Extensibility Guide
Oracle Financial Services Lending and Leasing
Release 14.0.0.0.0
[April] [2013]

Oracle Part Number E51531-01

ORACLE
FINANCIAL SERVICES

Table of Contents

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 11.1.1.6.0 as well as ADF Skin Editor 11.1.2.3.0.

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 and 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 made to the application, 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."

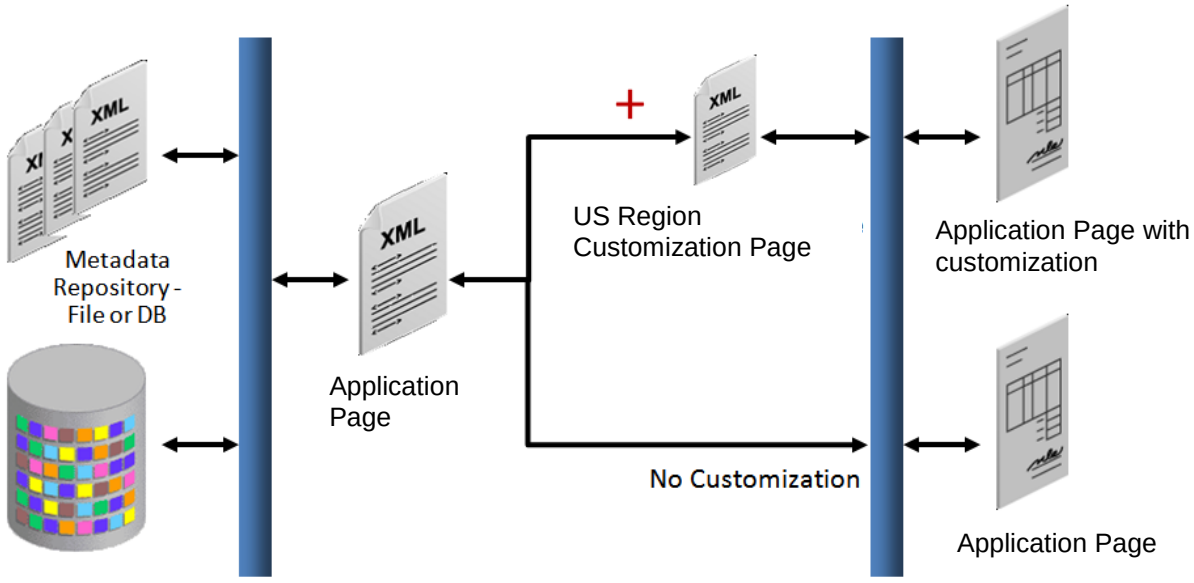
2.2 Understanding Customization Layers

The application contains customization layer that allow you to make customizations that 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 personalizations, 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 personalizations 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

You use JDeveloper when you need to customize or create business objects or new pages. While you use JDeveloper to both customize existing standard objects and to create new custom objects, the procedures you use for each 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 the customizations using the customization developer role ensures that your changes are saved to the upgrade-safe MDS Repository, and not written directly to the standard object. In the future, when you patch or upgrade your Application, your customizations held in these metadata files will not be touched, and so, you will not have to redo them.

When customizing ADF artifacts, you can create a special customization application workspace; 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 you test and deploy your customizations, they will behave as though they were native Oracle Financial Services Lending and Leasing Application.

Using the default workspace, you switch roles to the customization developer role and customize the ADF artifact you want to customize. After completion, you package and deploy the artifacts in the workspace to the Oracle Financial Services Lending and Leasing environment.

Often, you will need to perform both customizations (customizing an existing standard object) and extensions (creating a new object). For example, say you want to create a new business object and expose that new object in an existing application module. First, because you are creating a new custom business object, you would create a standard application workspace and then create your entity object. After completion, you would package the workspace as an ADF Library, and place it into a directory. Next, using the default workspace provided, add the new entity object library and the library that contains the application module to which you need to add the entity object. After both are imported, you log in using the customization developer role and make the customizations to the application module. After customizations are complete, you would deploy the customizations to the test environment.

