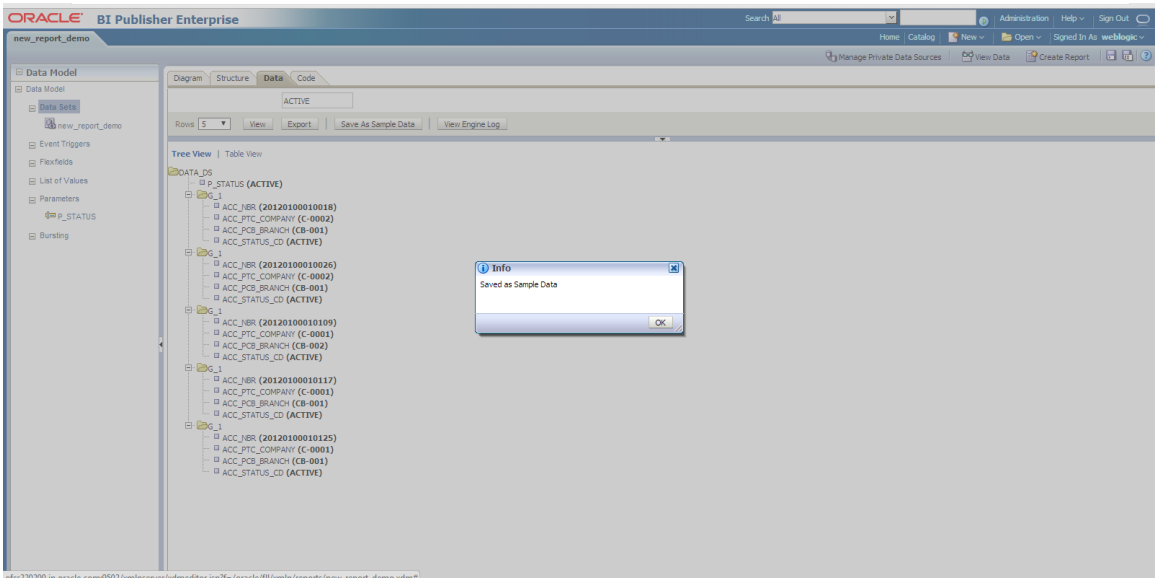

14. Specify the following value in Parameter (P_STATUS).




15. Click ‘View’. The sample data is created as indicated.



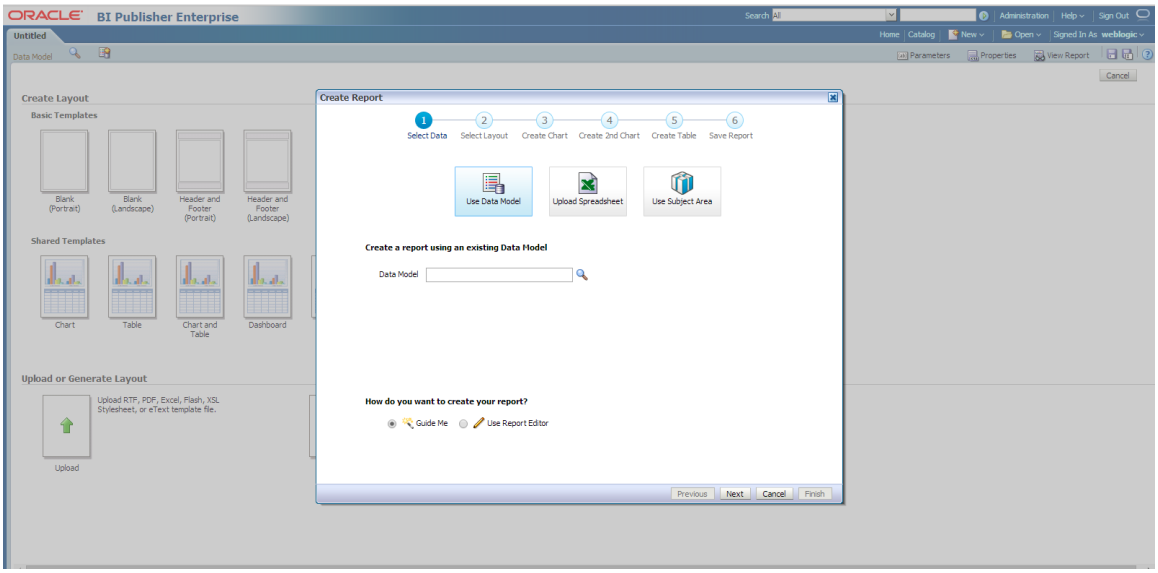

16. Click on ‘Save As Sample Data’.



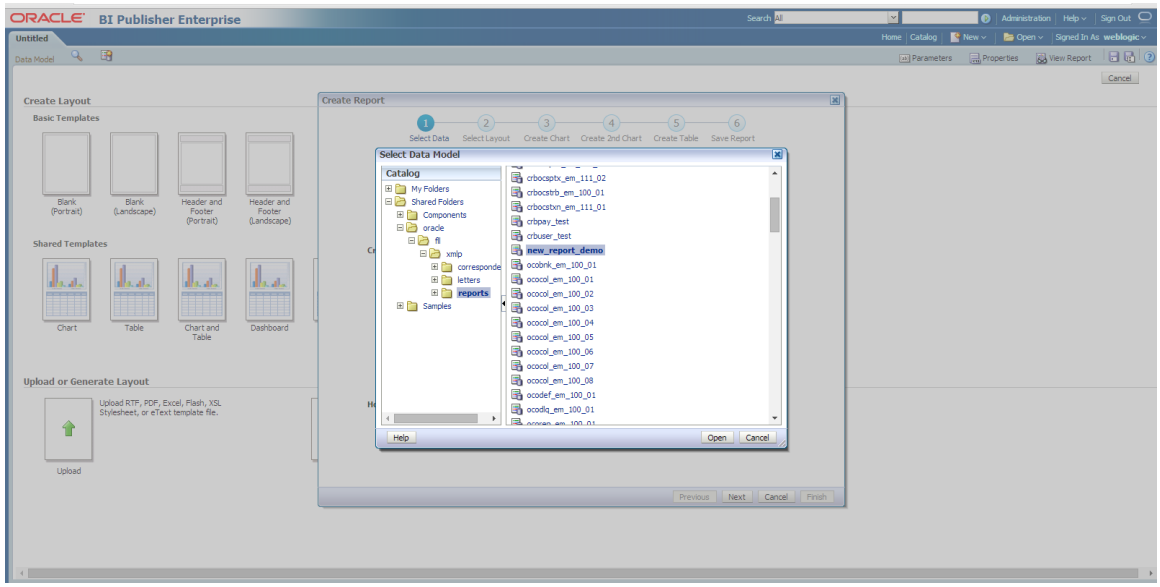

17. The Data model is created and can be use to create the report layout.

## 6.1 Create Report Layout

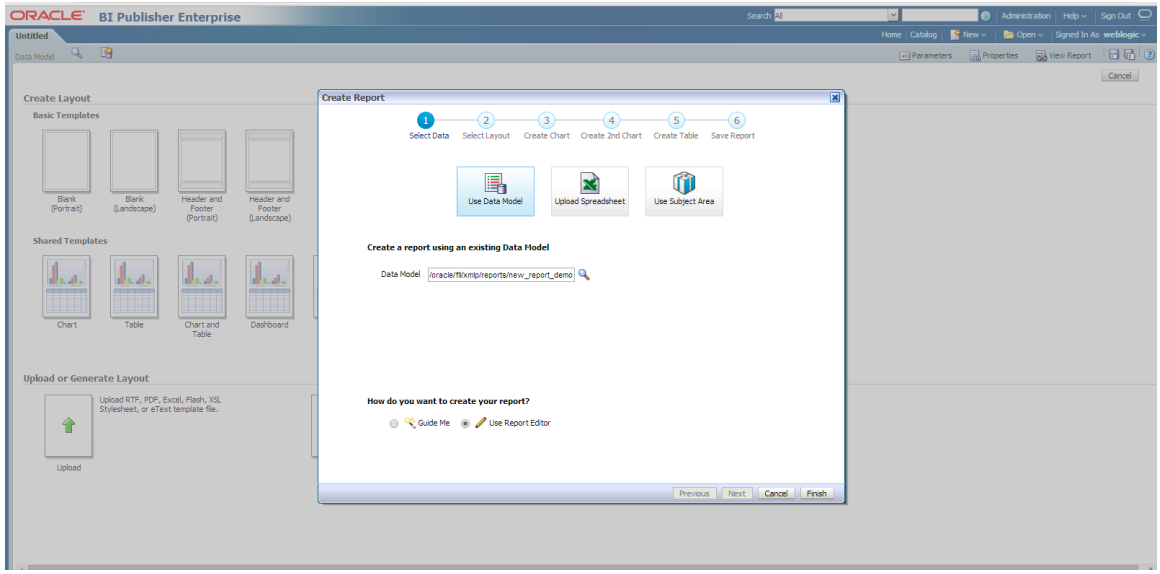
1. Navigate to Home Page and create the report layout as indicated below.
2. Click on Create > Report on LHS panel.




3. Select the ‘Use Data Model’ and click to browse the existing data model , and select the data model just created.

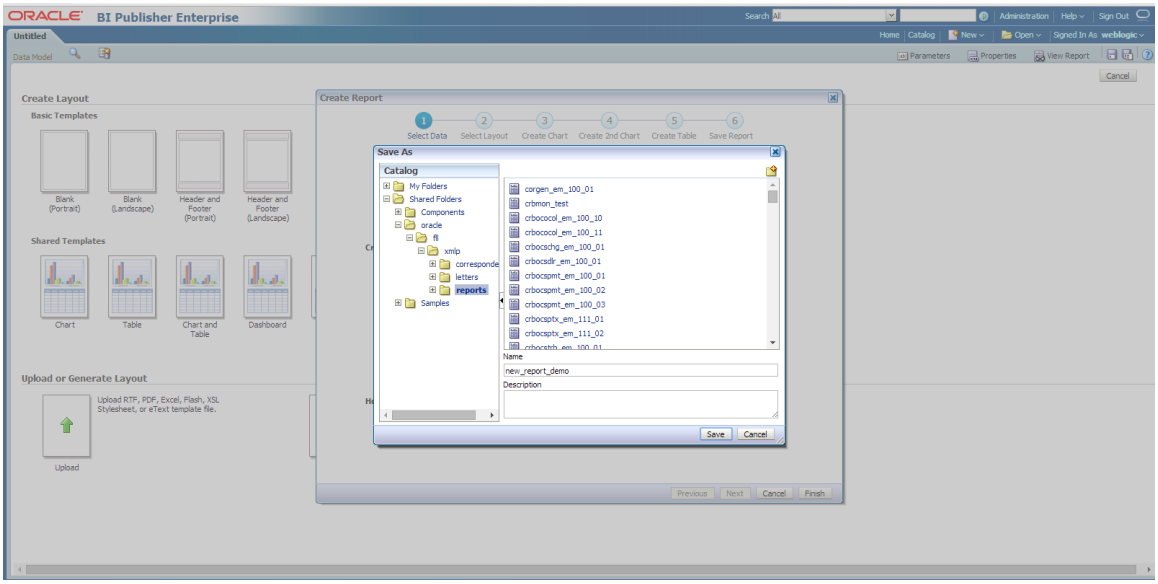



4. Click ‘Open’.

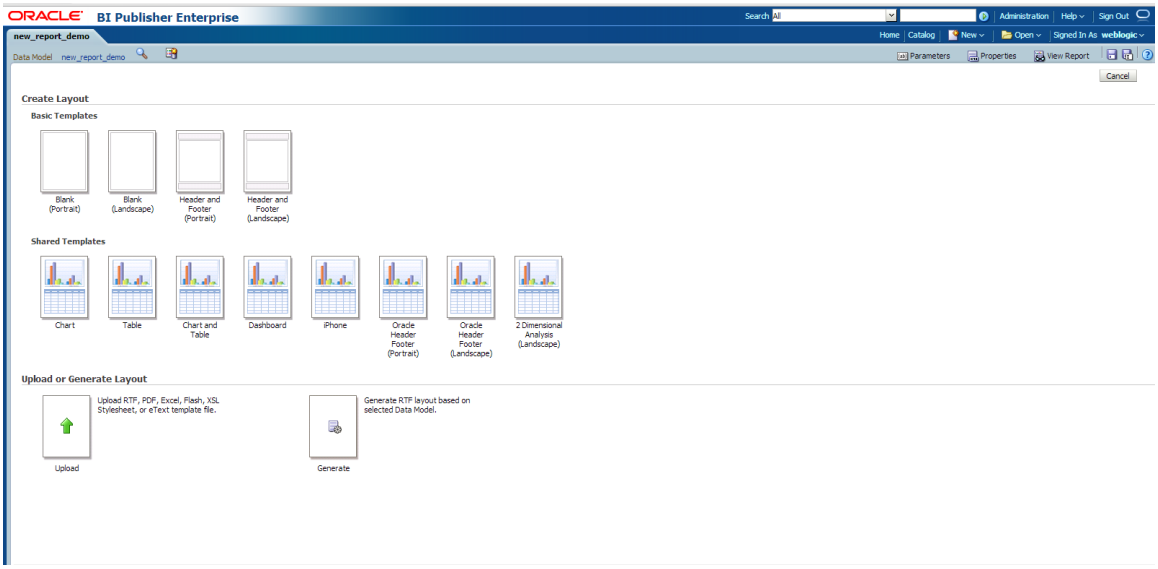



5. Select ‘Use report Editor’ option (If user have the BIP Client installed on his machine , he can create and the .rtf layout from MS Word).

6. Click ‘Finish’ and save the report with the same name as data model.



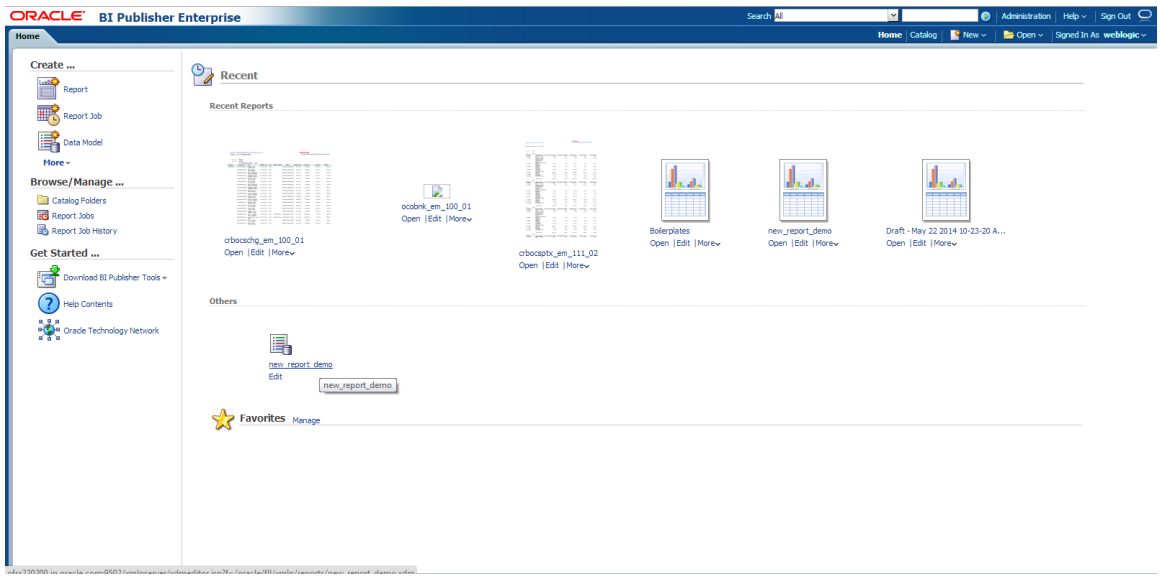

The below screen indicates the saved report.



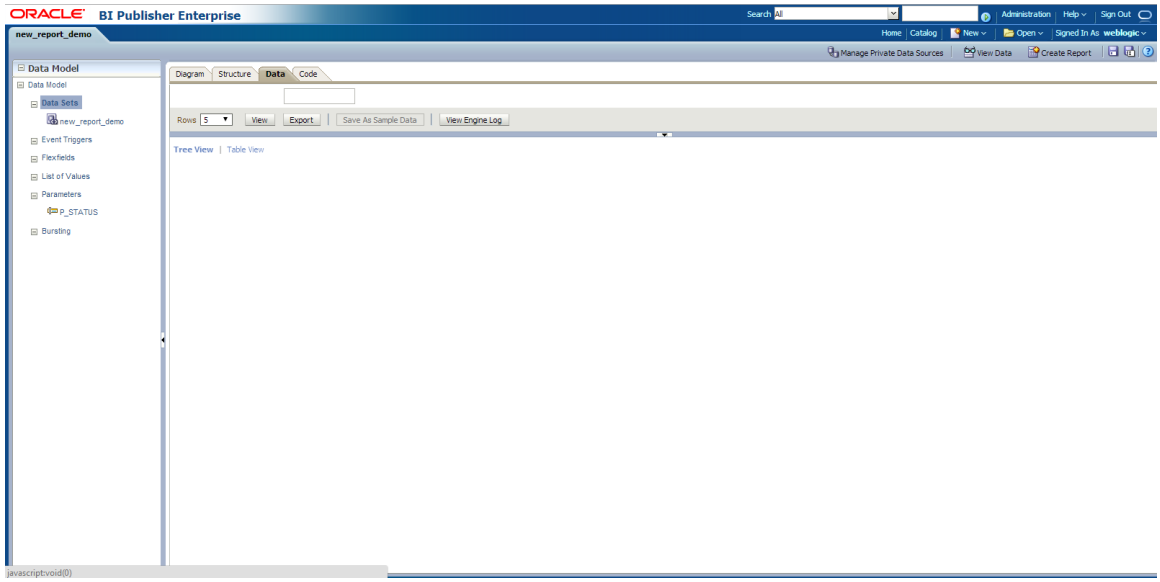

## 6.2 Create XML data

Create XML data to create the layout on local editor (MS Word)

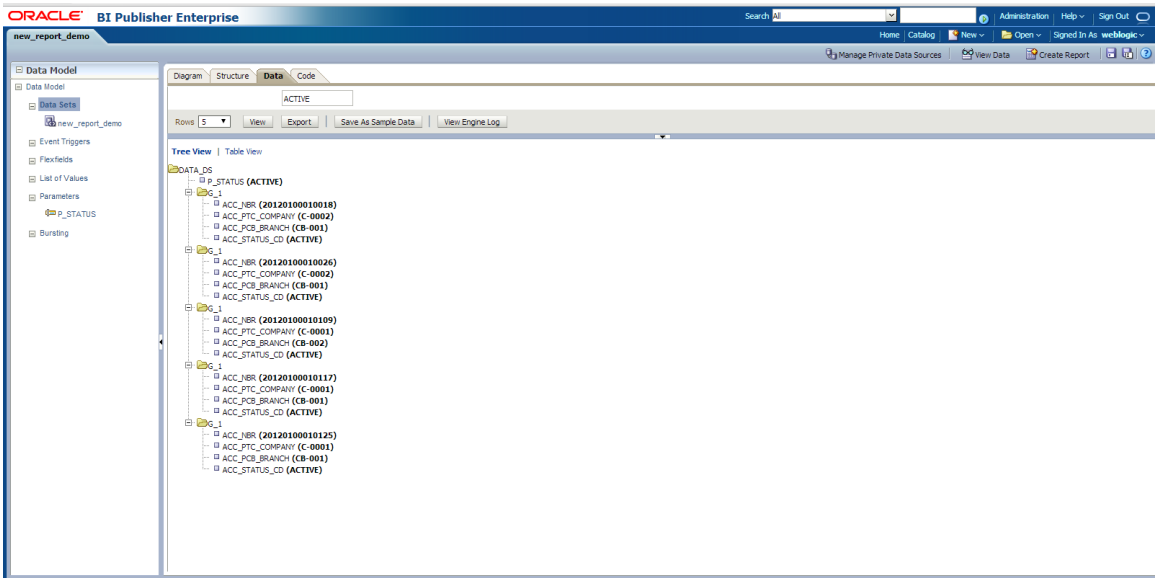
1. Navigate to Home Page and select the data model created.