Note

Before running JDeveloper is 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 related to an end-user task.

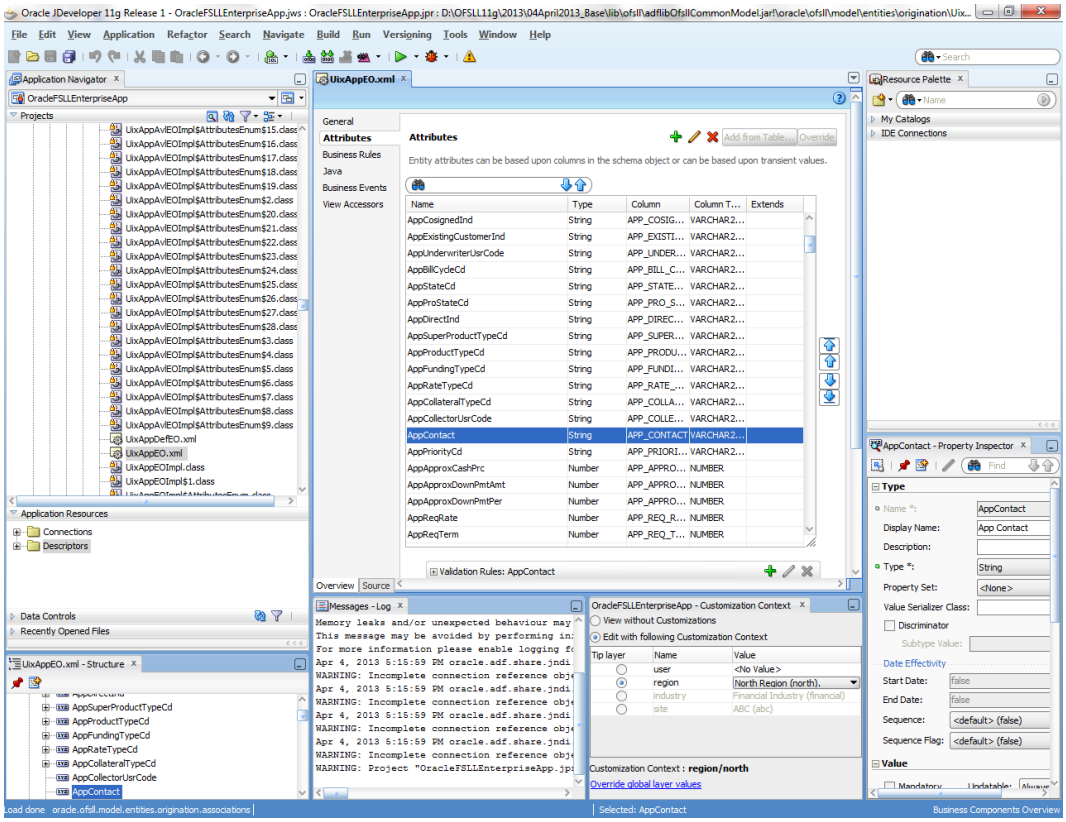
- Entity objects: An entity object represents a row in a database table and simplifies modifying its data by handling all data manipulation language (DML) operations for you. It can encapsulate business logic to ensure that your business rules are consistently enforced. You associate an entity object with others to reflect relationships in the underlying database schema to create a layer of business domain objects to reuse in multiple applications.

- View objects: A view object represents a SQL query and simplifies working with its results. You use the SQL language to join, filter, sort, and aggregate data into the shape required by the end-user task being 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, your view objects collaborate with entity objects to consistently validate and save the changes.

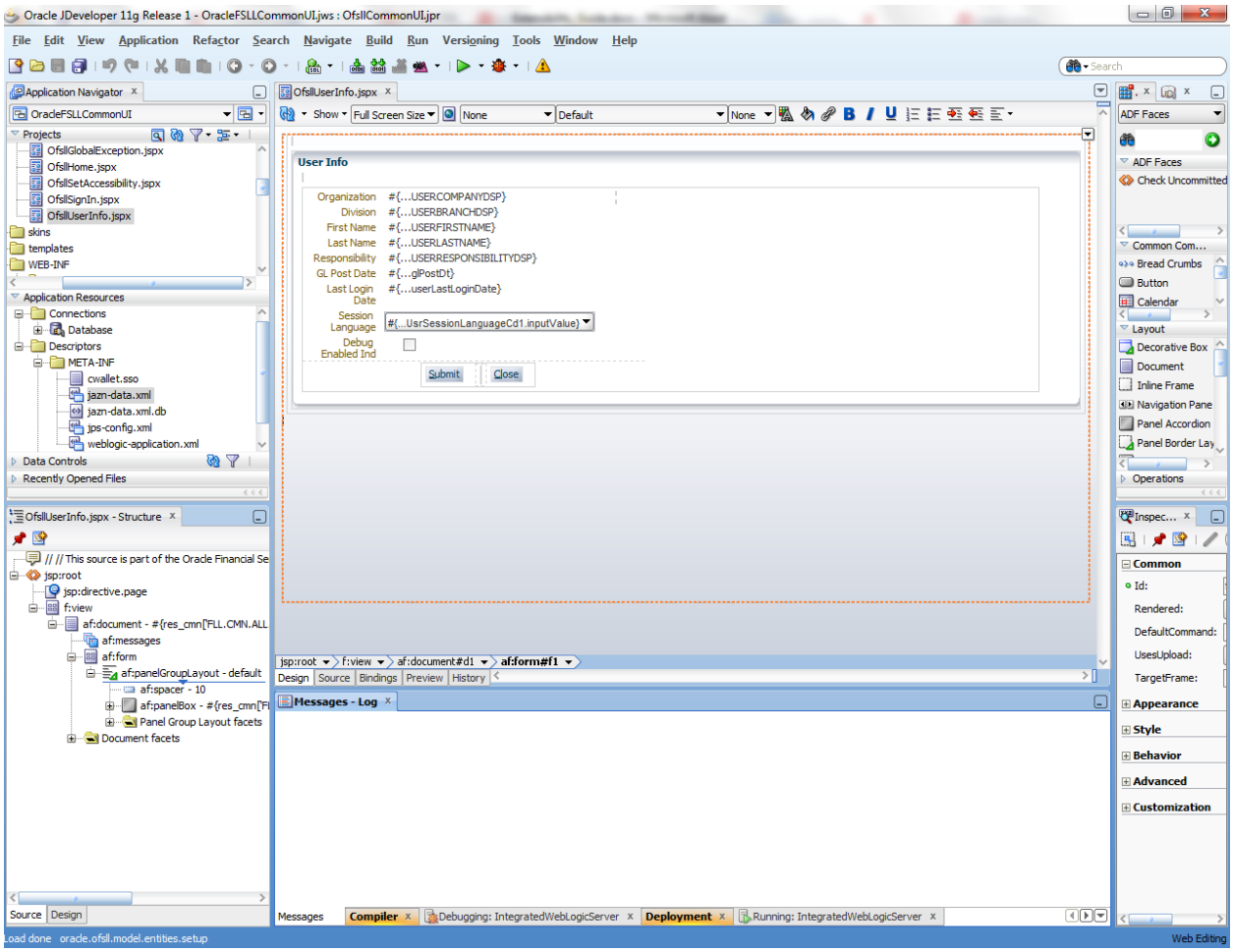
- Task flows: Task flows define the flow of control throughout an application. They also can be included in a page as a region, where users can navigate through a series of page fragments, without leaving the original page.