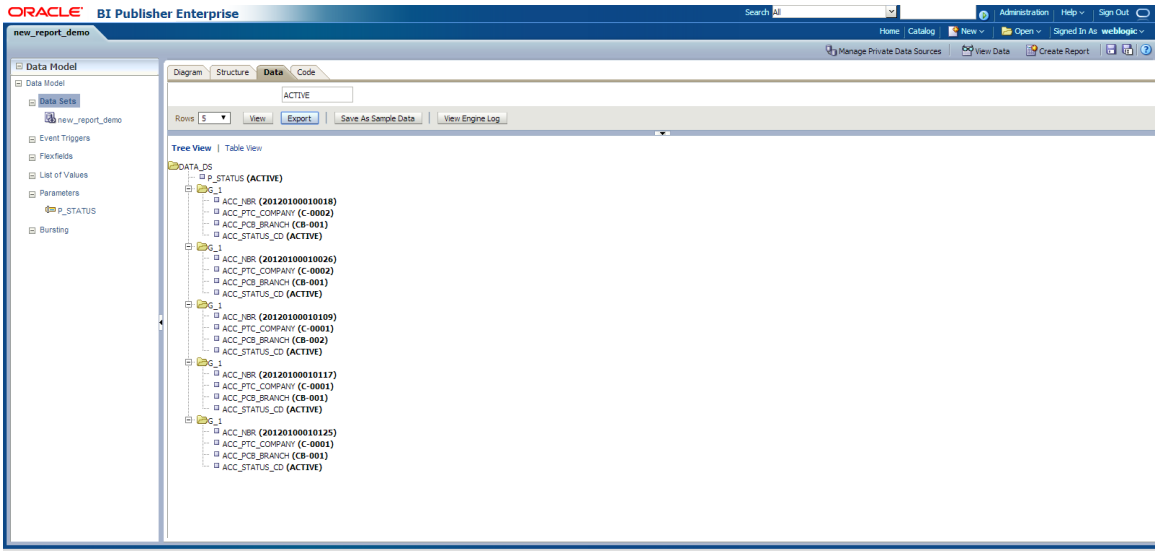

2. Select ‘Data’ Tab.



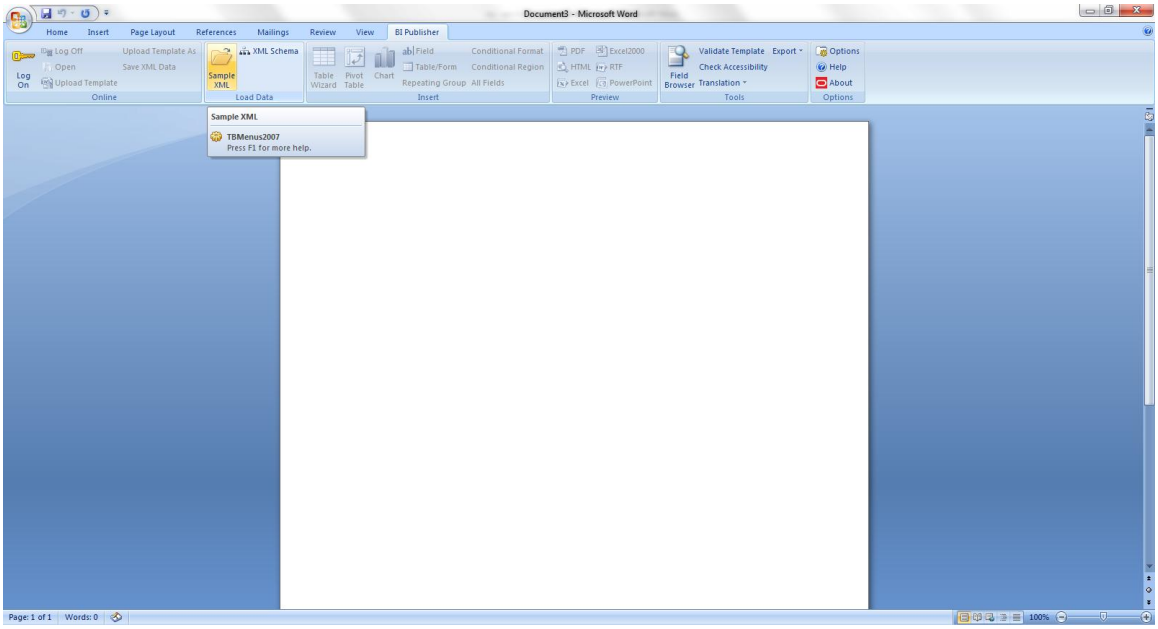

3. Enter the Parameter Value and click ‘View’.



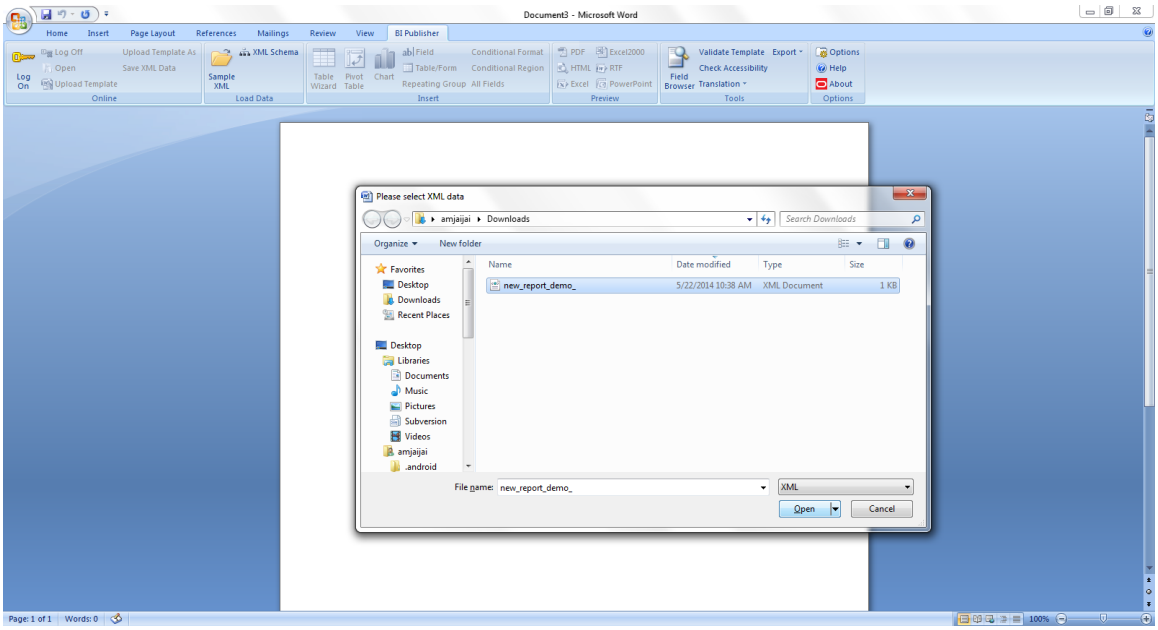

4. Click ‘Export’ (it will create the XML file locally in download folder).



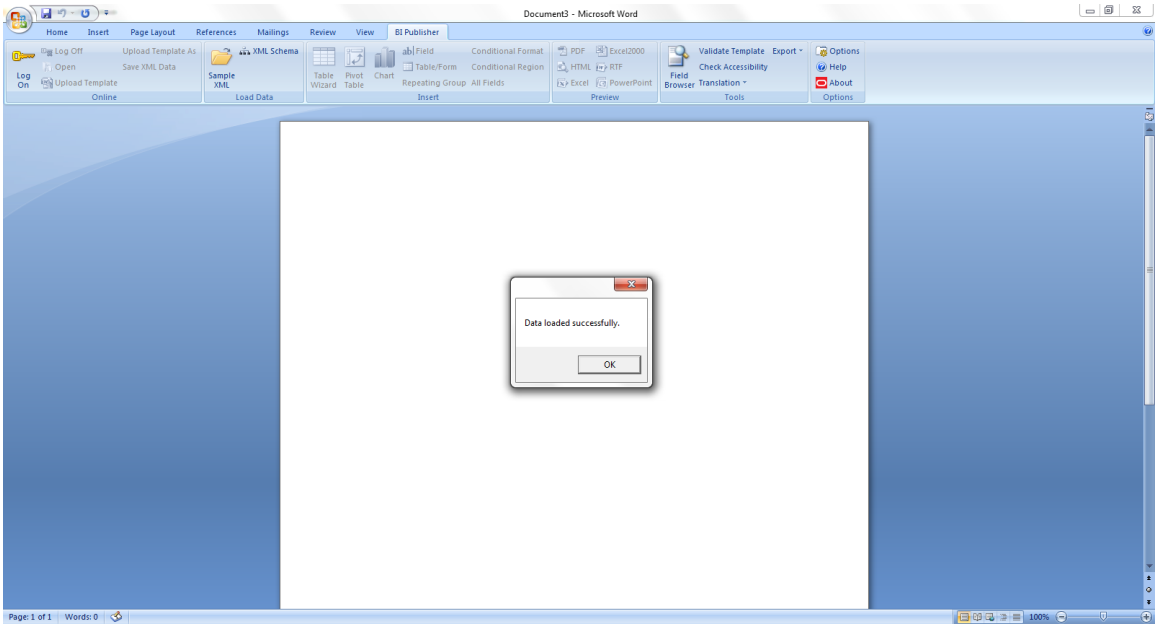

5. Open MS Word ( Ensure that the BIP client is installed on User’s Machine).
6. Access BI Publisher tab and click ‘Sample data’.



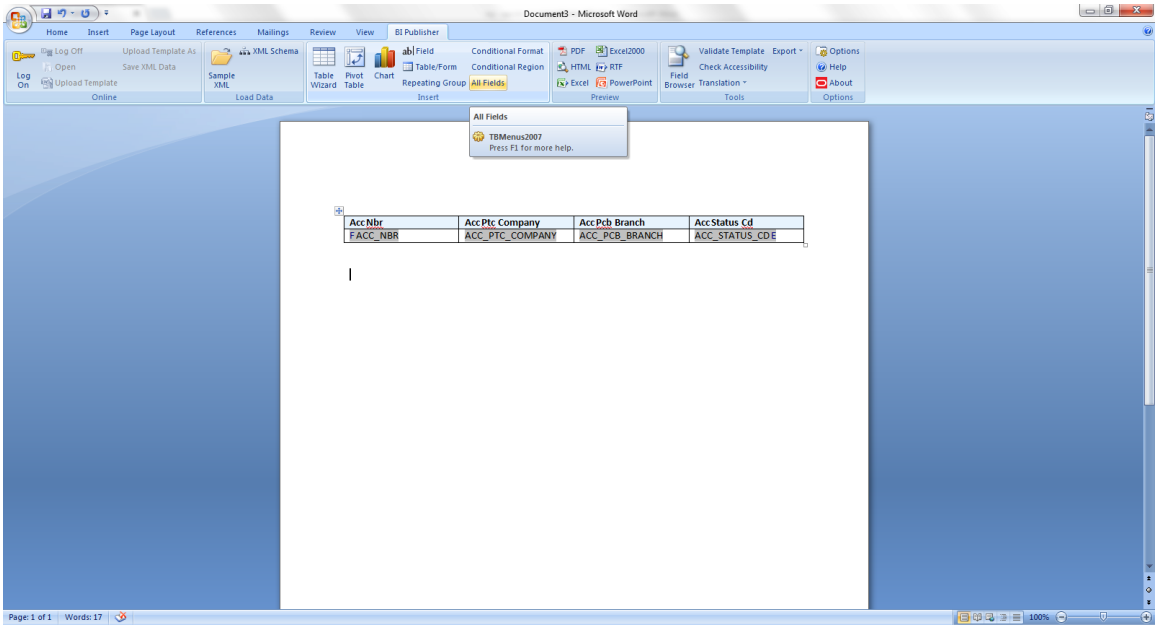

7. Select the same XML file from download.



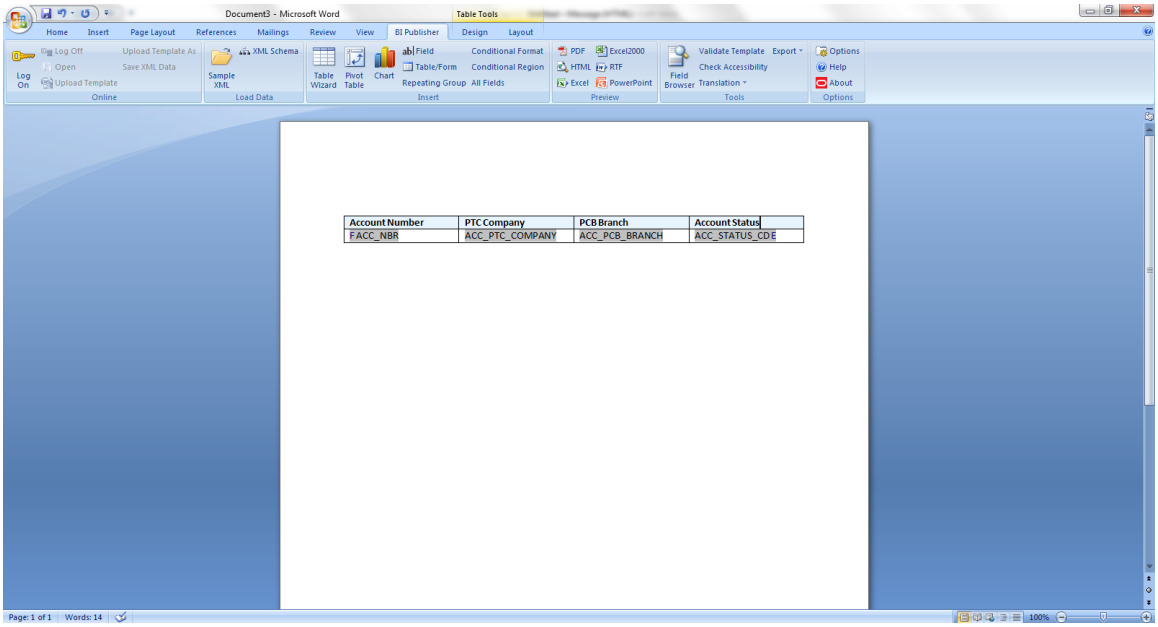

8. Select ‘Open’. Required data is loaded.



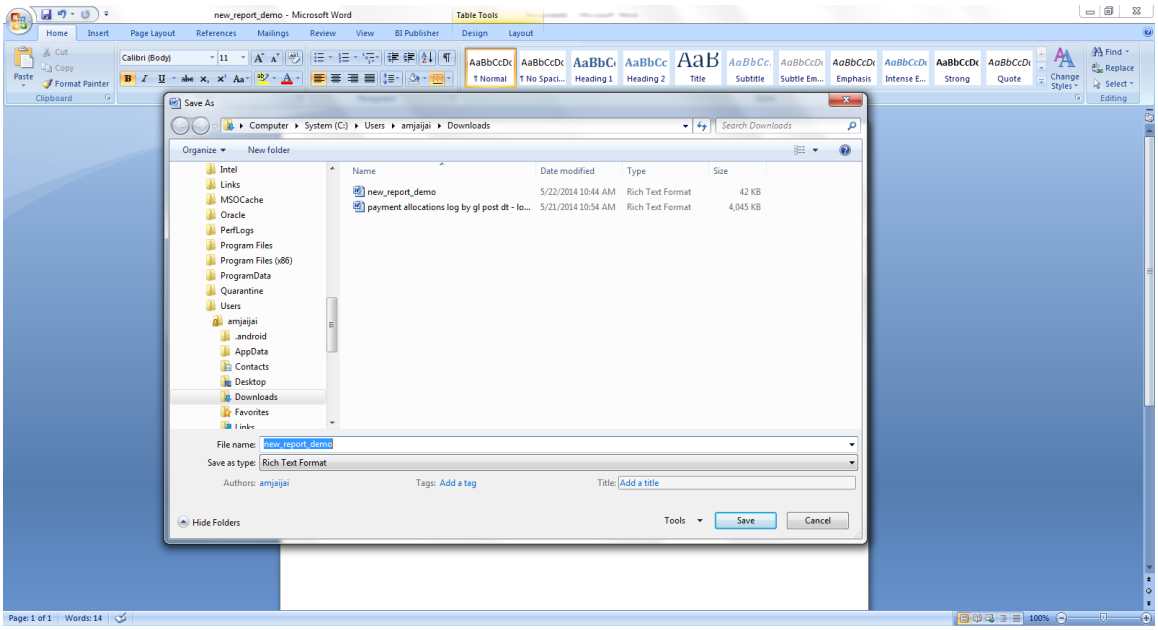

9. You can use the available option is MS word to create the desired layout.
10. First use “All Fields” option. This creates all the fields from sample xml and create the layout as follows in tabular form.



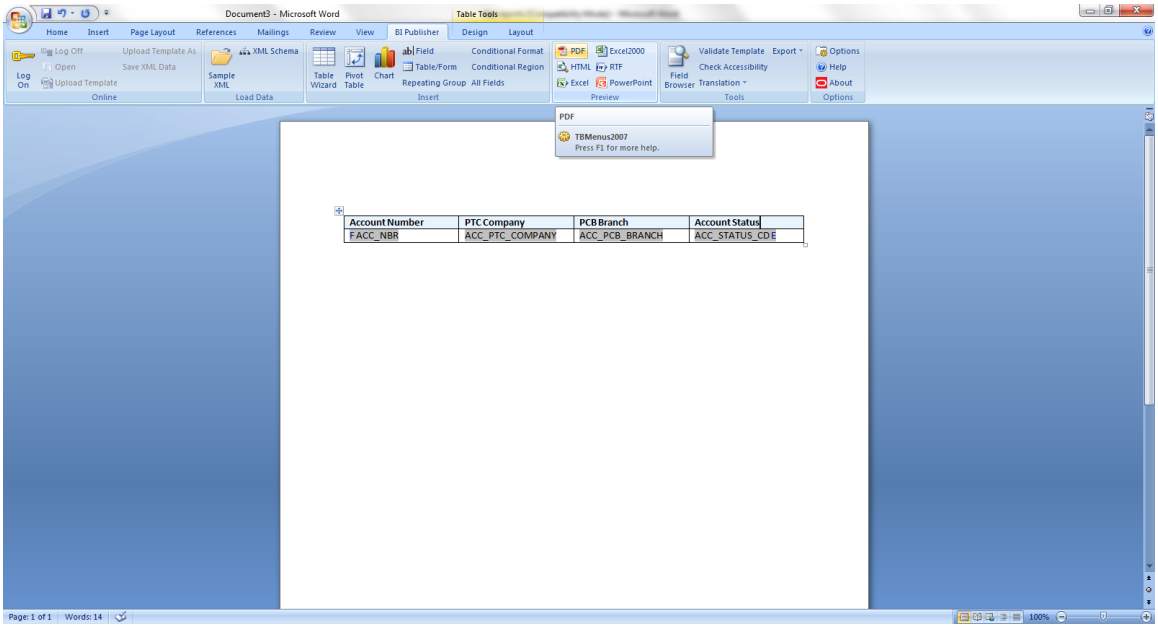

11. You can modify the headings, add new static text, logos and so on.



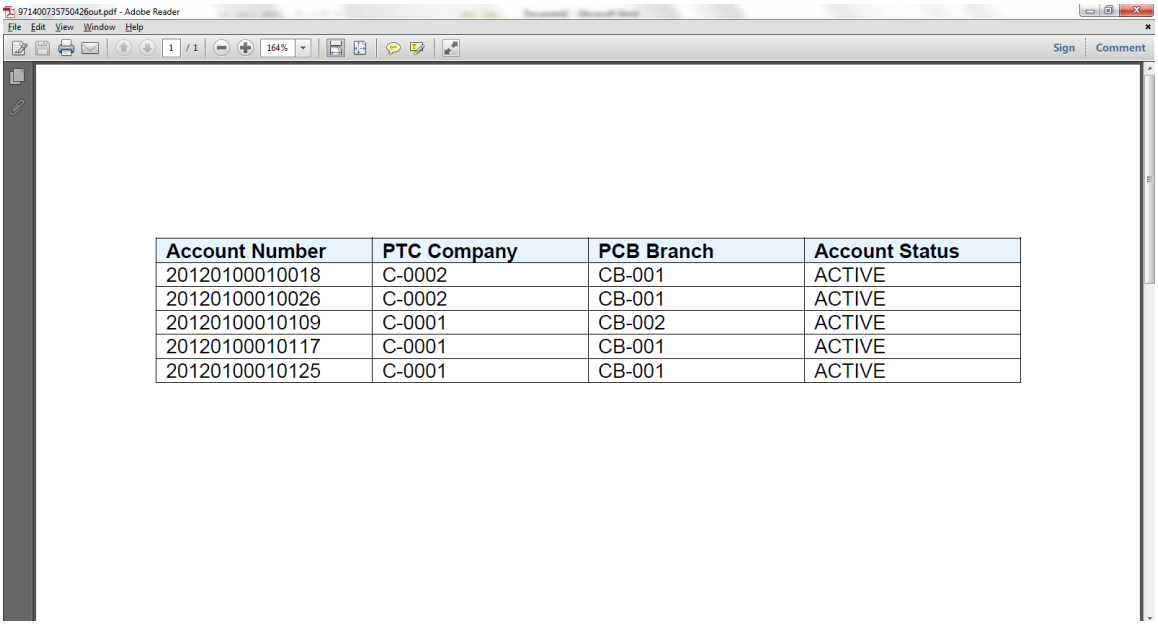

12. Save the layout with the same name as the data model and report in BIP.




13. To preview the layout, click ‘PDF’ in preview tab.




14. The report is displayed in PDF format.

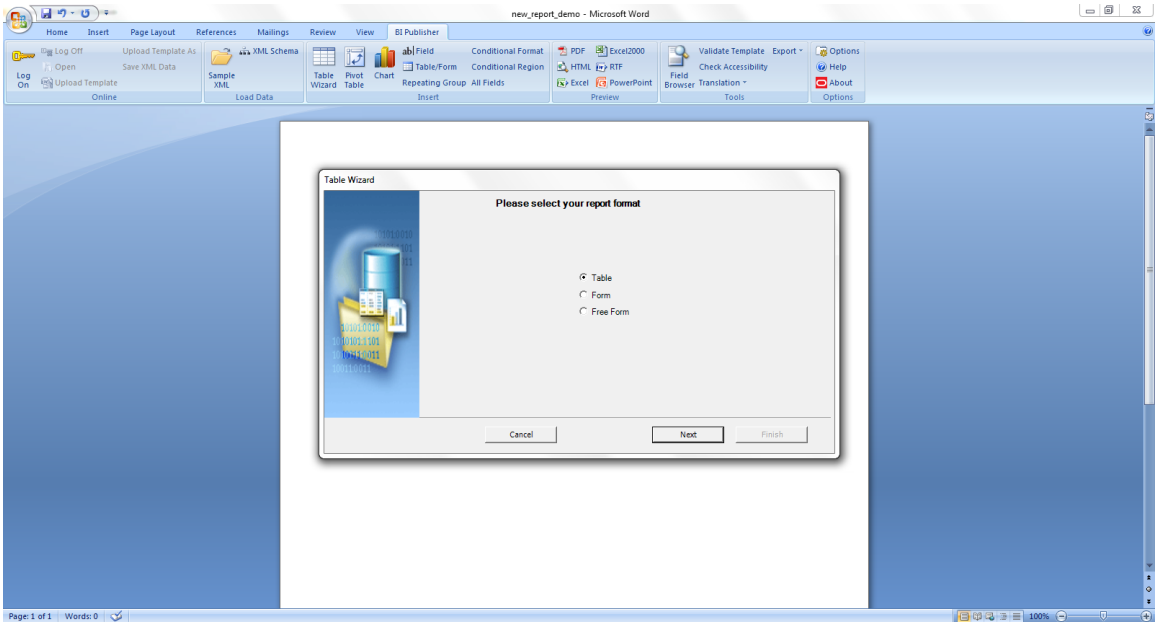



| Account Number | PTC Company | PCB Branch | Account Status |
|---|---|---|---|
| 20120100010018 | C-0002 | CB-001 | ACTIVE |
| 20120100010026 | C-0002 | CB-001 | ACTIVE |
| 20120100010109 | C-0001 | CB-002 | ACTIVE |
| 20120100010117 | C-0001 | CB-001 | ACTIVE |
| 20120100010125 | C-0001 | CB-001 | ACTIVE |

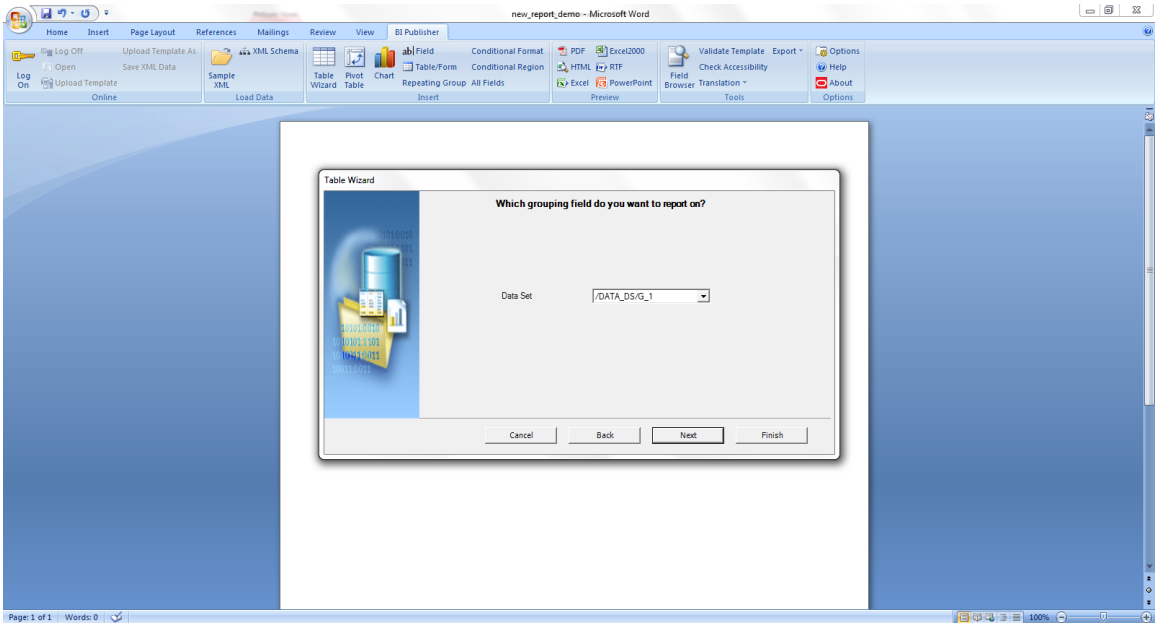
## 6.3 Add Dynamics to Report

You can use table wizard to give more dynamic report as indicated.

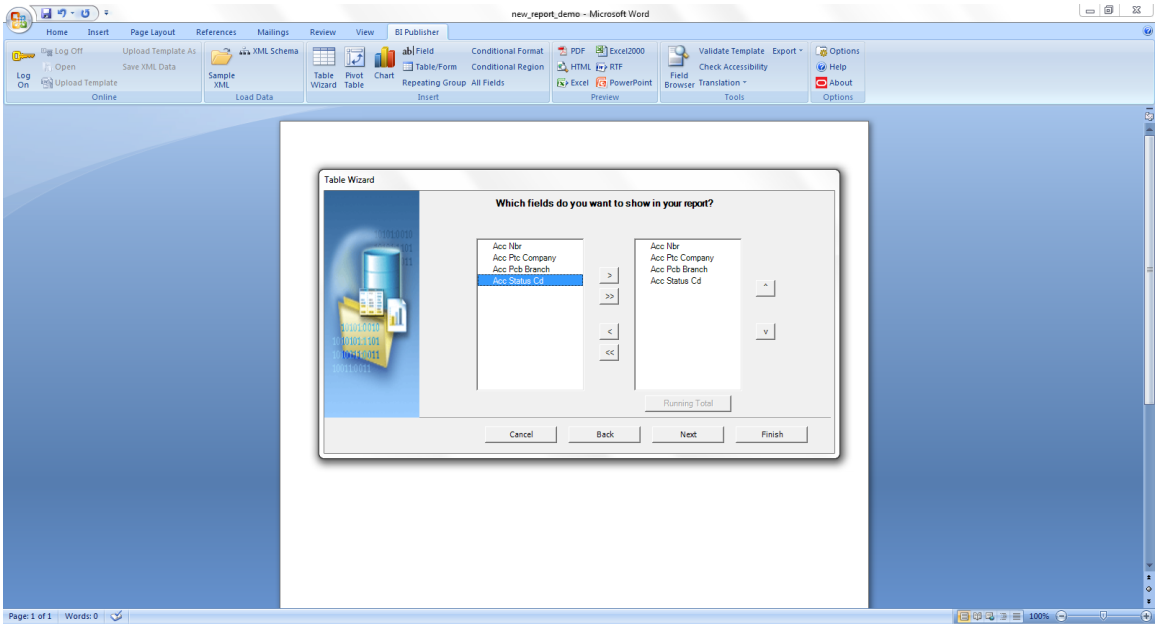
1. Select the Table Wizard from BI Publisher tab.




2. Select the Table and press ‘Next’.

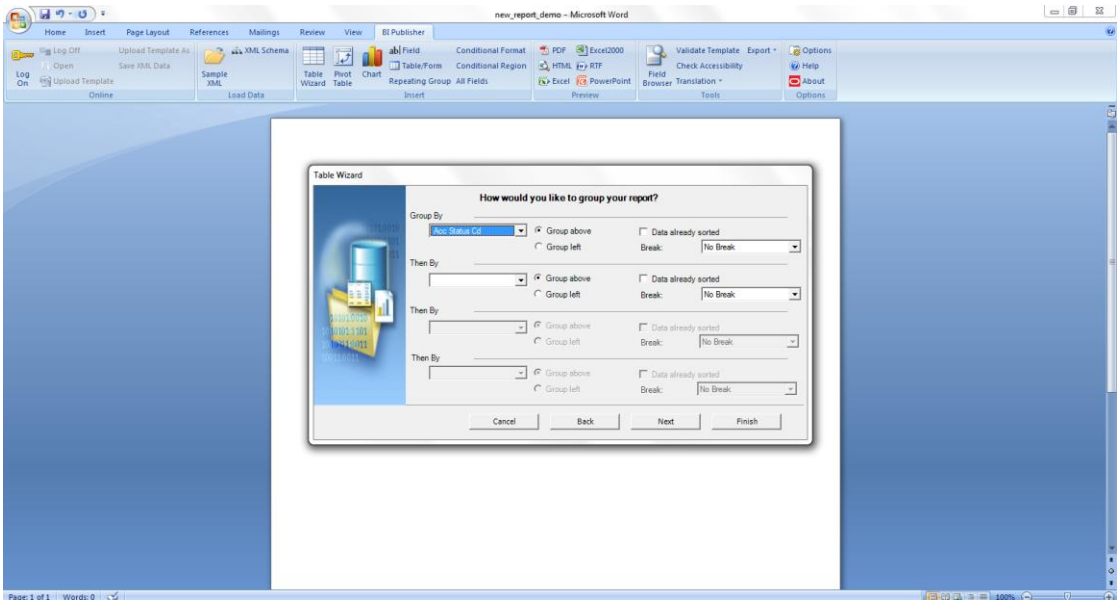



3. Use the same Data SET.

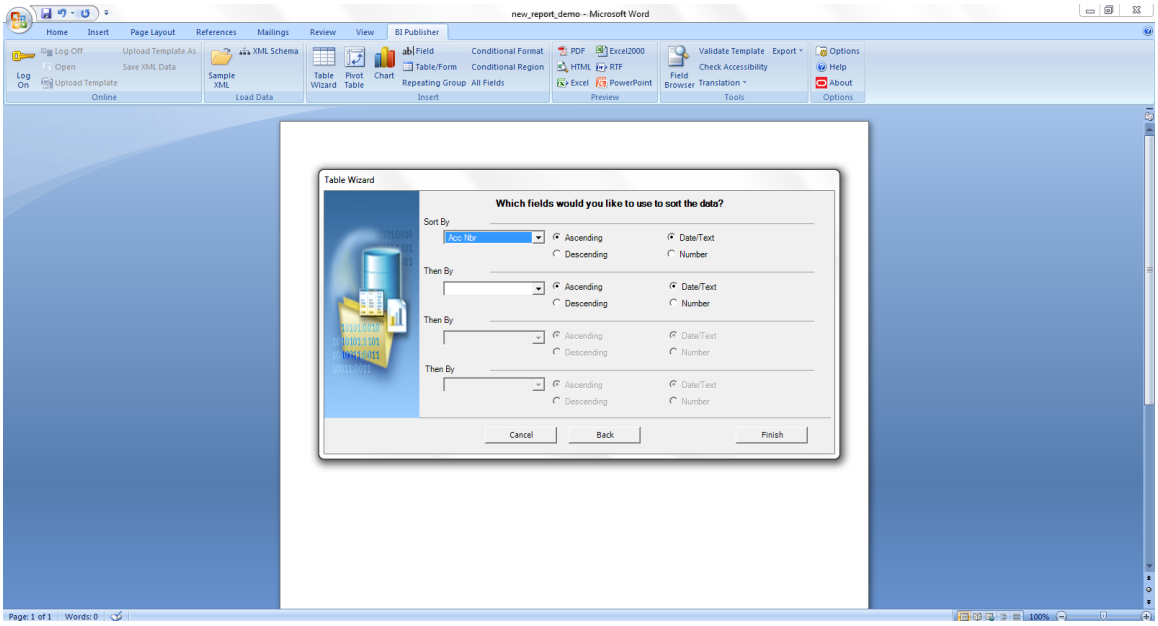



4. Select all the fields and press ‘Next’.

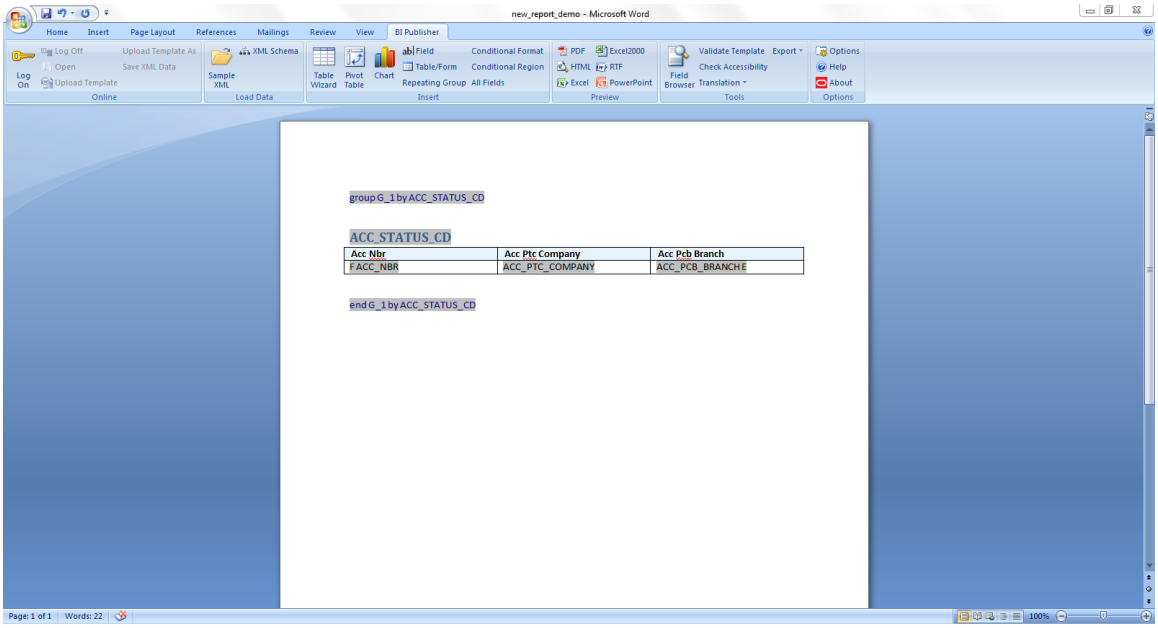
5. You can create the group as follows:



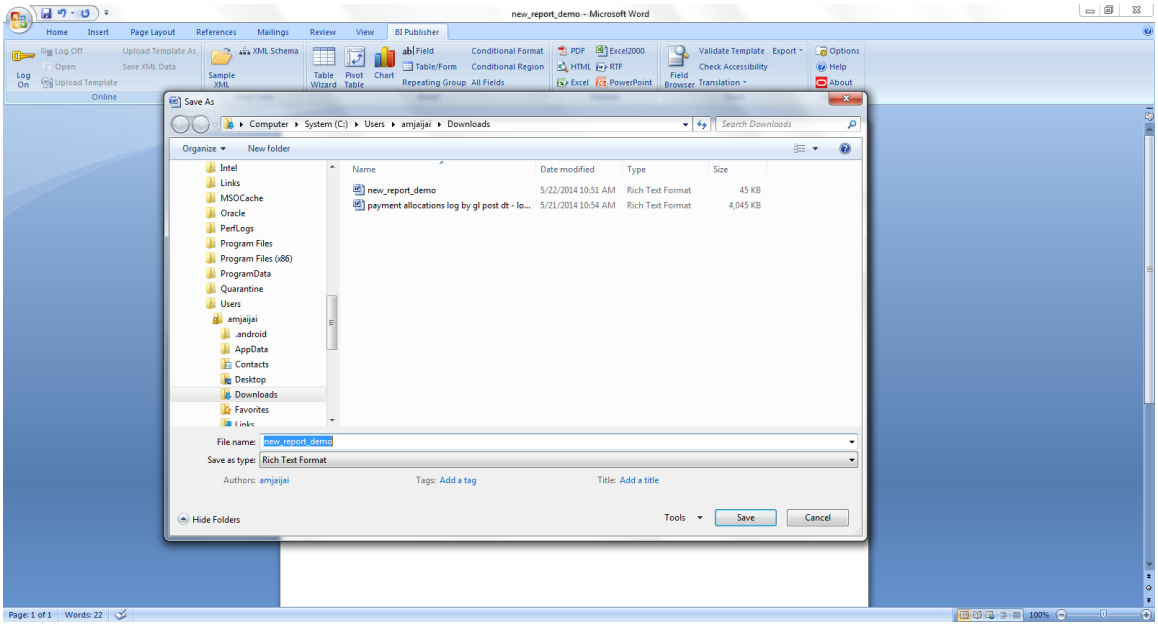

6. Click ‘Next’. You can give the sorting option as indicated.



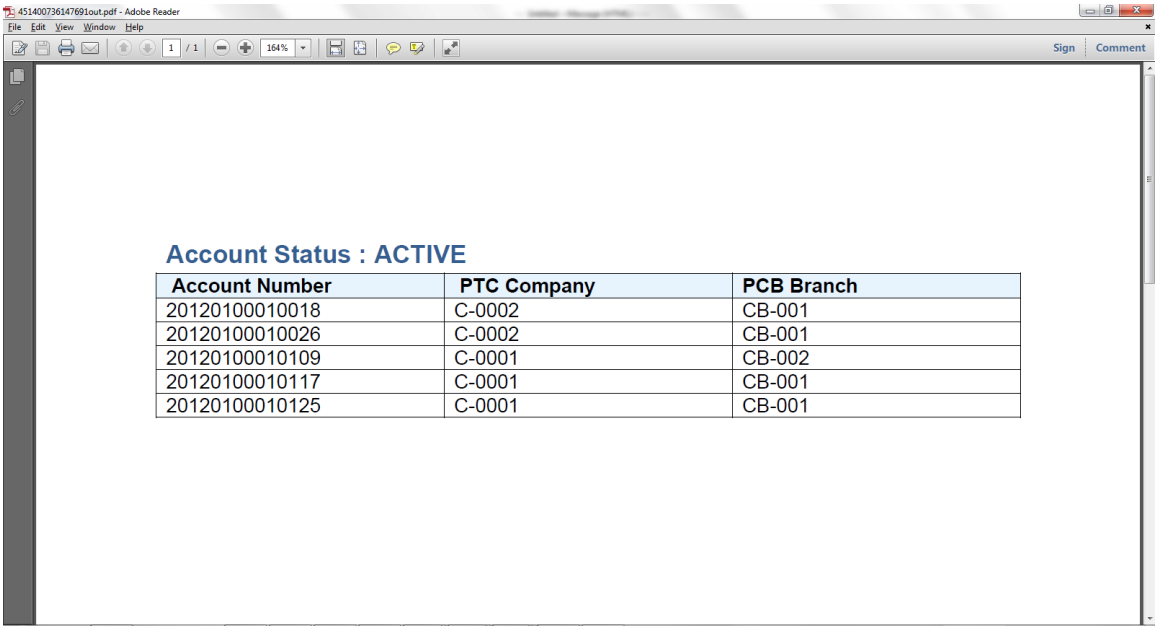

7. Click ‘Next’. The wizard creates the layout as indicated.




8. You can add,modify static Text , logo in the layout and Save the layout with the same name of datamodel or report.




9. Preview of the report layout is as indicated.

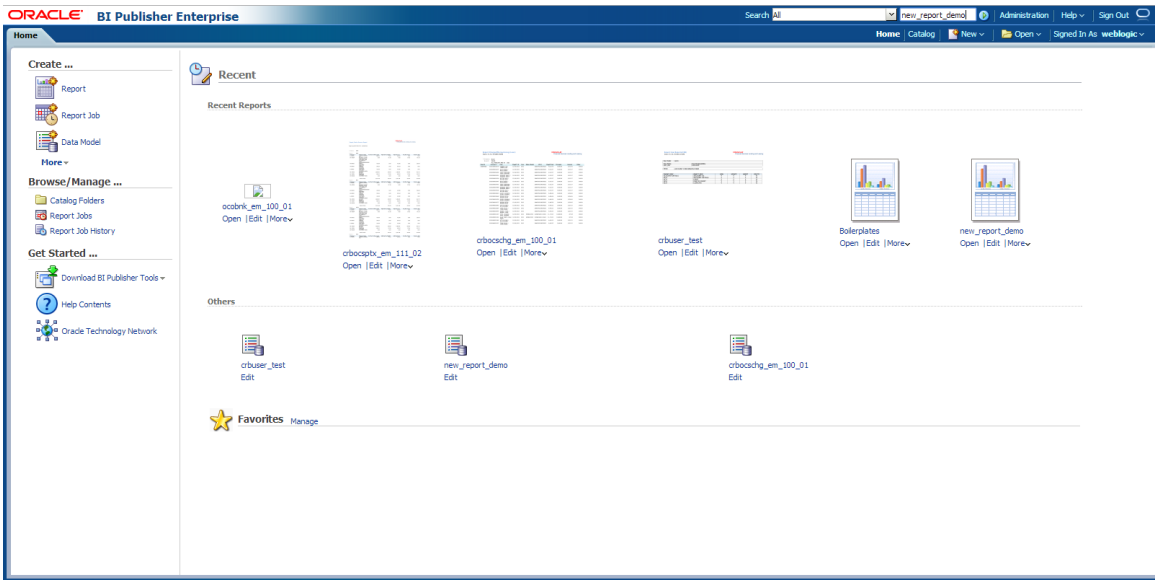


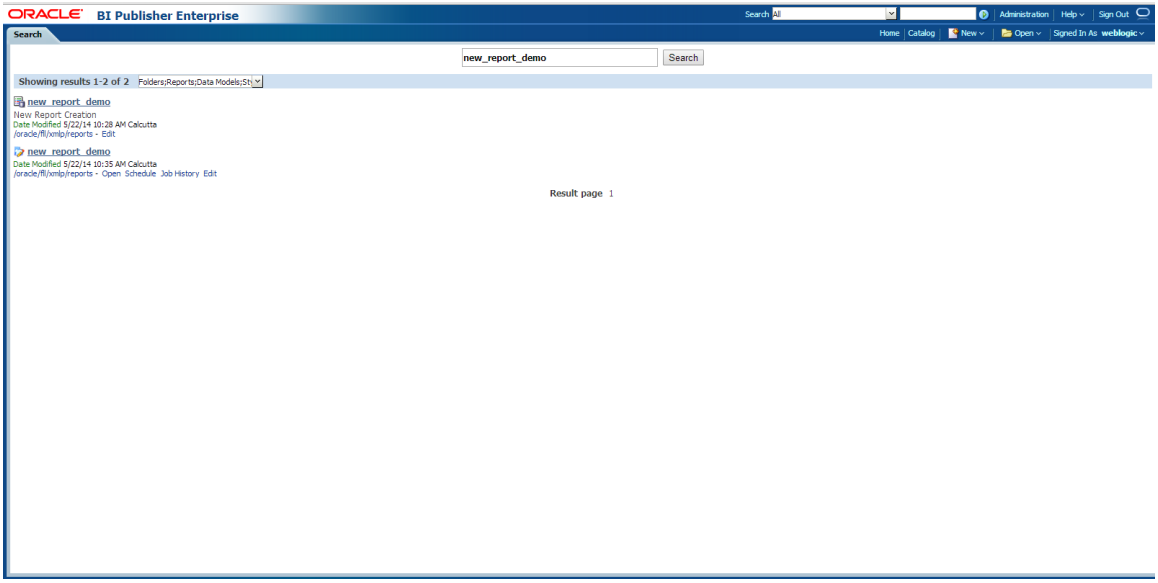

Now Layout is just created successfully and saved.

## 6.4 Upload Report in BIP

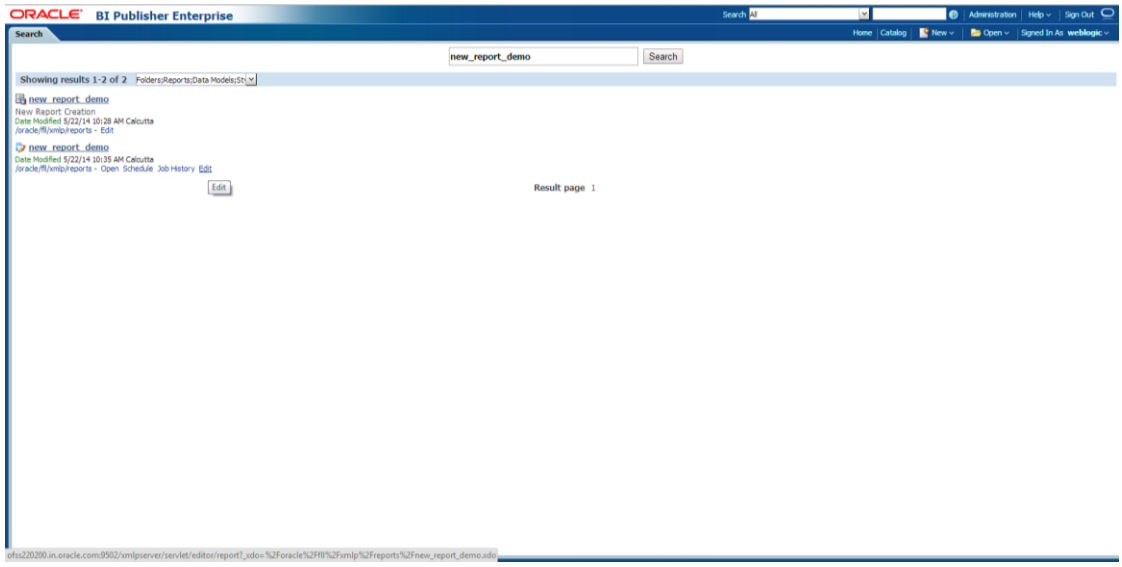
You can upload the report in BIP as indicated.