- 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 you customize Oracle ADF artifacts, you usually work in an overview editor that allows you to make your customizations declaratively. For example, below figure shows the editor for an entity object. Among other things, you can set validation or change how the UI displays the data.



For JSPX pages, you work in a WYSIWYG environment using the Design tab in the editor window.



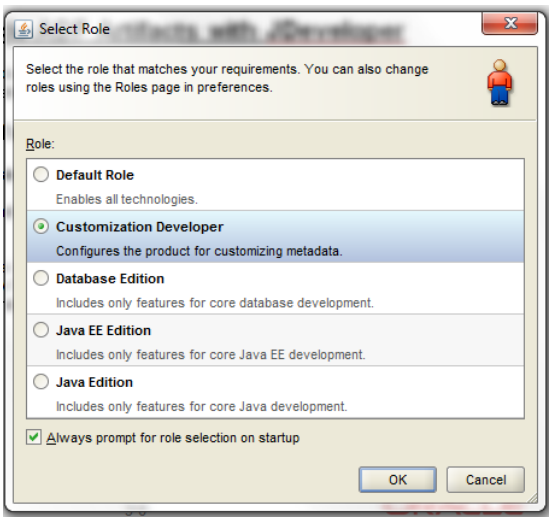
3.3 Customizing Oracle ADF Artifacts with JDeveloper

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

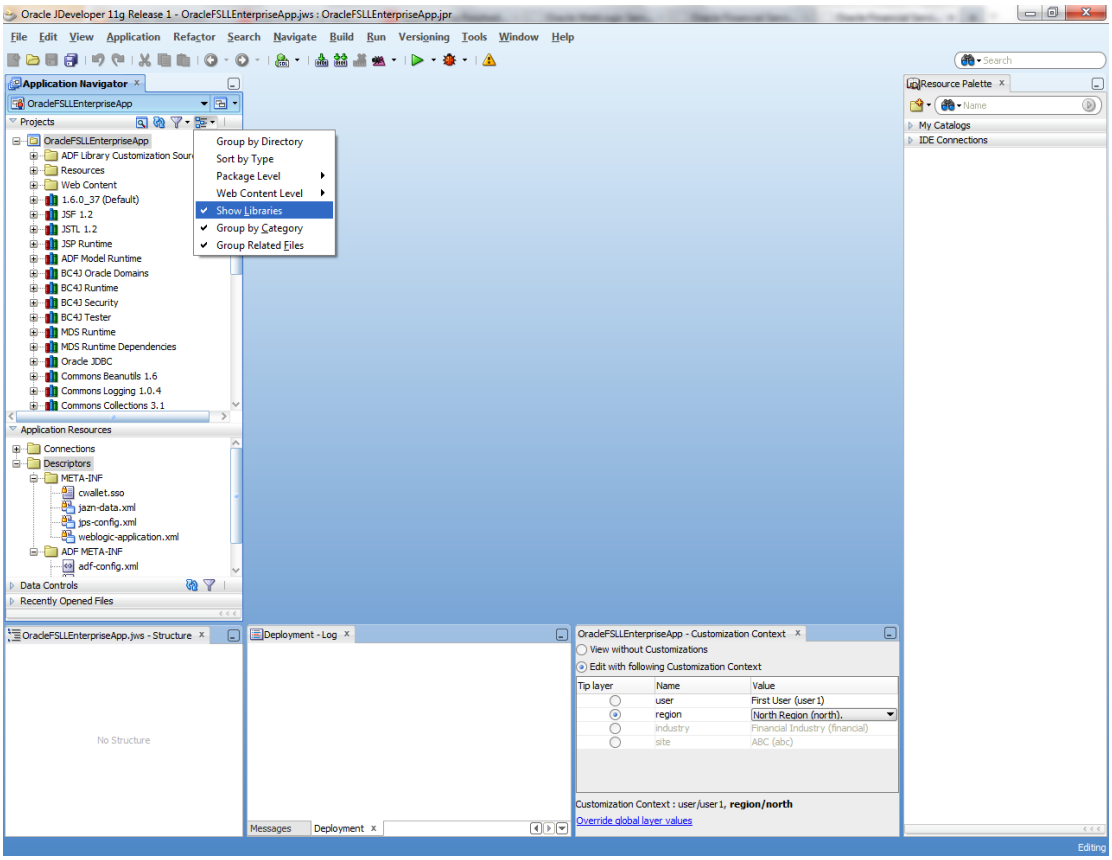
Customizing the Artifacts

You need to switch to the **Customization Developer** role before you can begin customizing.

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



The artifacts from the imported library now display in the Application Navigator pane, under the ADF Library Customizations node, and the artifact you 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.

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. You can also extend your application 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 you will need to use JDeveloper.

4.2 Customizable Application Libraries

All of the 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 are available in this library.
adflibOracleFSLLCommonTFPF.jar	Contains task flow templates and page flow templates. All the re-usable templates are available in this library
adflibOfsllCommonUI.jar	Contains all the User Interface fragments (JSFF) and taskflows (TFs) are available in this library.

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 and libraries are available as part of the application installer bundle under */cust_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 you run the JDeveloper in Customization Developer role, you need to configure JDeveloper with following settings:

1. Open JDeveloper in Default role and open the OracleFSLLCustomization/ OracleFSLLCustomization.jws project and edit the customization.properties file with appropriate values for your *Region* key layer, *Industry* key layer and *Site* key layer.

2. Rebuild the *OfsllCustomization.jar* using the default deployment profile.

3. Copy the *OfsllCustomization.jar* onto your JDeveloper installation location under *$JDEV_HOME/jdeveloper/jdev/lib/patches.*

4. 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 your *customization.properties.*

5. Copy the *CustomizationLayerValues.xml* onto your JDeveloper installation location under *$JDEV_HOME/jdeveloper/jdev.*