1. Navigate to BIP Console.
2. On Home page, use the search option and search for the new report created.

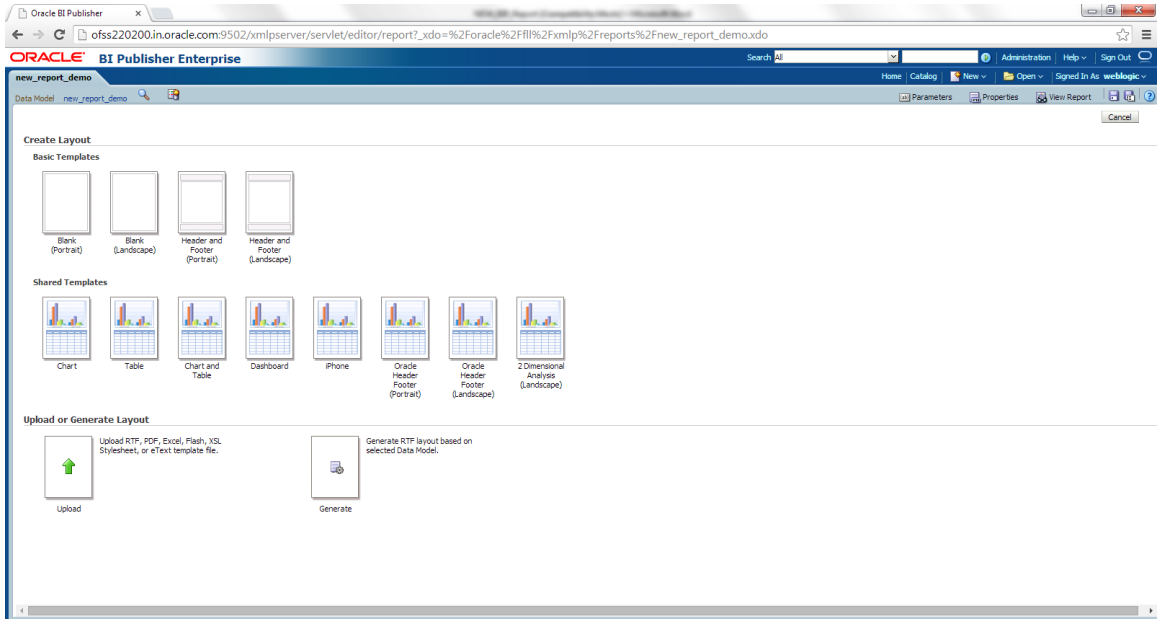



3. Click ‘Edit’ on New report layout.

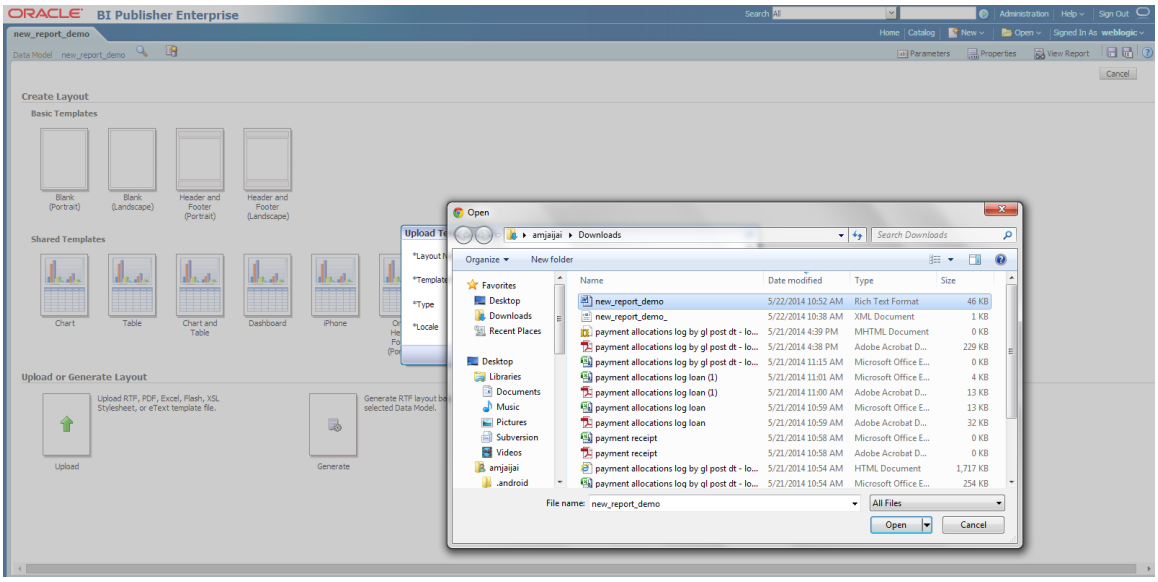



4. The following screen is displayed and allows you to upload the layout.

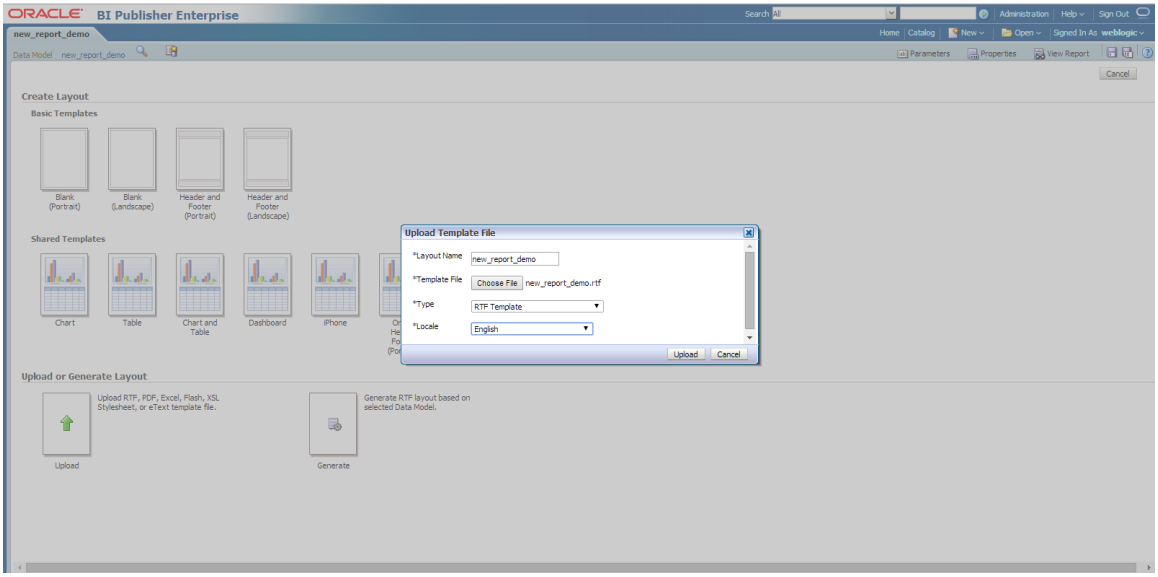



5. Click ‘Upload’.

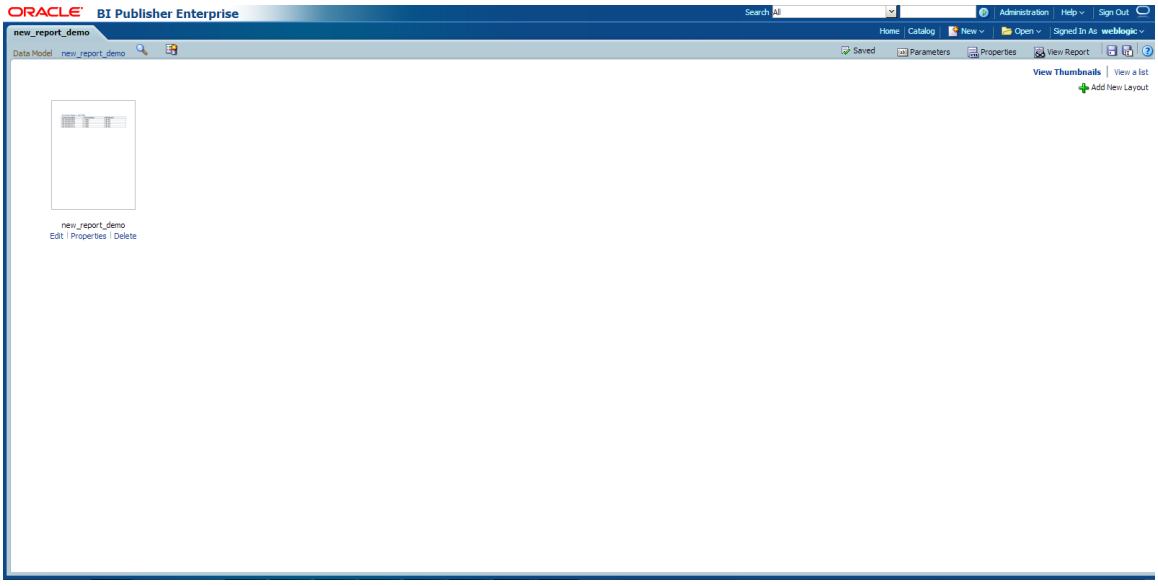
6. Specify the same layout name (new_report_demo) and choose the file.



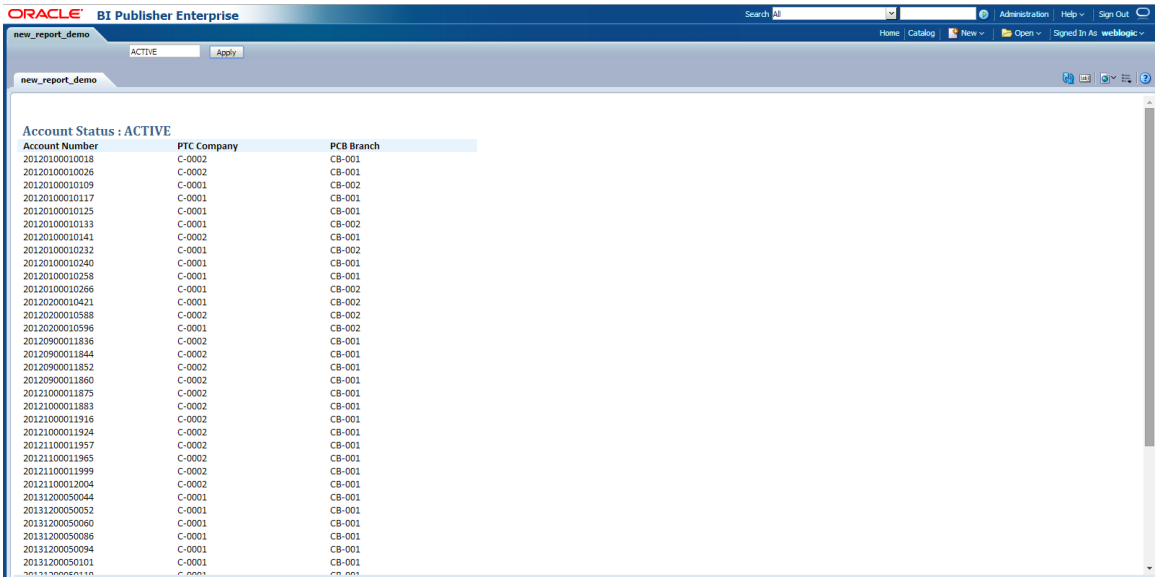

7. Select the type as ‘RTF Template’ and locale as ‘English’.




8. Click ‘Upload’.




9. Click ‘Save’. The report is created successfully.
    - Check the properties for caching and View a List for Output Format.
    - Click on View Report to verify the report.
    - Enter the parameter Value and press ‘Apply’.




**Note**; These are steps to create a new Report /Letter and then user can setup report/letter accordingly from OFSLL. Please see section 7 and 8 for the Steps to setup report and letter from UI (OFSLL).

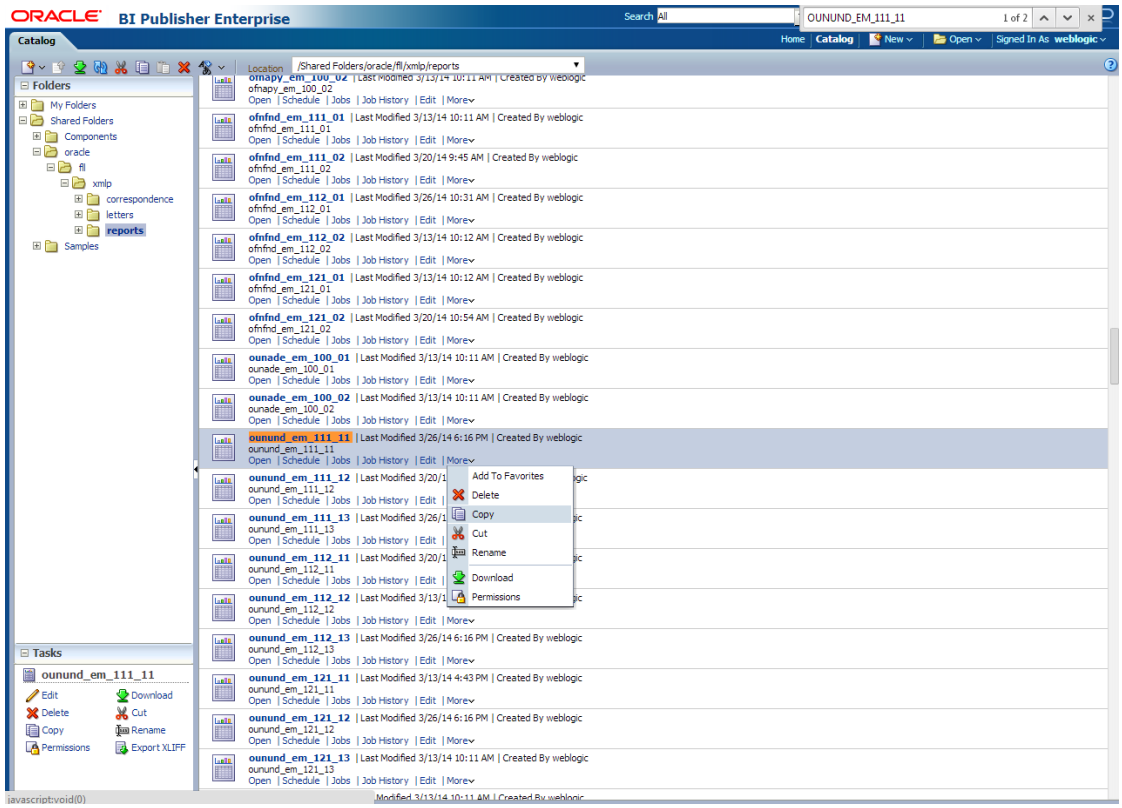
# 7. Customizing Existing Base BIP Reports

To customize a report, follow the steps given below:

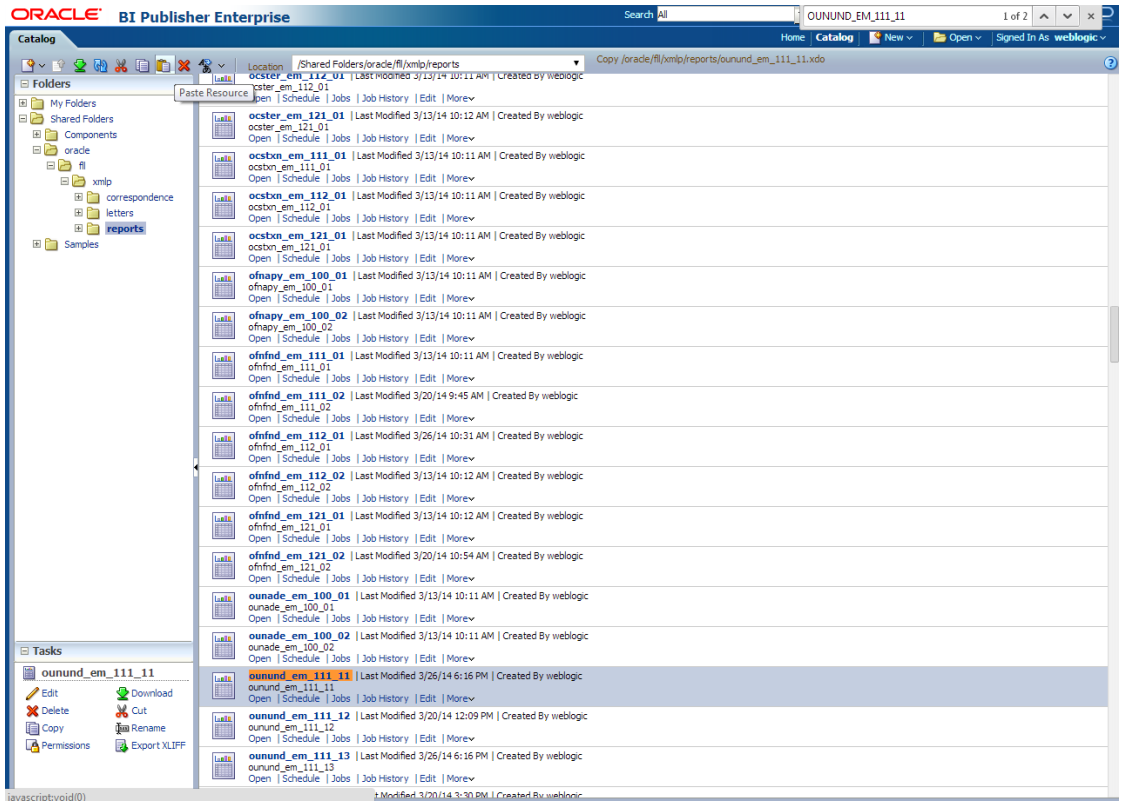
- Do not modify the OFSLL Base Report.
- Create a copy of the OFSLL Base Report, rename and modify that report. A new report also can be created. Name the report as **xyz**_<report_name>.
- Register the new report and it's parameters in OFSLL using reports setup.

**Pre-Requisites**

1. Please make sure that we should not change any base reports
2. The reports should be placed into the same folder structure
   - i.e. For Reports → Shared Folders/oracle/fll/xmlp/reports
   - For Letters → Shared Folders/oracle/fll/xmlp/letters
   - For Correspondences → Shared Folders/oracle/fll/xmlp/correspondence
3. Consider a base report OUNUND_EM_111_11 (UNDERWRITING STATUS BY MONTH AND PRODUCER LOAN)
4. Let us assume we will do some customizations on the base report and create a new report called XYZOUNUND_EM_111_11 (Here XYZ is bank code)
5. Search for the base report and press ***More → Copy*** as shown in the image below.

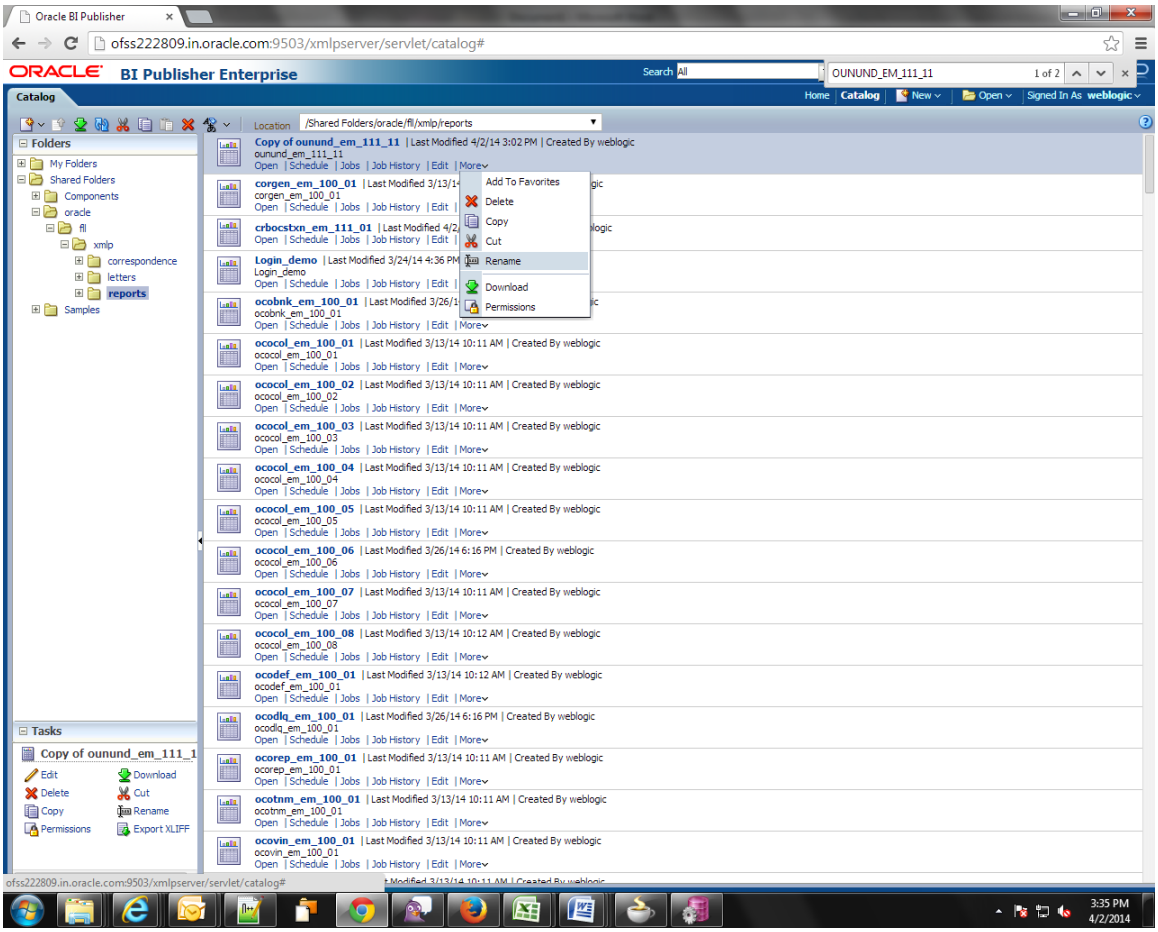


6. After pressing copy go to the folder where you want to paste the new report and press the ***Paste Resource*** button as shown in the image below

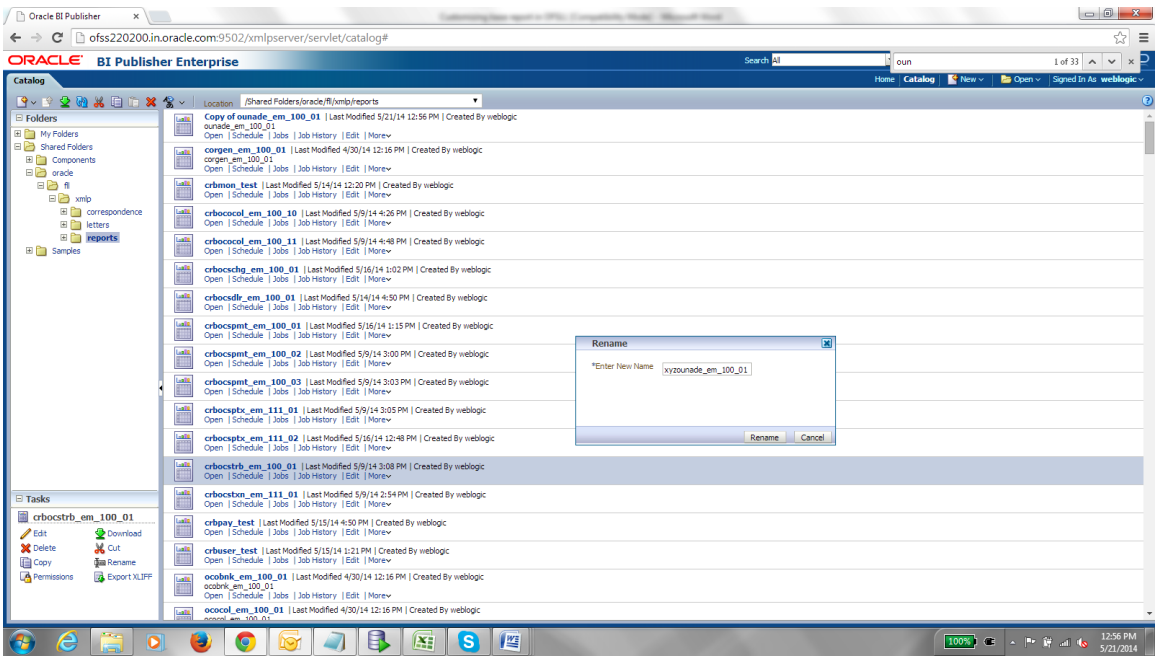



7. On pressing ***Paste Resource*** button, a new report will be created in the directory with name as ***Copy of ounund_em_111_11***