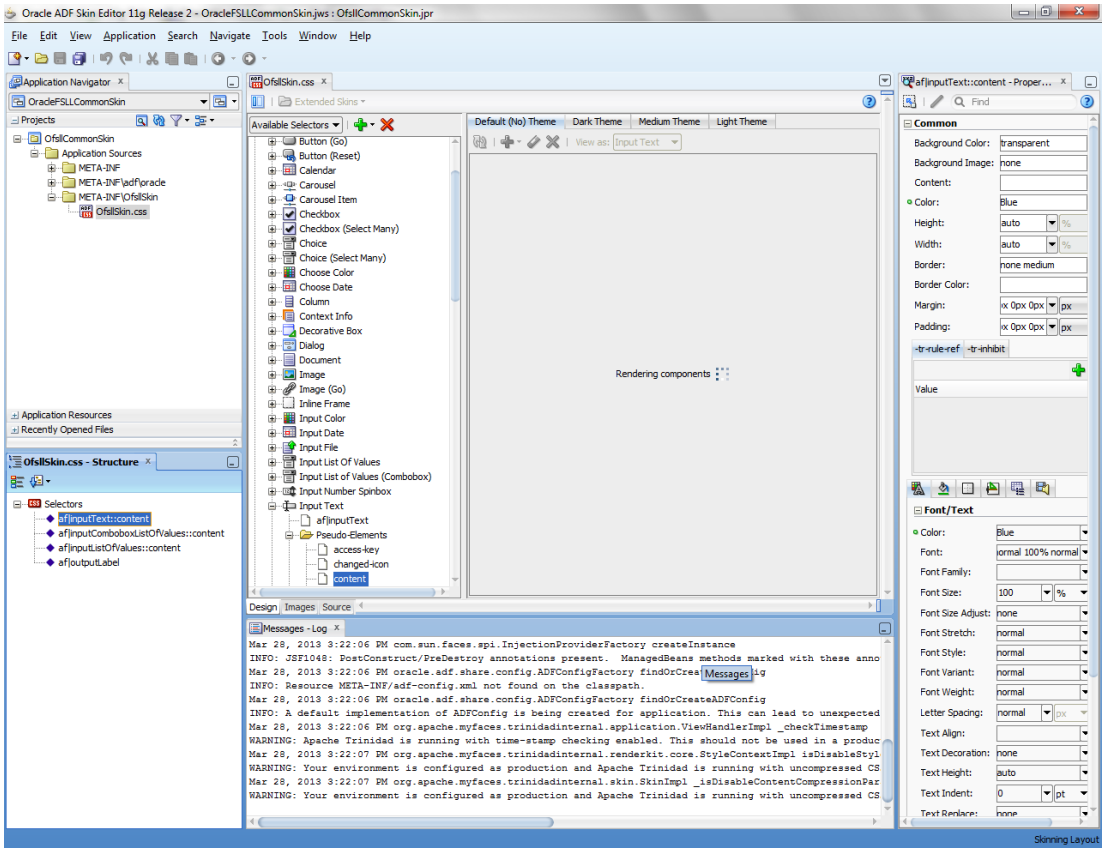
4.4 Customizing the Skin

One method of customizing skin is opening the bundled *OracleFSLLCommonSkin/OracleFSLLCommonSkin.jws* project in Oracle ADF Skin Editor Application and customize the skin details. Once the skin are customized the same can be bundled as ADF library and deploy 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

For skin can be customized using Oracle ADF Skin Editor (11.1.2.3.0) which can be downloaded from Oracle site.

If you change the default skin family name 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 you cannot add new keys.

Because you cannot define a new key in the shipped resource bundle, you need to create a new override bundle. You can accomplish this in JDeveloper by creating an XLIFF file from the New Gallery. After the file is generated, you can then enter new keys and their associated text in the XLIFF file.

To make the newly created resource bundle available for customization, you need to register the resource bundle with the customization project. To do this, package it into an ADF Library JAR file, and import the JAR file into the customization project.

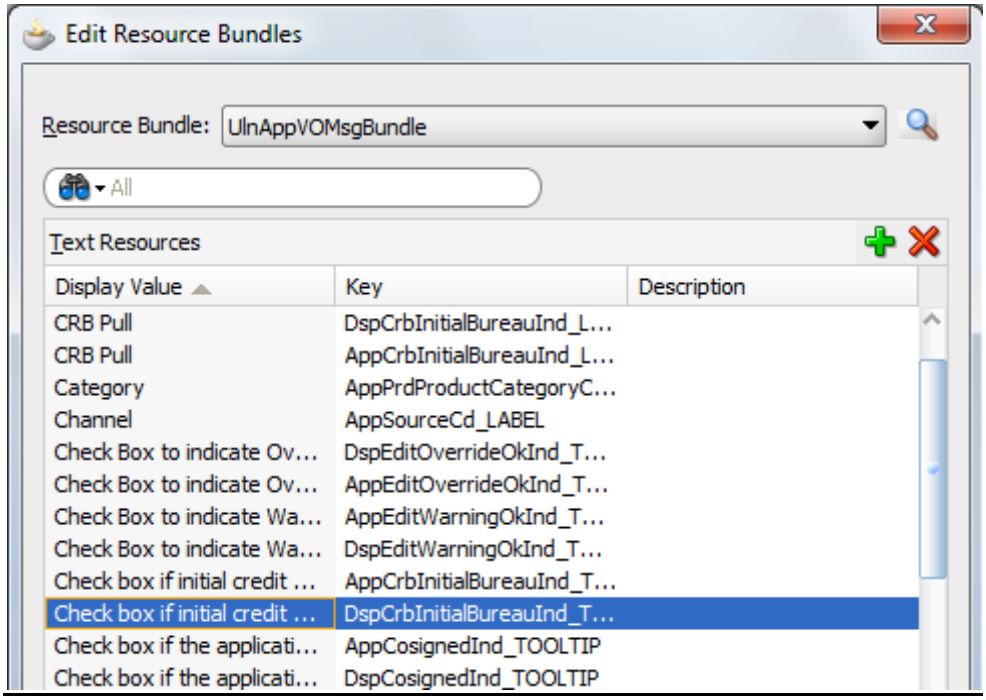
Note

All custom JAR file names must begin with the suffix Xx, for example *adflibMyJarXx.jar*