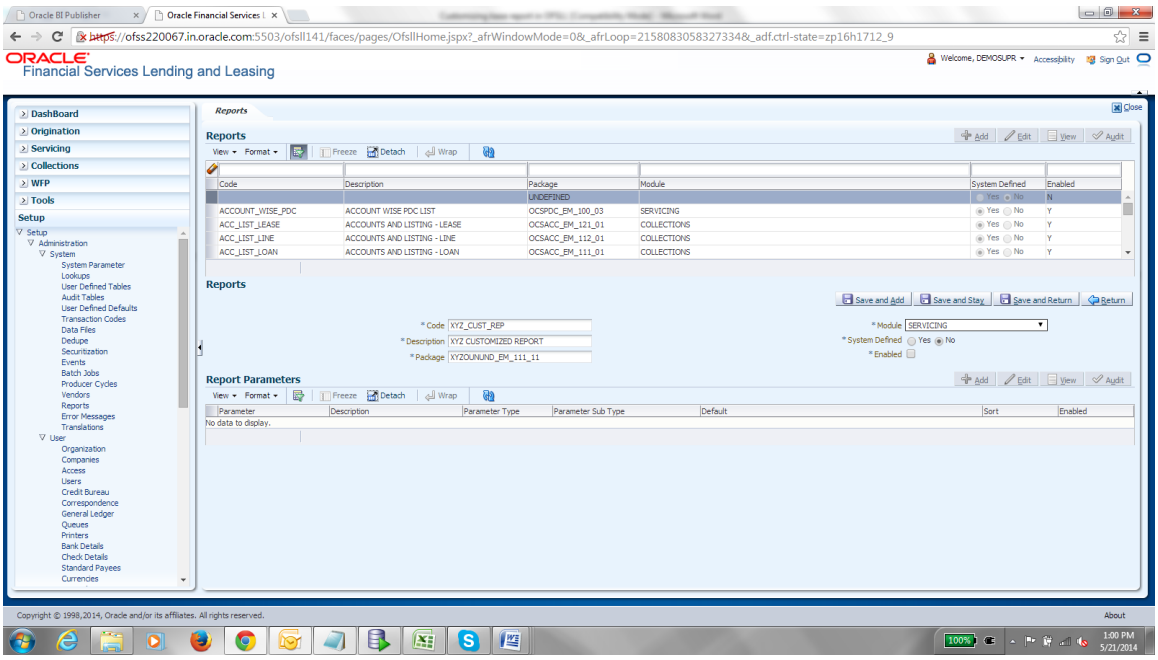
8. Select the particular report (***Copy of ounund_em_111_11 )*** and press ***More→ Rename*** as shown in the image below



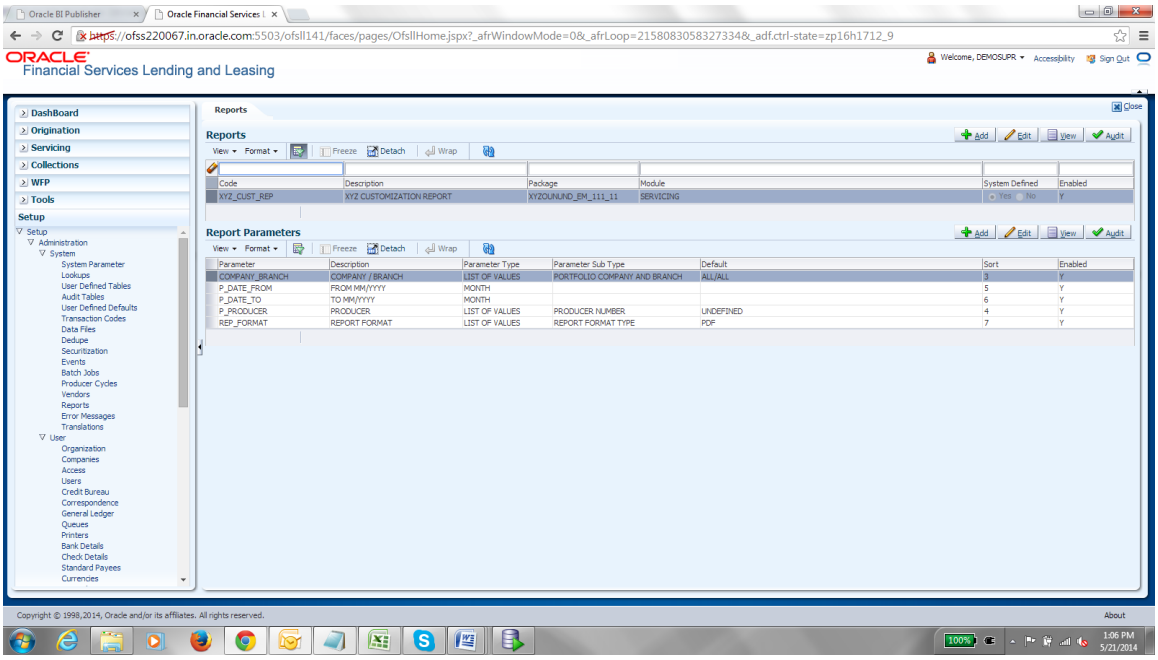

9. Enter the new name as xyz**ounund_em_111_11** and press ***Rename*** button as shown in the image below



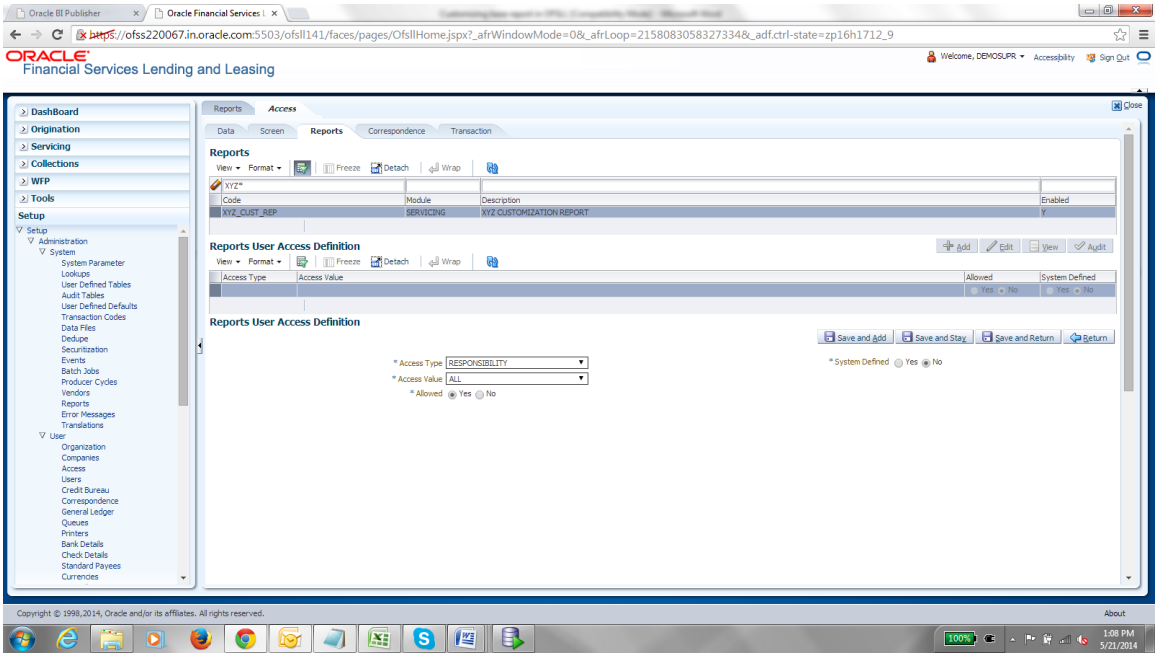
10. Do similar Copy, Paste and rename activity for the data model side also for the particular report which will be with the same name as of the report in the same directory.
11. After the new report is ready i.e. xyz**ounund_em_111_11**. We can now do our customizations whatever is required on this new report.
12. So after Completing from bi publisher side we need to make an entry in the database for the new report to be available in the front end application
13. Go to Setup→ System → Reports in the front end and add a new record with package name as XYZOUNUND_EM_111_11 as shown in the image below

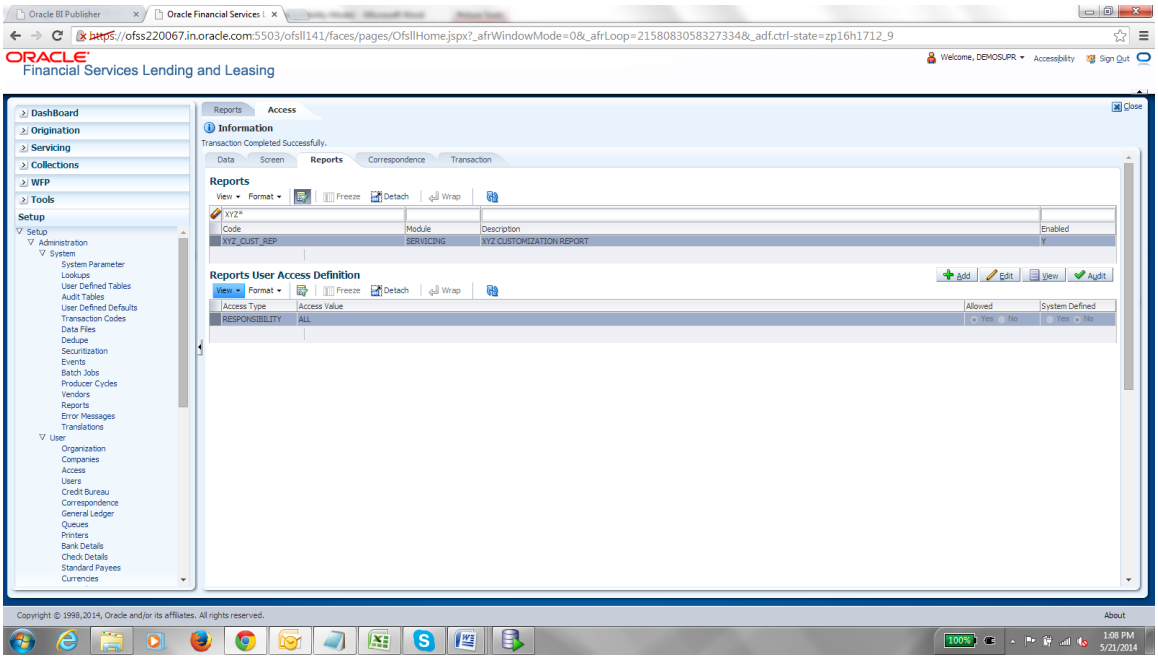



14. After Save and return please add the report parameters for the particular report as shown in the image below.

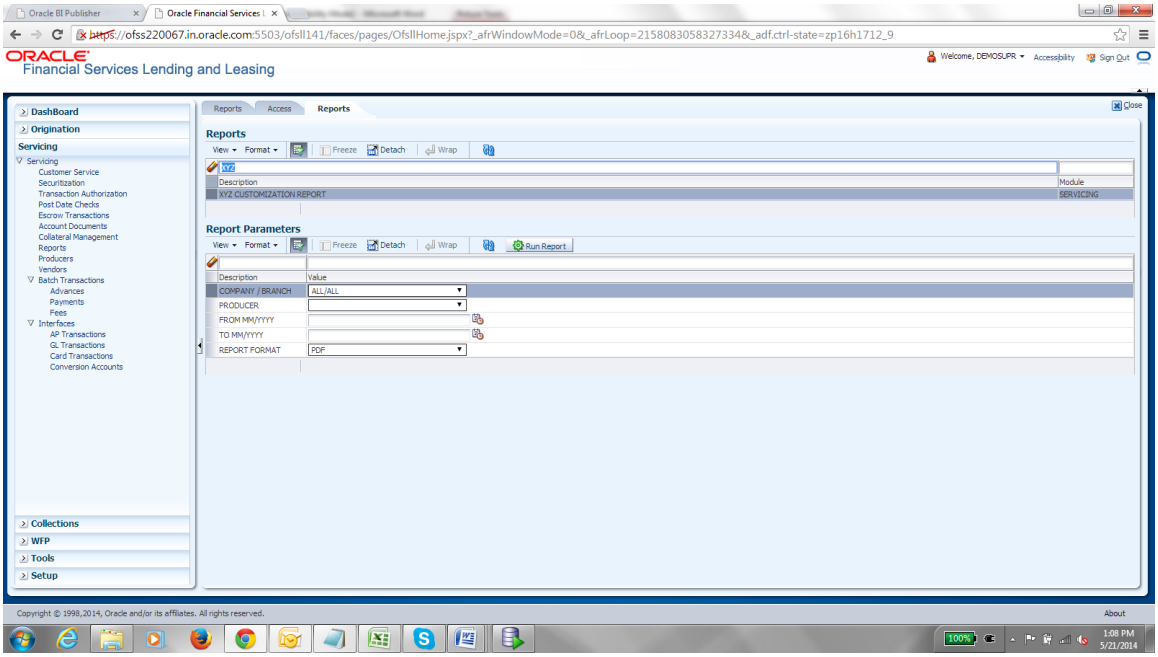



15. Now open ***User→Access→Reports*** Tab and select the newly created report and add the responsibility to it in Reports user access definition screen as shown in the image below






16. Now go to ***Servicing → Reports*** screen and you will be able to find the new report in the list as shown in the image below

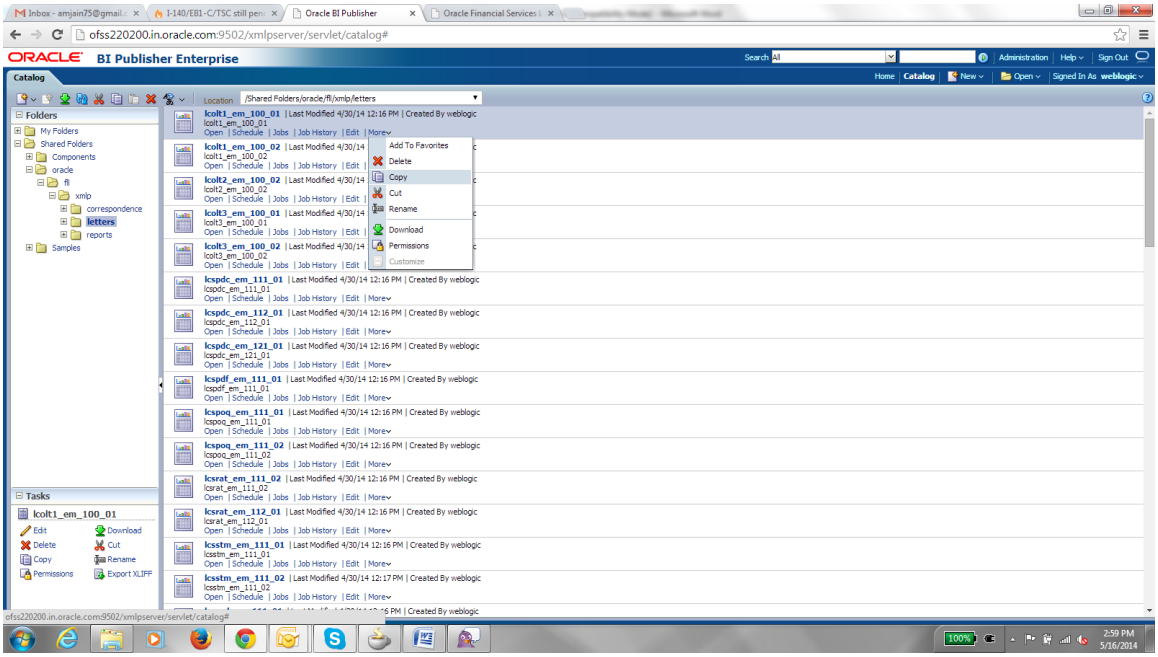



**Note:** There is no Impact on customization reports when a new base patch applied in the system. All customized report will not be override, removed or modified.

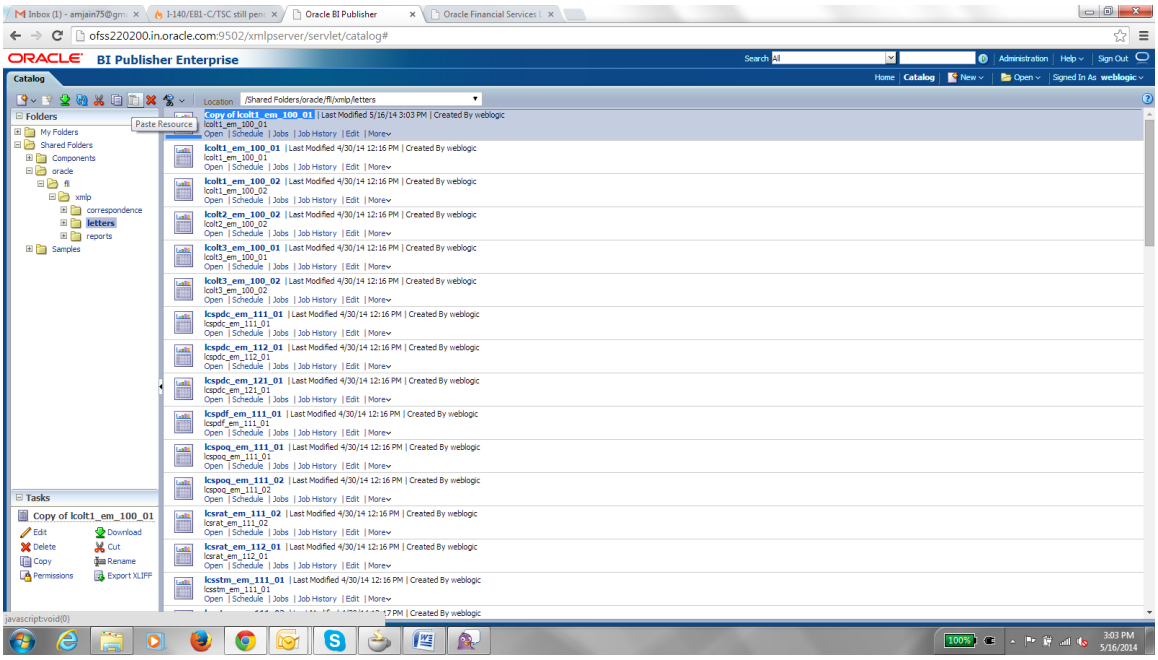
# 8. Customizing Existing Base BIP Letters

**Pre-Requisites**

1. Please make sure that we should not change any base letters
2. The letters should be placed into the same folder structure
   - For Letters → Shared Folders/oracle/fll/xmlp/letters
3. Consider a base report lcolt1_em_100_01 (Collection Letter)
4. Let us assume we will do some customizations on the base report and create a new report called xyzlcolt1_em_100_01(Here XYZ is bank /customer code)
5. Search for the base letter and press ***More → Copy*** as shown in the image below

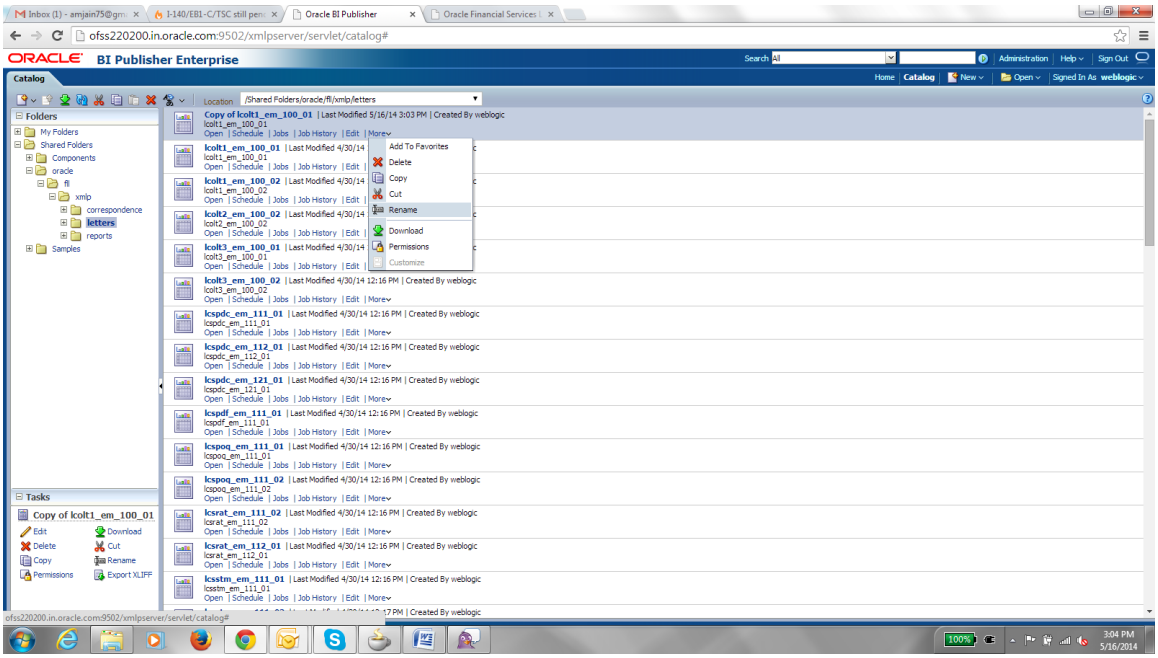



6. After pressing copy go to the folder where you want to paste the new report and press the ***Paste Resource*** button as shown in the image below.