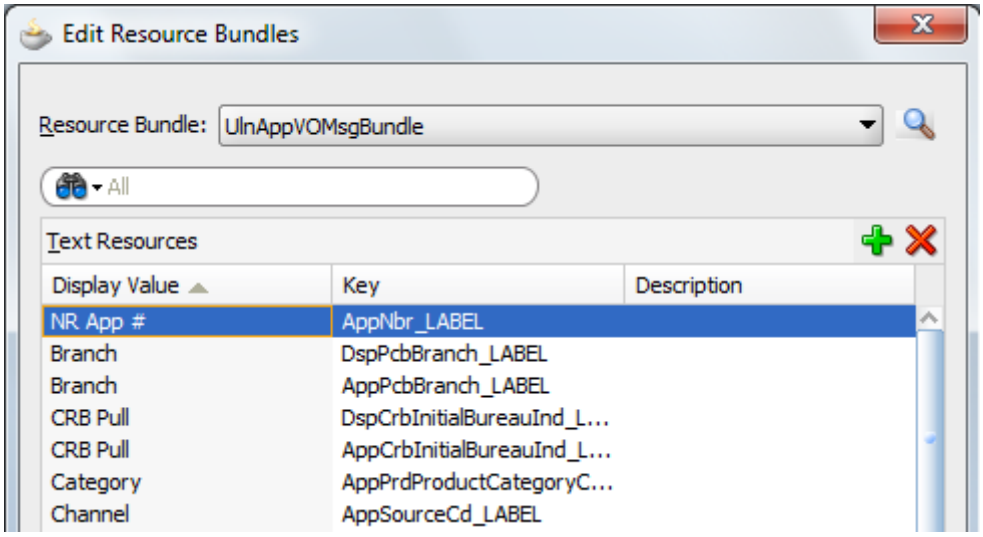
To test your customizations locally in JDeveloper Integrated Oracle WebLogic Server, you must also include the ADF Library JAR file in the APP-INF/lib directory.

Step to override a message bundle is shown:

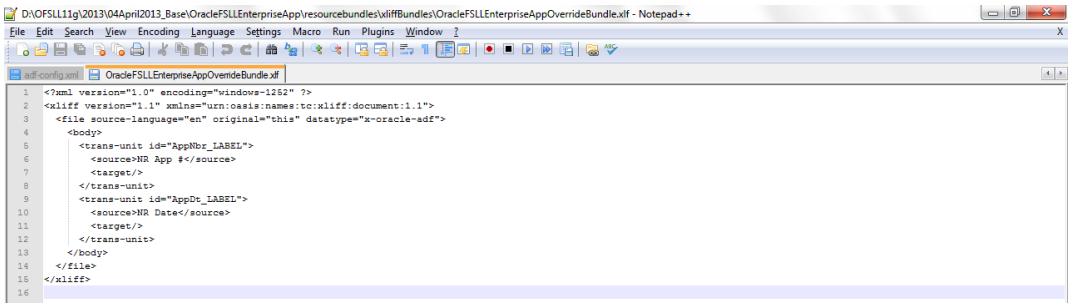
1. The Edit Resource Bundles Dialog box available under Application Menu. Navigate to the XLIFF Bundle that you are interested in overriding.