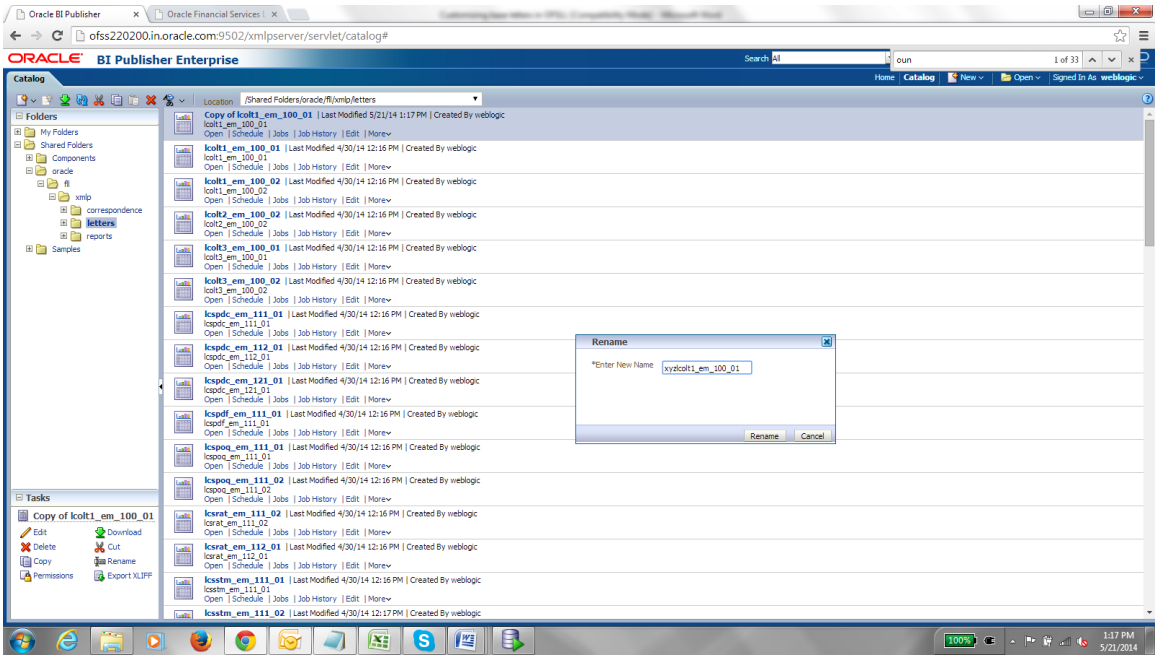



7. On pressing ***Paste Resource*** button, a new report will be created in the directory with name as ***Copy of lcolt1_em_100_01***

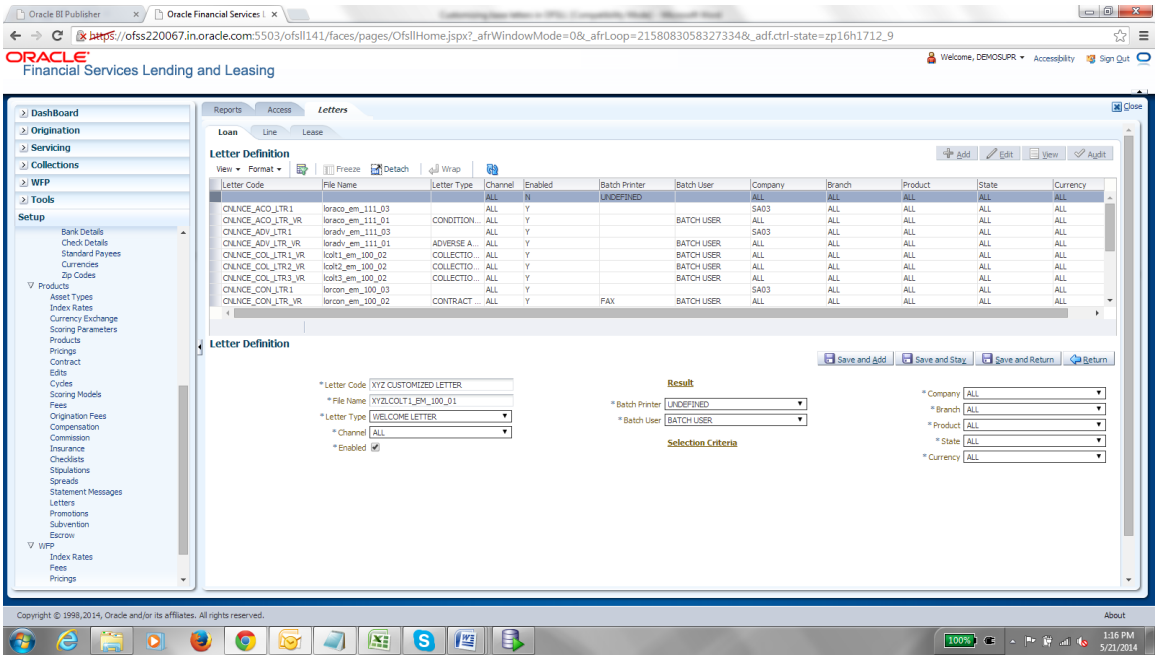
8. Select the particular report (***Copy of lcolt1_em_100_01)*** and press ***More→Rename*** as shown in the image below




9. Enter the new name as xyz**lcolt1_em_100_01** and press ***Rename*** button as shown in the image below.




10. Do similar Copy, Paste and rename activity for the data model side also for the particular report which will be with the same name as of the report in the same directory.
11. After the new report is ready i.e. xyzlcolt1_em_100_01. We can now do our customizations whatever is required on this new report.
12. So after Completing from bi publisher side we need to make an entry in the database for the new report to be available in the front end application
13. Go to Setup→ Products → Letters in the front end and add a new record with package name as xyzlcolt1_em_100_01 as shown in the image below , Only one letter can be saved only for following combination
    - Letter Type
    - Company
    - Branch
    - Product
    - State
    - Currency




14. Save and return.

**Note:** There is no Impact on customization letters when a new base patch applied in the system. All customized letters will not be override , removed or modified.

# 9. Create Custom Correspondence

The Correspondence screen enables you to define who will receive the documents you created on the Document Definition page by creating correspondence sets. Each document must belong to a set, and a set can have more than one document.

You can set up the various documents and the data fields that the system compiles together when creating a correspondence. The system provides two different document formats: Word or XFDF: XML-based format.

---

**Note**

Oracle Financial Services Software assumes that the user is familiar with Word and the Merge Document command. If the user is creating e-form documents with XFDF, then Oracle Financial Services Software assumes that person is familiar with Adobe forms.

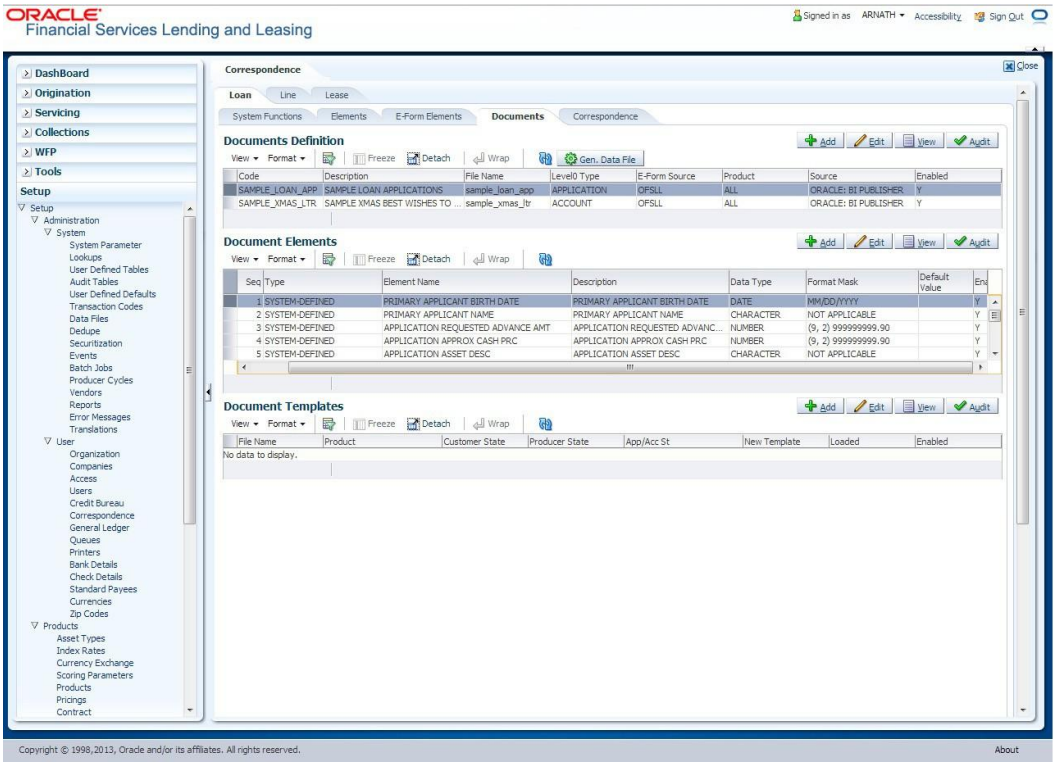
---

**To create a Correspondence**

1. On the Oracle Financial Services Lending and Leasing home page, click **Setup→Setup→Administration→User→Correspondence→Loan/Line/ Lease→Documents**
2. In the Document definition block, add a record. *For example:* SAMPLE_LOAN_APP A brief description is given below:

| Field: | Do this: |
|---|---|
| Code | Specify the document code to define the name for the new document. |
| Description | Specify the document description for the new document. This entry appears in the **Correspondence** section on the Request page, when you generate an ad hoc correspondence. |
| File Name | Specify the document file name for the resulting file (Word or XFDF document). |
| Level0 Type | Select the level0 type from the drop-down list. |
| E-form Source | Select the element e-form source from the drop-down list. |
| Product | Select the document product from the drop-down list. |
| Source | Select the document source type from the drop-down list. |
| Enabled | Check this box to enable the document definition. |

3. In the **Document Elements** section, add the elements required in the correspondence.

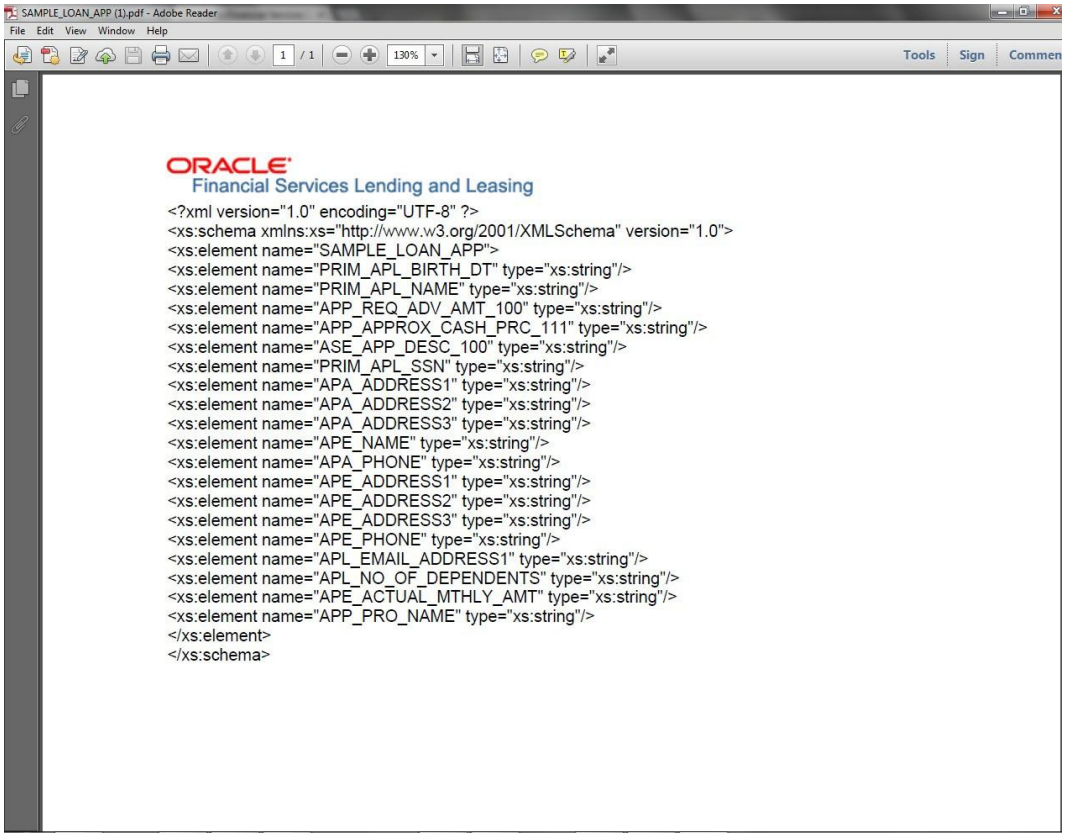



A brief description of the fields are given below:

| **Field:** | **Do this:** |
|---|---|
| Seq | Specify the sequence number to order the document elements. |
| Type | Select element type from the following from the drop-down list. This list provides the following options:<br><br>*System-defined* – If you select, the value is supplied by the system and cannot be changed in the Correspondence Request page.<br><br>*Constant.*<br><br>*User Defined Element* – If you select, you can choose the value and change it in the Correspondence Request screen.<br><br>*User Defined Constant* – If you choose, you can choose the value, but you cannot change it in the Correspondence Request screen.<br><br>*Translated Element* – If a document contains an e-form element and you do not select this option, then the value will not be translated. |
| Element Name | Select the element name from the drop-down list. |

| Field: | Do this: |
|---|---|
| Description | Specify element description.<br><br>**Notes**:<br><br>1. Check that the element name does not have blank spaces or special characters, such as the forward slash “/” or backward slash “\”.<br><br>2. If th element is system-defined, then the system will automatically complete this field. |
| Data Type | Select the element data type from the drop-down list. |
| Format Mask | Select the element format mask from the drop-down list. |
| Default Value | Specify the element default value. |
| Enabled | Check this box to include the element in the document. |

4. Click on Gen.Data File to generate PDF file of the report.




5. Copy and save the content in the pdf file as an xml file. The saved xml file should have the same name as entered in the Code column of Document Definition section. *For Example:* SAMPLE_LOAN_APP.xml.

6. Open MS Word.

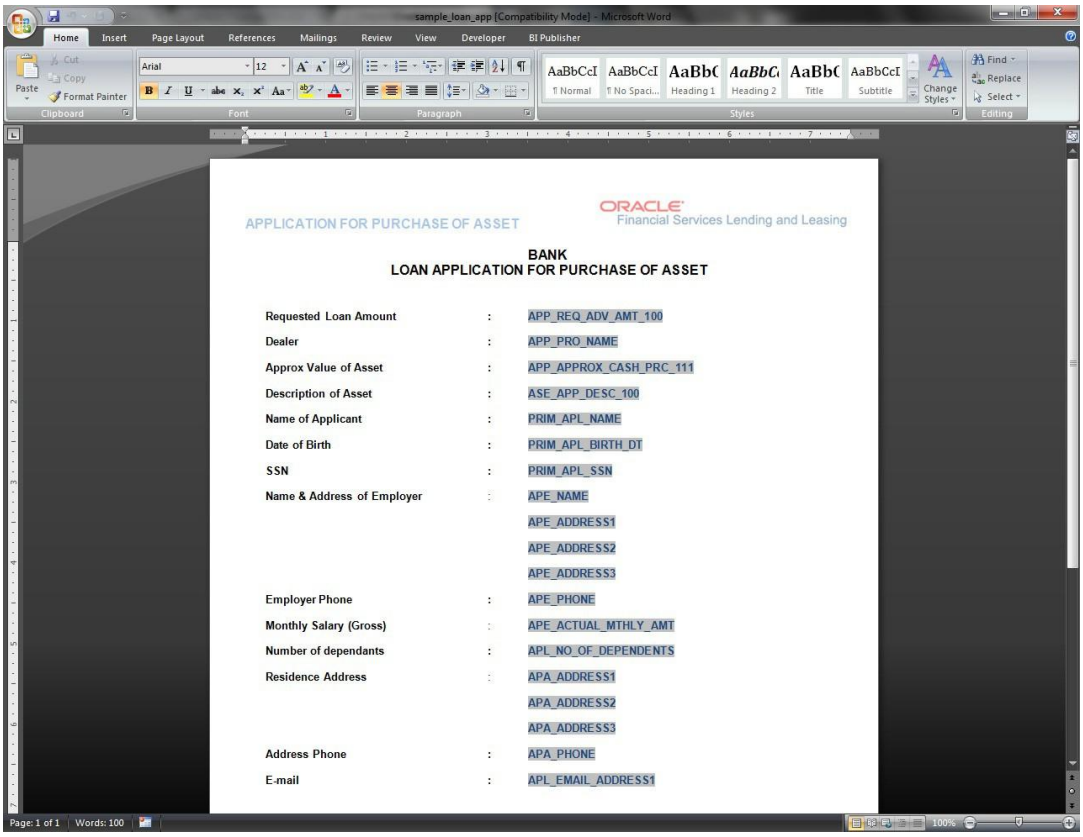
---

**Note**

Oracle Financial Services Software assumes that BIP Desktop Tool is installed and the user is familiar with the BIP Report Tool.

---

7. In BI Publisher Tab in MS Word, click on Sample XML and import the saved xml file. *For Example:* SAMPLE_LOAN_APP.xml.
8. Create the template by inserting required elements tag.



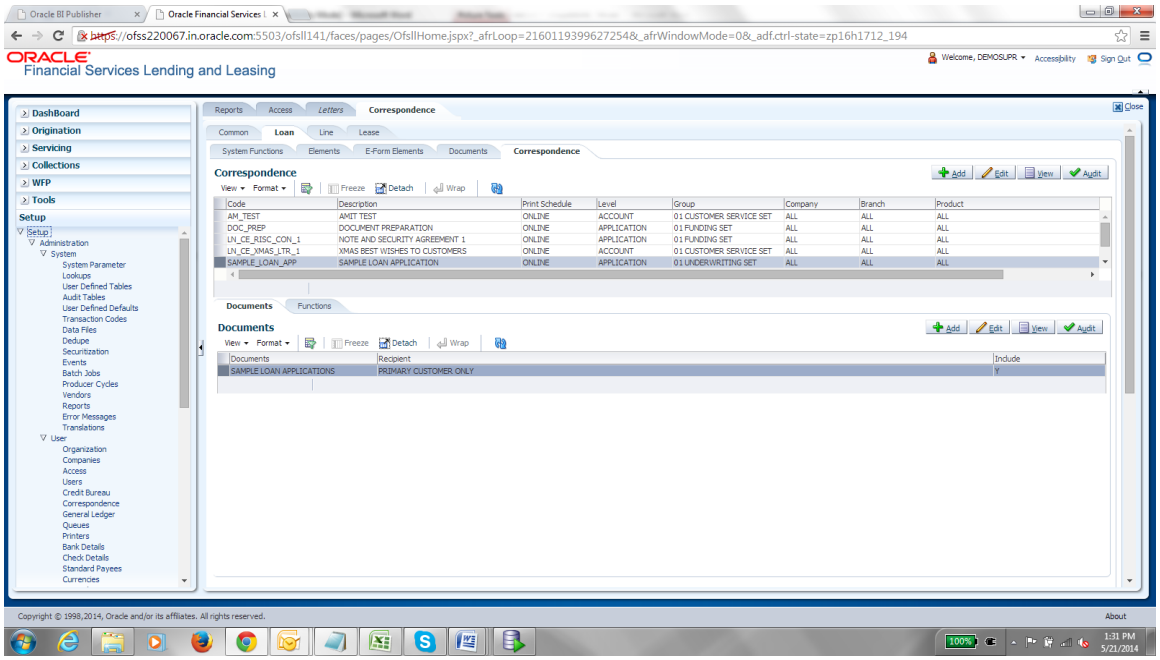

9. The template created in MS Word should be saved with **.rtf** extension. *For Example*: SAMPLE_LOAN_APP.rtf

---

**Note:** The **.xml** and **.rtf** file should be saved with the same name as entered in the 'Code' column of Document Definition section.

---

10. Upload the rtf template in the BIP and create the data model with SQL query as “select CDO_XML_DOCUMENT from correspondence_docs where cdo_id = :docId”.
11. After the data model creation, launch the correspondence screen and click Correspondence tab.
12. You can setup a correspondence with the created doc.

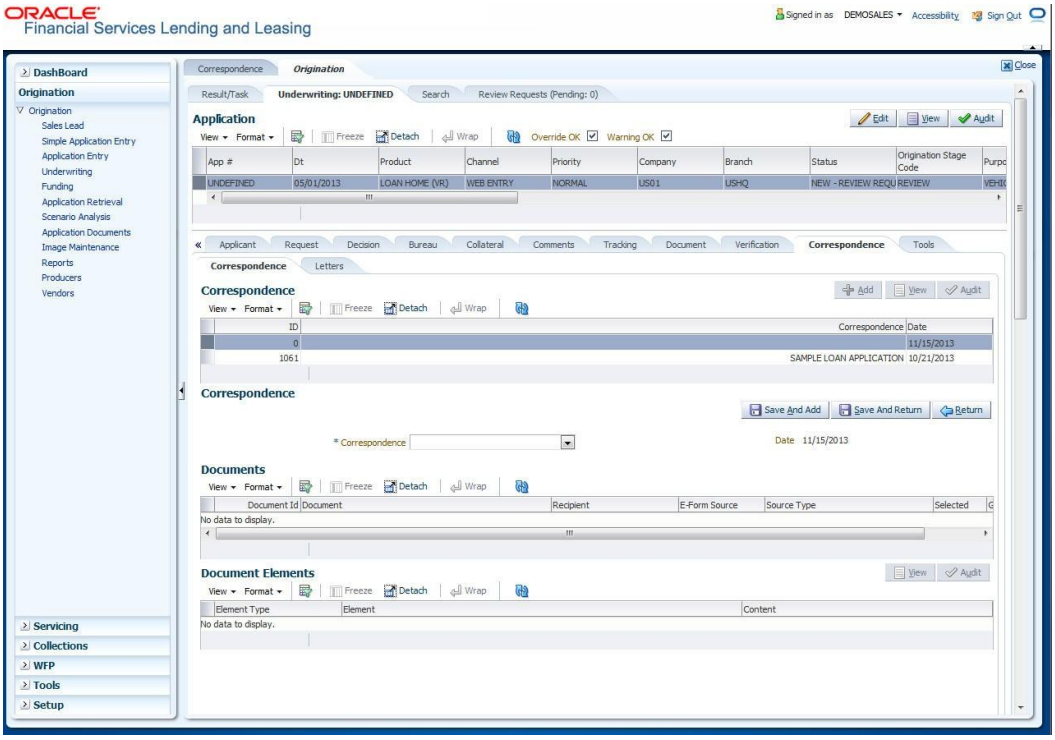


# 10. Generating Correspondence

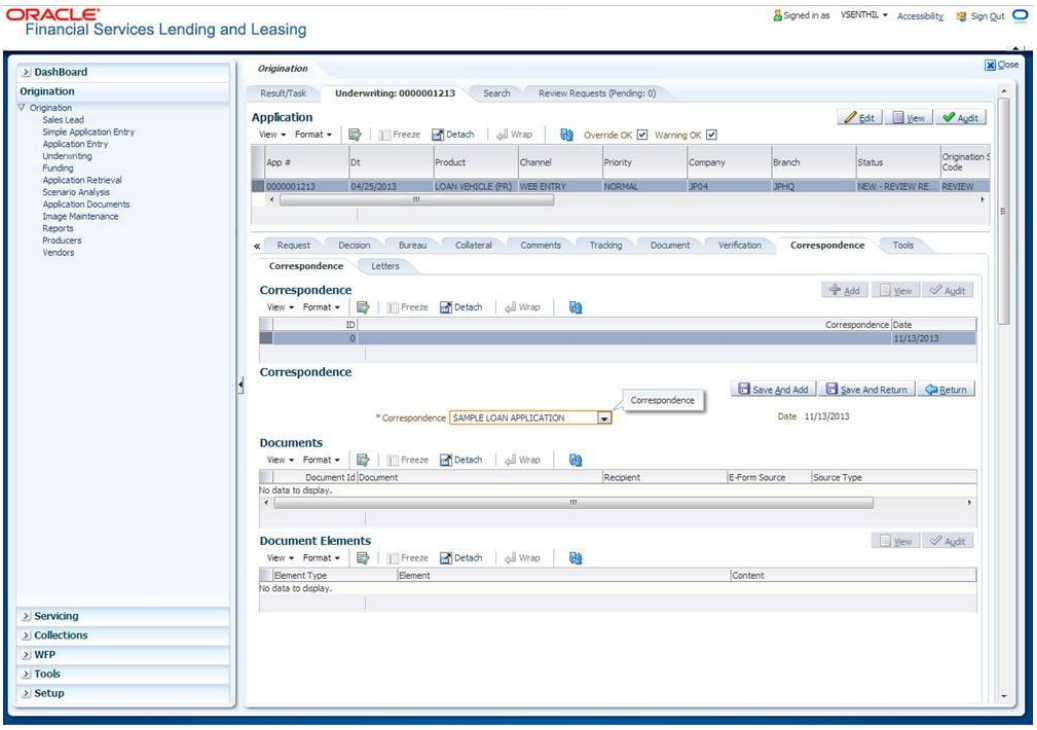
You can generate a correspondence once the respective correspondence is created in the database.

**To generate a Correspondence**

1. On the Oracle Financial Services Lending and Leasing home page, click **Origination**→ **Origination**→ **Underwriting**
2. Open the application for which the correspondence should be generated.
3. Click **Correspondence** tab. In the **Correspondence** section, click on **Add.**




4. Select the created **Correspondence.** Click **Save and Add** to save and add a new record.
5. Click to **Save and Return** save and return to main screen. Click **Return** to return to main screen without modifications.




6. Click **Generate** to generate the selected correspondence and **View Correspondence** to view the Correspondence in PDF format.

APPLICATION FOR PURCHASE OF ASSET

ORACLE
Financial Services Lending and Leasing

**BANK**
**LOAN APPLICATION FOR PURCHASE OF ASSET**

| | | |
|---|---|---|
| **Requested Loan Amount** | : | 20000.00 |
| **Dealer** | : | |
| **Approx Value of Asset** | : | .00 |
| **Description of Asset** | : | 2005 TOYOTA CAMRY |
| **Name of Applicant** | : | ANDREW WATT |
| **Date of Birth** | : | 07/15/1975 |
| **SSN** | : | XXXXX2147 |
| **Name & Address of Employer** | : | |
| | | 58, EAST 19TH STREET |
| | | HOLTSVILLE NY 00544 |
| **Employer Phone** | : | 0 |
| **Monthly Salary (Gross)** | : | 552230.00 |
| **Number of dependants** | : | 0 |
| **Residence Address** | : | 34, WEST 69TH ST N BCH N |
| | | NEW YORK MA 01730 US |
| **Address Phone** | : | 0 |
| **E-mail** | : | ANDREW.WATT@XYZ.COM |

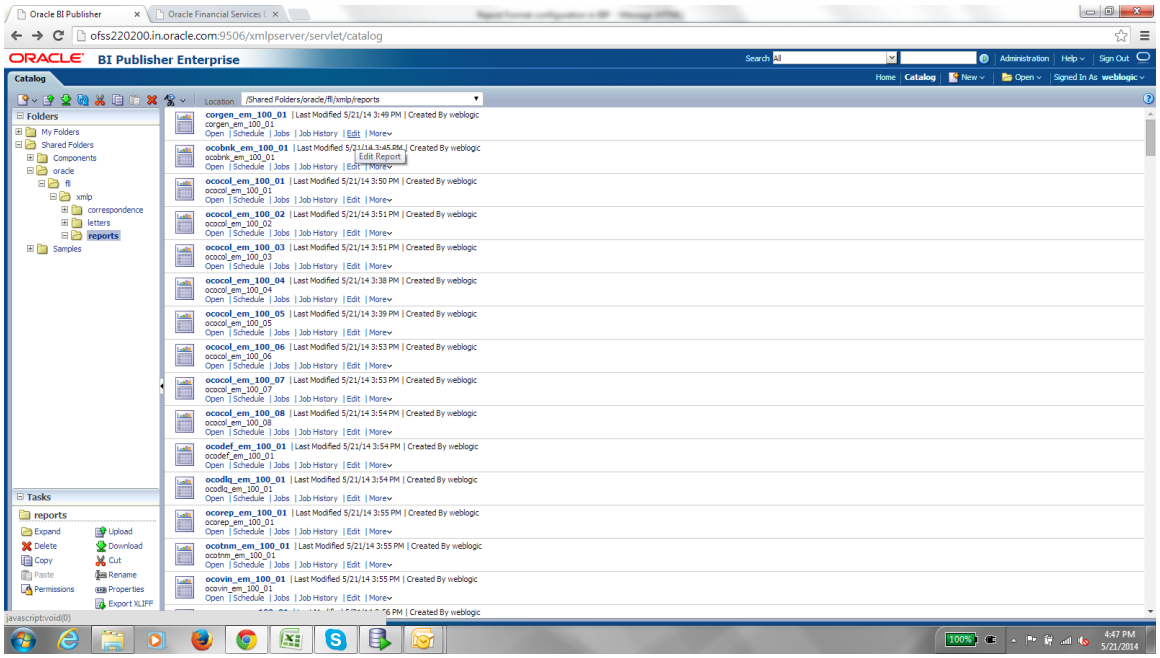
I declare that the information given in the application is true to the best of my knowledge and belief

Signature of the Applicant _____

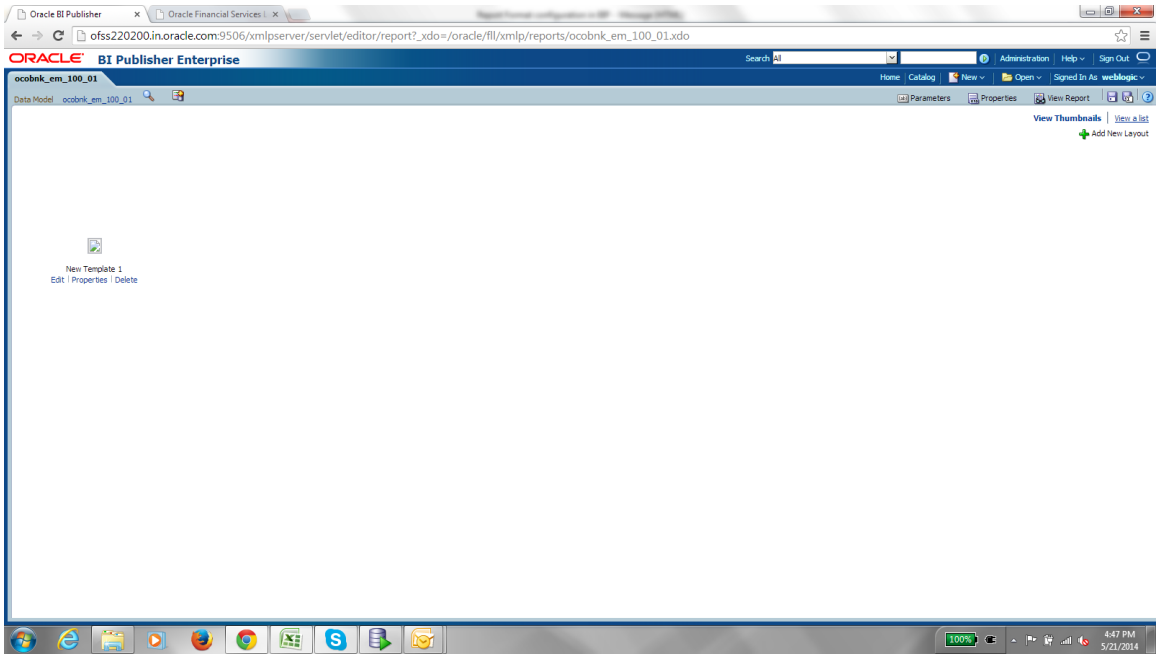
**Note:** There is no Impact on customization letters when a new base patch applied in the system. All customized letters will not be override , removed or modified.

# 11. Setting up the output Format For BIP Reports

1. Go to catalog and select the desired report (XDO) , Press the edit.

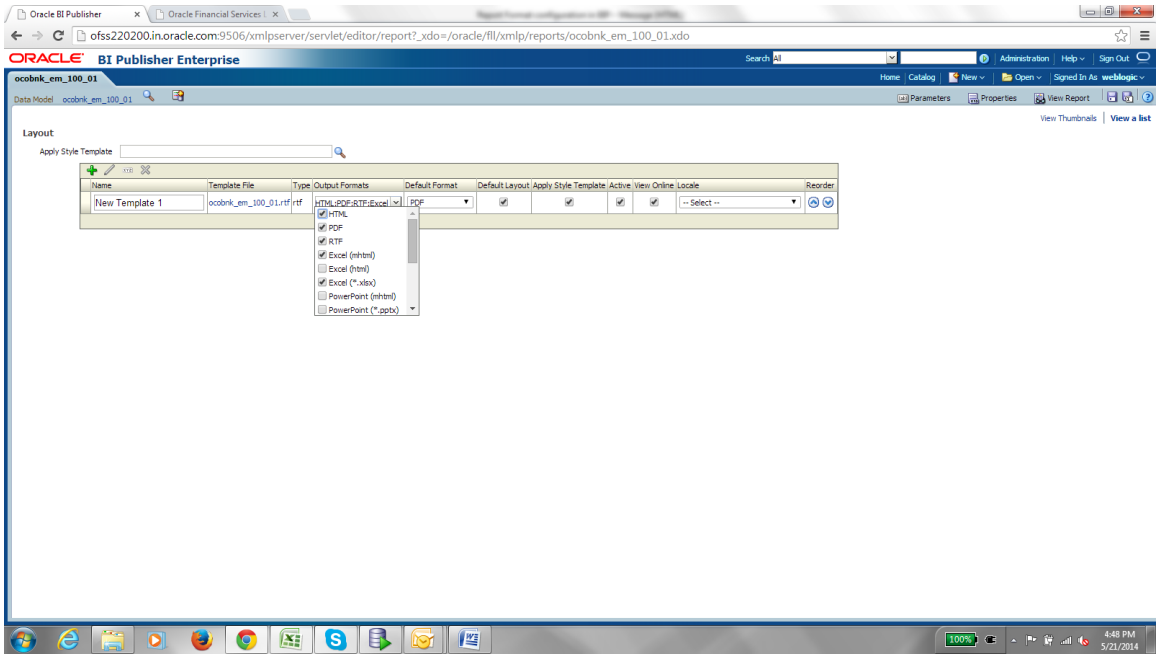



2. On Edit it will open the following page, On right corner there is “View a List “ link , click on it.

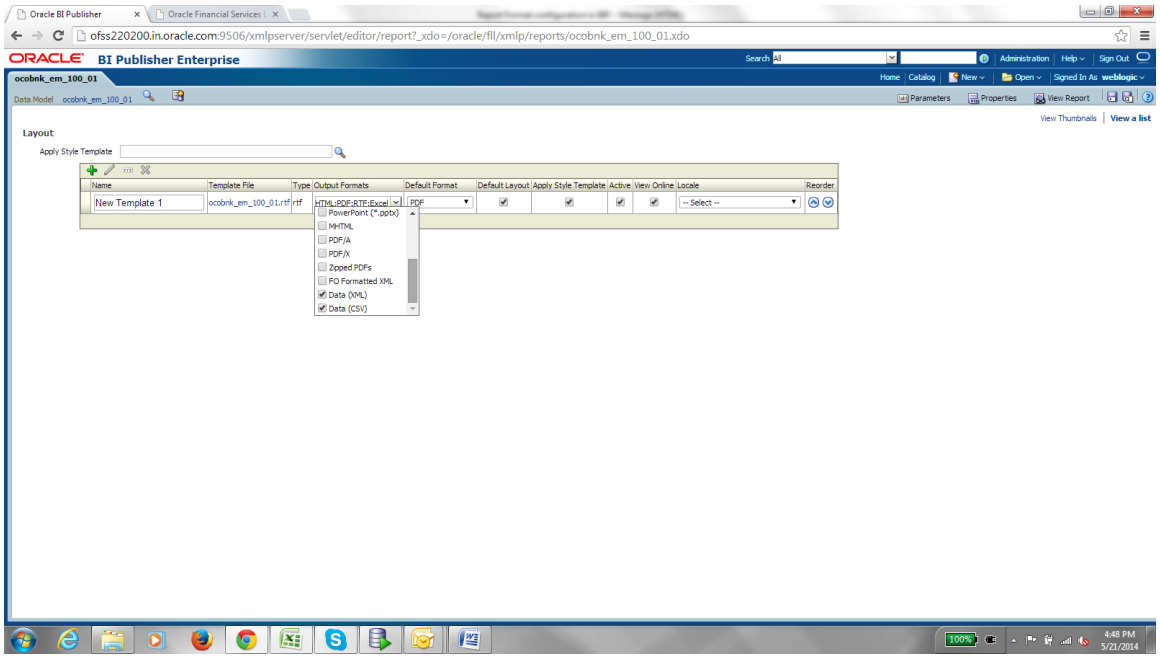



3. It will open the Layout format, click on the “Output Formats” dropdown. Select the following formats

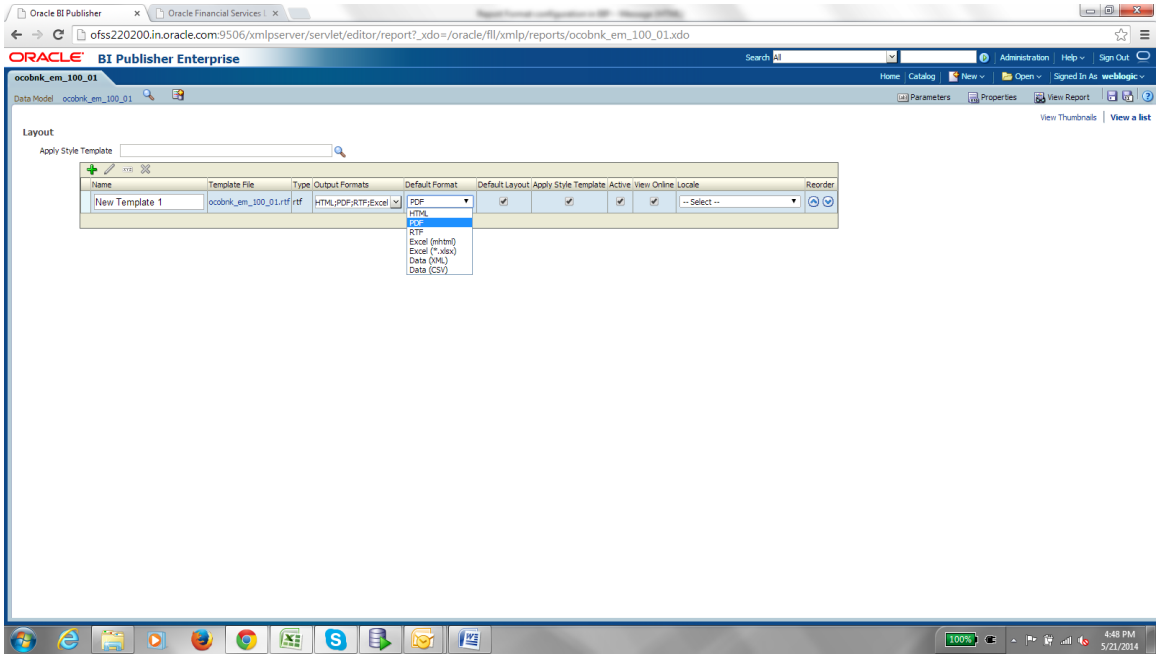
HTML,PDF ,RTF,EXCEL (.xlsx),



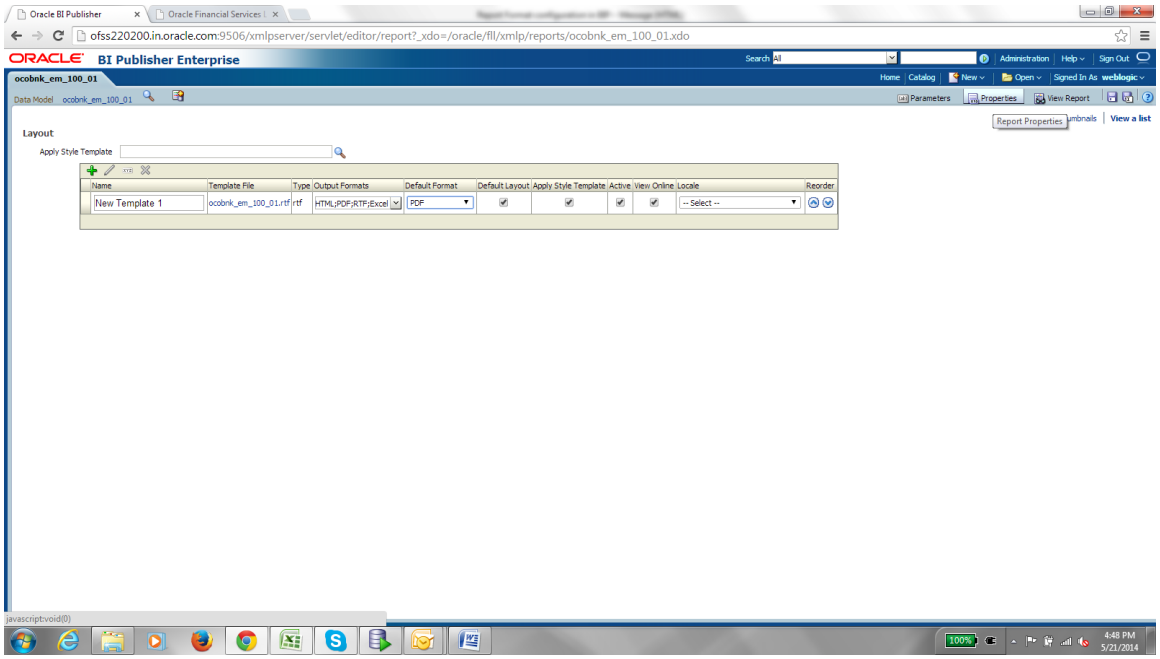

4. And in the end of drop down select DATA(XML) and DATA(CSV).



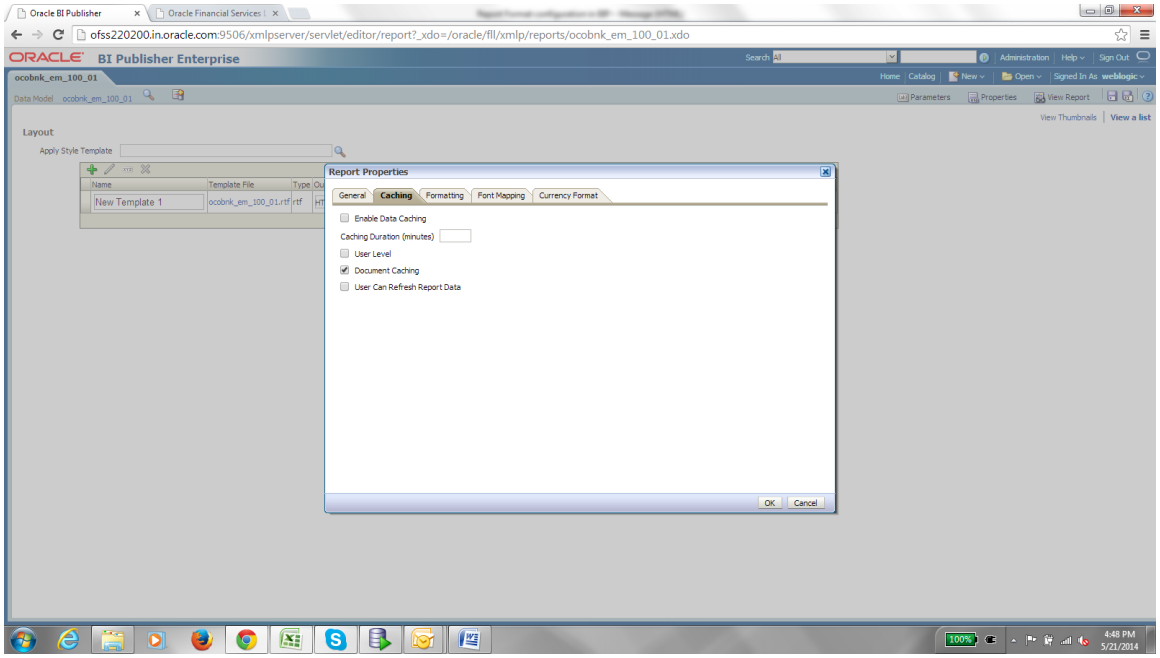

5. Also please select the Default Format as “PDF”



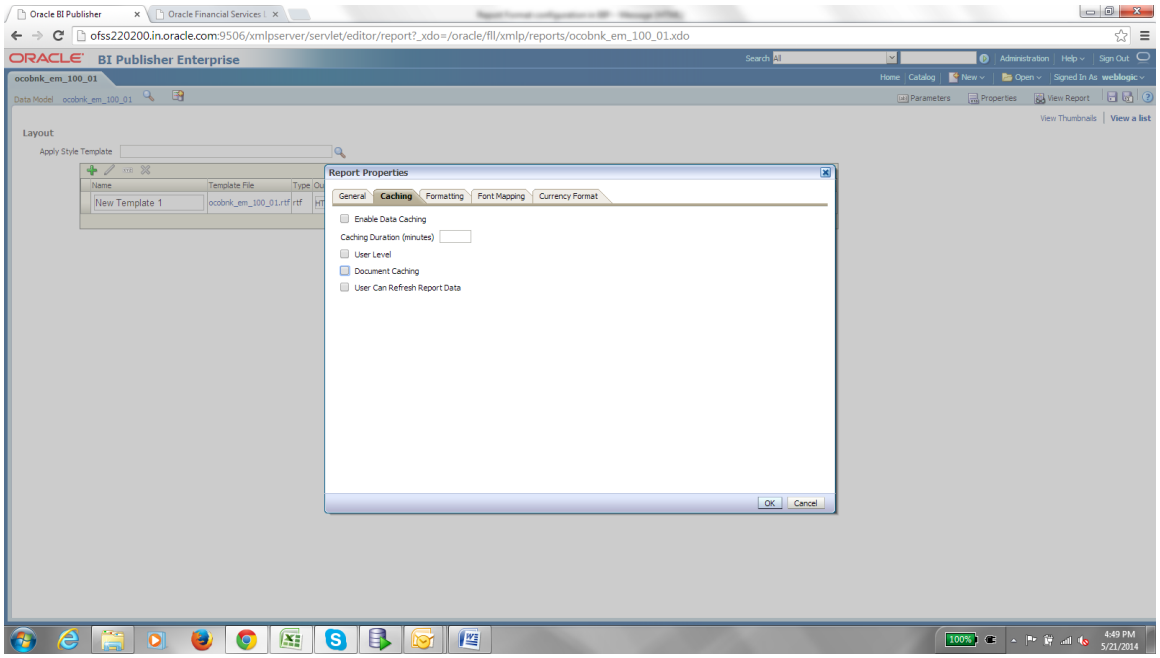

6. Once Output Format is done, I would recommend to make the Caching “False” for online reports as follows. On Right side Click on the Properties Button.



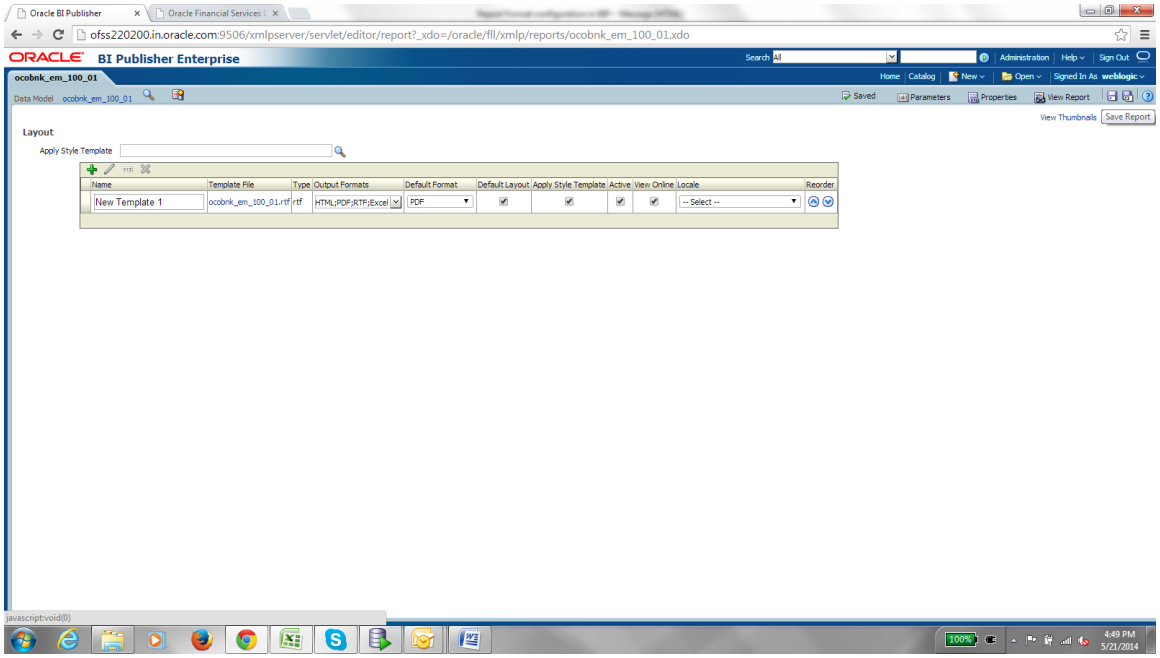

7. Select the TAB Caching, check if any of the caching is selected.




8. Unchecked all the caching if any of them are checked, By Default Document Caching is always true, make it false.




9. Save the changes.


Oracle BI Publisher
Oracle Financial Services
ofss220200.in.oracle.com:9506/xmlpserver/servlet/editor/report?_xdo=/oracle/fll/xmlp/reports/ocobnk_em_100_01.xdo
ORACLE BI Publisher Enterprise
Search All
Administration
Help
Sign Out
ocobnk_em_100_01
Home
Catalog
New
Open
Signed In As weblogic
Data Model
ocobnk_em_100_01
Saved
Parameters
Properties
View Report
View Thumbnails
Save Report
Layout
Apply Style Template
Name
Template File
Type
Output Formats
Default Format
Default Layout
Apply Style Template
Active
View Online
Locale
Reorder
New Template 1
ocobnk_em_100_01.rtf
rtf
HTML;PDF;RTF;Excel
PDF
-- Select --
javascript:void(0)
100%
4:49 PM
5/21/2014

# 12. Naming Convention for Customized Objects

| Object | Naming Convention | Comment |
|---|---|---|
| New Table | <table_name>_xyz | Same as Column Naming Convention |
| New View | <view_name>_xyz | Same as Column Naming Convention |
| New Column in OFSLL Base Version Table | abc_<column_name>_xyz | |
| New Column in OFSLL Base Version View | abc_<column_name>_xyz | |
| New Sequence | abc_seqnum_xyz | |
| New Unique Index | abc_udx_xyz/ abc_udx2_xyz | |
| New Non Unique Index | abc_idx_xyz/ abc_idx2_xyz | |
| New System Parameter (Seed Data) | <system_parmeter>_xyz | |
| New Lookup Type (Seed Data) | <lookup_type>_xyz | |
| New Lookup Code (Seed Data) | <lookup_code>_xyz | |
| New Other (Seed Data) | <seed_code>_xyz | |
| New Correspondence Function | <function_name>_xyz | |
| New Correspondence Element | <element_name>_xyz | |
| New Package Name (EM/EN) | xyz<package_name> | |
| New Package File Name(EM/EN) | xyz<package_name> | |
| **New Package Name (EX)** | <package_name> | |
| **New Package File Name(EX)** | **xyz<package_name>** | |
| New Report File Name | xyz<report_name> | |

| Object | Naming Convention | Comment |
|---|---|---|
| View File Name | xyz<view file name> | |

- Where ‘xyz’ is Customer Unique Id
- Signature of Base OFSLL Package Functions, Package Procedures, Reports, Correspondences and Faxes should not be changed.
- No New Functions or Procedures should be added to OFSLL Base Packages.
- List of Objects with exceptions must be published.

When checking-in custom code in version control software, follow theguidelines given below:

1. Instead of putting all the code in one directory, follow the Base Engine directory structure.
2. For New custom Engine Create a New Engine directory.
3. Follow the naming convention for the files. All package files should **start** with three-character client name.

e.g.: ulnapp_el_100_01.pkb will become : xyzulnapp_el_100_01.pkb for XYZ Bank.

uln_evw.sql will become : xyzuln_evw.sql for XYZ Bank.

DDL scripts should end with the three-character client name.

e.g. crt_vw_applications.sql will become crt_vw_applications_**xyz**.sql for XYZ Bank.

# 13. RESTful WebServices Extensibility

Refer to the following section for details on extensibility for the below RESTful WebServices:

- Generic Post Transaction
- Account On Boarding

## 13.1 Generic Post Transaction

Based on the type of element names in the below mentioned tables there is a sub element and child element along with their data types. If any custom field is required which is of date type, TransactionDateParameter should be used along with other fields and their values.

Same is applicable for other data types.

### 13.1.1 Producer related transaction

| Element name | Sub Element | Child Element | Data Type | Values(Example) |
|---|---|---|---|---|
| TransactionDate Parameter | | | | |
| | ParameterDetail s | | | |
| | | ParameterName | String | NA |
| | | ParameterValue | Date | NA |
| TransactionStrin gParameter | | | | |
| | ParameterDetail s | | | |
| | | ParameterName | String | ACC_NBR |
| | | ParameterValue | String | 123654 |
| | ParameterDetail s | | | |
| | | ParameterName | String | PTX_COMMENT |
| | | ParameterValue | String | Test Comment |
| | ParameterDetail s | | | |

| Element name | Sub Element | Child Element | Data Type | Values(Example) |
|---|---|---|---|---|
| | | ParameterName | String | PTX_REFERENCE |
| | | ParameterValue | String | Test Reference |
| TransactionNumberParameter | | | | |
| | ParameterDetails | | | |
| | | ParameterName | String | PTX_AMT |
| | | ParameterValue | Number | 150 |

### 13.1.2 Other Transactions

| Element name | Sub Element | Child Element | Data Type | Values |
|---|---|---|---|---|
| TransactionDateParameter | | | | |
| | ParameterDetails | | | |
| | | ParameterName | String | |
| | | ParameterValue | Date | |
| TransactionStringParameter | | | | |
| | ParameterDetails | | | |
| | | ParameterName | String | |
| | | ParameterValue | String | |
| TransactionNumberParameter | | | | |
| | ParameterDetails | | | |
| | | ParameterName | String | |
| | | ParameterValue | Number | |

**Sample format**

```
<?xml version="1.0" encoding="UTF-8" ?>
<PostTransactionRequest >
  <UserCode></UserCode>
  <TransactionDetails>
    <TransactionType></TransactionType>
    <EntityReferenceNumber></EntityReferenceNumber>
    <TransactionCode></TransactionCode>
    <TransactionDateParameter>
      <ParameterDetails>
<ParameterName> DateParamName</ParameterName>
  <ParameterValue> DateParamValue</ParameterValue>
   </ParameterDetails>
    </TransactionDateParameter>
    <TransactionStringParameter>
      <ParameterDetails><ParameterName> StringParamName</ParameterName>
  <ParameterValue> StringParamValue</ParameterValue>
</ParameterDetails>
    </TransactionStringParameter>
    <TransactionNumberParameter>
      <ParameterDetails><ParameterName> NumberParamName</ParameterName>
  <ParameterValue> NumberParamValue</ParameterValue>
</ParameterDetails>
    </TransactionNumberParameter>
    <Result>
      <ResultId></ResultId>
```

<Status></Status>

<StatusDetails></StatusDetails>

</Result>

</TransactionDetails>

</PostTransactionRequest>

**Below are the package details for generic post transaction**

xcsupd_ew_100_01. xcsupd_ew_100_01( );

xcsupd_em_100_01. post_txn();

xcsupd_en_100_01. post_batch_txn();

**You can do the customization on xcsupd_en_100_01. post_batch_txn();**

Below are the exit points:-

**BEFORE:**

xcsupd_ex_100_01.cv_ post_batch _txn_bfr =

cmncon_cl_000_01.CUSTOMIZED THEN

xcsupd_ex_100_01.post_batch_txn_bfr(iv_txn_tab_t IN xae_att_tab_t, iv_txn_result_rec_t OUT xcs_txn_result_rec_t);

**REPLACE:**

xcsupd_ex_100_01.cv_post_batch_txn_rep =

cmncon_cl_000_01.CUSTOMIZED THEN

xcsupd_ex_100_01.post_batch_txn_rep(iv_txn_tab_t IN xae_att_tab_t , iv_txn_result_rec_t OUT xcs_txn_result_rec_t );

**AFTER:**

- xcsupd_ex_100_01.post_batch_txn_afr()

xcsupd_ex_100_01.cv_post_batch_txn_afr =

cmncon_cl_000_01.CUSTOMIZED THEN

xcsupd_ex_100_01.post_batch_txn_afr(iv_txn_tab_t IN xae_att_tab_t , iv_txn_result_rec_t OUT xcs_txn_result_rec_t );

**IN parameter is Tab Type object:**

TYPE xae_att_rec_t AS OBJECT (

ATT_NAME VARCHAR2(30),

ATT_VALUE VARCHAR2(240))

**OUT parameter is Rec Type object:**

xcs_txn_result_rec_t AS OBJECT (

XTR_BMT_ID NUMBER

, XTR_TXN_RESULT VARCHAR2(2000)

, XTR_BMT_STATUS VARCHAR2(30)

, XTR_TXN_ERROR VARCHAR2(2000))

## 13.2 Account On Boarding

Below mentioned table has element name which indicates which type of custom data is passed in that enclosing the name and its value in keyname and keyvalue respectively.

| Element name | Sub Element | Data Type |
|---|---|---|
| CustomUserDefinedStringData | KeyName | String |
| | KeyValue | String |
| | | |
| | | |
| CustomUserDefinedNumberData | KeyName | Strign |
| | KeyValue | Number |
| | | |
| | | |
| CustomUserDefinedDateData | KeyName | String |
| | KeyValue | Date |

**Sample XML**

<Custom>

<CustomUserDefinedStringData>

<KeyName>Middle Name</KeyName>

<KeyValue>Singh</KeyValue>

</CustomUserDefinedStringData>

<CustomUserDefinedNumberData>

<KeyName>Age</KeyName>

<KeyValue>25</KeyValue>

</CustomUserDefinedNumberData>

<CustomUserDefinedDateData>

<KeyName>FiestPmtDate</KeyName>

<KeyValue>2016-07-14T11:53:40</KeyValue>

</CustomUserDefinedDateData>

</Custom>

**Below are the package details for Account on Boarding**

acxprc_ew_100_01. acxprc_ew_100_01( );

acxprc_em_100_01. process_account ();

Lookup Validations- involves the following packages related to

applicant details -> acxapl_en_100_01

field investigation -> acxafi_en_100_01

business details -> acxbsd_en_100_01

assets -> acxase_en_100_01

seller details -> acxsdi_en_100_01

contract details -> acxcon_en_100_01

Deriving the product data -> acxsel_el_100_01

Deriving the contract data -> acxsel_en_111_01 (for loan)

acxsel_en_112_01 (for line)

acxsel_en_121_01 (for lease)

Insertion in to the iTables -> acxins_en_100_01.ins()

Edits Validation -> aceval_ew_100_01. aceval_ew_100_01()

Account Creation -> acraai_ew_100_01. acraai_ew_100_01()

Any error occurs in the process -> acxprc_el_100_01. insert_error()

**You can do the customization on the following packages**

acxprc_em_100_01

**BEFORE:**

acxprc_em_100_01. CV_PROCESS_ACCOUNT_BFR=

cmncon_cl_000_01.CUSTOMIZED THEN

acxprc_ex_100_01.process_account_bfr(iv_app_rec ,ov_res_rec);

**REPLACE:**

acxprc_ex_100_01.CV_PROCESS_ACCOUNT_REP = cmncon_cl_000_01.CUSTOMIZED THEN

acxprc_ex_100_01.process_account_bfr(iv_app_rec ,ov_res_rec);

**AFTER:**

acxprc_ex_100_01.CV_PROCESS_ACCOUNT_AFR = cmncon_cl_000_01.CUSTOMIZED THEN

acxprc_ex_100_01.process_account_afr(iv_app_rec,ov_res_rec);

**acxins_en_100_01**

For this package, all the procedures are having the before, replace and after exit points:-

The procedures are:-

Main procedure that calls other procedures to insert the payload data:-

ins(iv_app_rec IN OUT NOCOPY acx_acc_rec_t ,ov_res_rec IN OUT NOCOPY acx_res_rec_t ,iv_axn_rec IN acx_axn_evw%ROWTYPE ,iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t ,iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t ,iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application details:-

insert_app(iv_app_rec IN OUT NOCOPY acx_app_rec_t,iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant details:-

insert_apl(iv_apl_rec IN OUT NOCOPY acx_apl_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant telecoms details:-

insert_apt(iv_apt_rec IN OUT NOCOPY acx_apt_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant address details:-

insert_apa(iv_apa_rec IN OUT NOCOPY acx_apa_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant employment details:-

insert_ape(iv_ape_rec IN OUT NOCOPY acx_ape_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant tracking details:-

insert_alt(iv_alt_rec IN OUT NOCOPY acx_alt_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the business details details:-

insert_bsd(iv_bsd_rec IN OUT NOCOPY acx_bsd_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the business affiliates details:-

insert_bsl(iv_bsl_rec IN OUT NOCOPY acx_bsl_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the business partners details:-

insert_bsp(iv_bsp_rec IN OUT NOCOPY acx_bsp_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the business address details:-

insert_bsa(iv_bsa_rec IN OUT NOCOPY acx_bsa_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the business telecoms details:-

insert_bst(iv_bst_rec IN OUT NOCOPY acx_bst_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application assets details:-

insert_ase(iv_ase_rec IN OUT NOCOPY acx_ase_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application asset valuations details:-

insert_avl(iv_avl_rec IN OUT NOCOPY acx_avl_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application asset attributes details:-

insert_atr(iv_atr_rec IN OUT NOCOPY acx_atr_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application asset tracking details:-

insert_atk(iv_atk_rec IN OUT NOCOPY acx_atk_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application asset tracking attribute details:-

insert_ata(iv_ata_rec IN OUT NOCOPY acx_ata_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application seller details:-

insert_sdi(iv_sdi_rec IN OUT NOCOPY acx_sdi_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, _ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application seller address details:-

insert_sda(iv_sda_rec IN OUT NOCOPY acx_sda_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application repayment schedule details:-

insert_apc(iv_apc_rec IN OUT NOCOPY acx_apc_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application payment change schedule details:-

insert_acs(iv_acs_rec IN OUT NOCOPY acx_acs_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, _ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application contract details:-

insert_acd(iv_acd_rec IN OUT NOCOPY acx_acd_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, _ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application contract insurance details:-

insert_acd_ins(iv_app_rec IN OUT NOCOPY acx_acc_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application insurance details:-

insert_ins(iv_ins_rec IN OUT NOCOPY acx_ins_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application tradein details:-

insert_apd(iv_apd_rec IN OUT NOCOPY acx_apd_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application subvention details:-

insert_asn(iv_asn_rec IN OUT NOCOPY acx_asn_rec_t, ov_res_rec IN OUT NOCOPY acx_res_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application subvention_details details:-

insert_asl(iv_asl_rec IN OUT NOCOPY acx_asl_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application ach details:-

insert_aac(iv_aac_rec IN OUT NOCOPY acx_aac_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application post dated check details:-

insert_pdc(iv_pdc_rec IN OUT NOCOPY acx_pdc_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application references details:-

insert_aar(iv_aar_rec IN OUT NOCOPY acx_aar_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application contract fees details:-

insert_afe(iv_afe_rec IN OUT NOCOPY acx_afe_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application comment details:-

insert_acm(iv_acm_rec IN OUT NOCOPY acx_acm_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to update the application details:-

update_con(iv_con_rec IN OUT NOCOPY acx_con_rec_t, iv_aro_rec IN OUT NOCOPY acx_aro_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the application tracking attribute details:-

insert_aat(iv_aat_rec IN OUT NOCOPY acx_aat_rec_t, iv_ext_rec_str IN OUT NOCOPY acx_ext_str_tab_t, iv_ext_rec_num IN OUT NOCOPY acx_ext_num_tab_t, iv_ext_rec_dt IN OUT NOCOPY acx_ext_dt_tab_t);

Procedure to insert the applicant field investigation details:-

insert_afd(iv_afd_rec IN OUT NOCOPY acx_afd_rec_t);

Procedure to insert the applicant field investigation_details details:-

insert_afd(iv_afd_rec IN OUT NOCOPY acx_afd_rec_t);

**acxsel_en_111_01(LOAN)**

**BEFORE**

acxsel_ex_111_01.CV_SEL_CON_DTLS_BFR = cmncon_cl_000_01.CUSTOMIZED THEN

    acxsel_ex_111_01.sel_con_dtls_bfr(iv_con_rec,iv_prd_rec);

**REPLACE**

acxsel_ex_111_01.CV_SEL_CON_DTLS_REP = cmncon_cl_000_01.CUSTOMIZED THEN

    acxsel_ex_111_01.sel_con_dtls_rep(iv_con_rec,iv_prd_rec);

**AFTER**

acxsel_ex_111_01.CV_SEL_CON_DTLS_AFR = cmncon_cl_000_01.CUSTOMIZED THEN

    acxsel_ex_111_01.sel_con_dtls_afr(iv_con_rec,iv_prd_rec);

Similarly the exit points have been added for the line and lease products also for selecting contract details in acxsel_en_112_01 and acxsel_en_121_01

**Acraai_en_111_01(LOAN)**

**BEFORE**

```
acraai_ex_111_01.CV_CREATE_ACCOUNT_BFR = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.create_account_bfr(iv_con_rec);
```

**REPLACE**

```
acraai_ex_111_01.CV_CREATE_ACCOUNT_REP = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.create_account_rep(iv_con_rec);
```

**AFTER**

```
acraai_ex_111_01.CV_CREATE_ACCOUNT_AFR = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.create_account_afr(iv_con_rec);
```

**BEFORE**

```
acraai_ex_111_01.CV_SET_ACC_NBR_BFR = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.set_acc_nbr_bfr(
                        iv_con_rec
                       ,iv_acc_aad_id
                       ,iv_acc_nbr);
```

**REPLACE**

```
acraai_ex_111_01.CV_SET_ACC_NBR_REP = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.set_acc_nbr_rep(
                        iv_con_rec
                       ,iv_acc_aad_id
                       ,iv_acc_nbr);
```

**AFTER**

```
acraai_ex_111_01.CV_SET_ACC_NBR_AFR = cmncon_cl_000_01.CUSTOMIZED THEN
    acraai_ex_111_01.set_acc_nbr_afr(
                        iv_con_rec
                       ,iv_acc_aad_id
```

,iv_acc_nbr);

**acraai_en_111_02**

**BEFORE**

acraai_ex_111_01.CV_LOAD_CURRENT_ACC_BFR = cmncon_cl_000_01.CUSTOMIZED THEN

acraai_ex_111_01.load_current_acc_bfr(

iv_con_rec

,iv_acc_aad_id);

**REPLACE**

acraai_ex_111_01.CV_LOAD_CURRENT_ACC_REP = cmncon_cl_000_01.CUSTOMIZED THEN

acraai_ex_111_01.load_current_acc_rep(

iv_con_rec

,iv_acc_aad_id);

**AFTER**

acraai_ex_111_01.CV_LOAD_CURRENT_ACC_AFR = cmncon_cl_000_01.CUSTOMIZED THEN

acraai_ex_111_01.load_current_acc_afr(

iv_con_rec

,iv_acc_aad_id);

**BEFORE**

acraai_ex_111_01.cv_convert_new_acc_bfr = cmncon_cl_000_01.CUSTOMIZED THEN

acraai_ex_111_01.convert_new_acc_bfr(

iv_con_rec

,iv_acc_aad_id);

**REPLACE**

acraai_ex_111_01.cv_convert_new_acc_rep = cmncon_cl_000_01.CUSTOMIZED THEN

acraai_ex_111_01.convert_new_acc_rep(

iv_con_rec

```
                                ,iv_acc_aad_id);
```

**AFTER**

```
acraai_ex_111_01.cv_convert_new_acc_afr = cmncon_cl_000_01.CUSTOMIZED THEN

    acraai_ex_111_01.convert_new_acc_afr(

                                 iv_con_rec

                                ,iv_acc_aad_id);
```

Similarly the exit points have been added for the line and lease products also for inserting the account details in acraai_en_112_01,acraai_en_112_02 and acxsel_en_121_01,acraai_en_121_02.

**Extensible parameters are Tab Type object**

```
CREATE OR REPLACE TYPE acx_ext_dt_rec_t AS OBJECT

(

   ext_key_name   VARCHAR2(30),

   ext_key_value  DATE

);

/

CREATE OR REPLACE TYPE acx_ext_num_rec_t AS OBJECT

(

   ext_key_name   VARCHAR2(30),

   ext_key_value  NUMBER

);

/

CREATE OR REPLACE TYPE acx_ext_str_rec_t AS OBJECT

(

   ext_key_name   VARCHAR2(30),

   ext_key_value  VARCHAR2(4000)

);

/
```

**Extensibility Guide**
**September [2016]**
**Version 14.3.0.1.0**

**Oracle Financial Services Software Limited**
**Oracle Park**
**Off Western Express Highway**
**Goregaon (East)**
**Mumbai, Maharashtra 400 063**
**India**

**Worldwide Inquiries:**
**Phone: +91 22 6718 3000**
**Fax:+91 22 6718 3001**
**www.oracle.com/financialservices/**