2. Change *Display Value* for the *Key* as per requirement



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

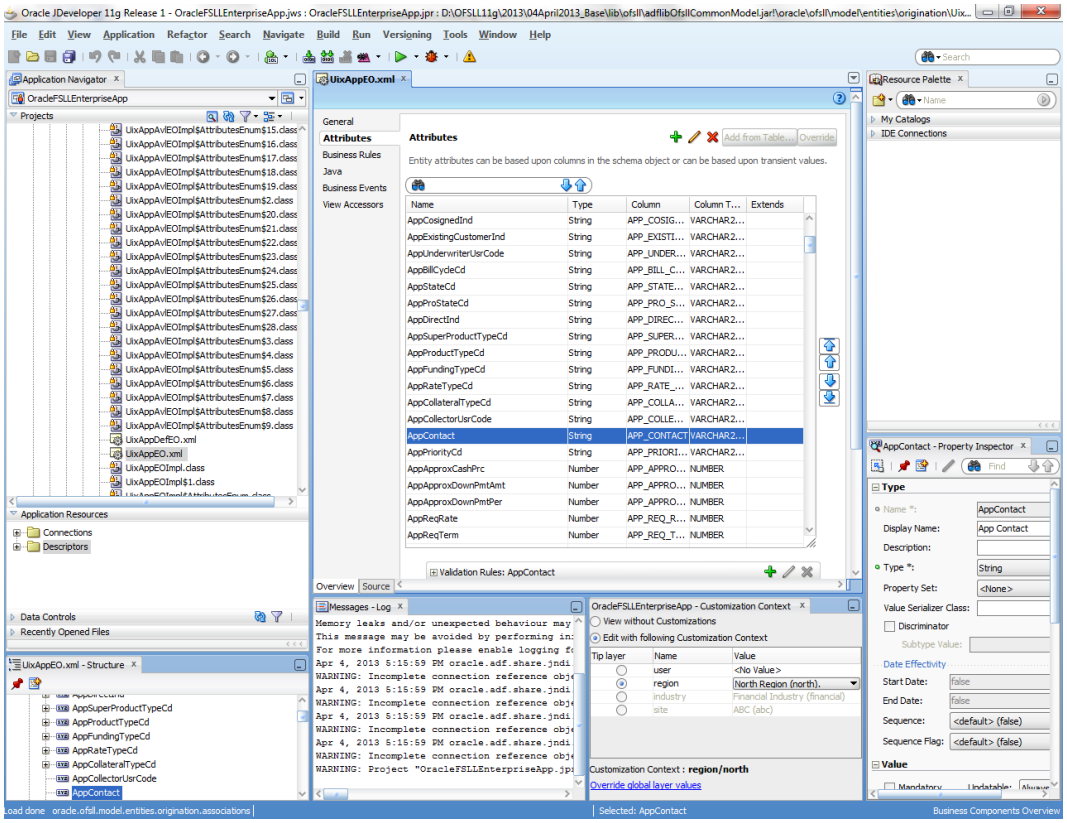


4.6 Editing Existing Business Components

Before you start customizing business objects, you'll need to determine which business objects you want to customize. Then when customizing ADF artifacts, you need to launch JDeveloper in the **Customization Developer** role, and select the appropriate layer.

Task: Edit Attributes

You can customize the properties of an attribute from an entity object or view object using JDeveloper. When you open an entity object or view object in the overview editor, you click the Attributes tab to see the attributes of the object. When you select an attribute, 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 you modify an attribute so that it requires a different UI component, you must also update the page. For example, if you add a list of values (LOV) to an attribute, you will need to edit the page to hide the existing UI component that displays the attribute, and add a new UI component that can display the LOV.

Note that some attribute properties defined in the entity object can be overridden in the view object. For example, you can define the label text for a field in an entity object and subsequently give it 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

You can add custom attributes to an entity object or view object using JDeveloper. To do this, you must launch JDeveloper in the **Customization Developer** role, and select a layer. When you open an entity object or view object in the overview editor, you click the Attributes tab to see the attributes of the object. To add a custom attribute, click the Add icon.

If you want your custom attribute to be stored in the database, you must first create the column that will be use to store it.

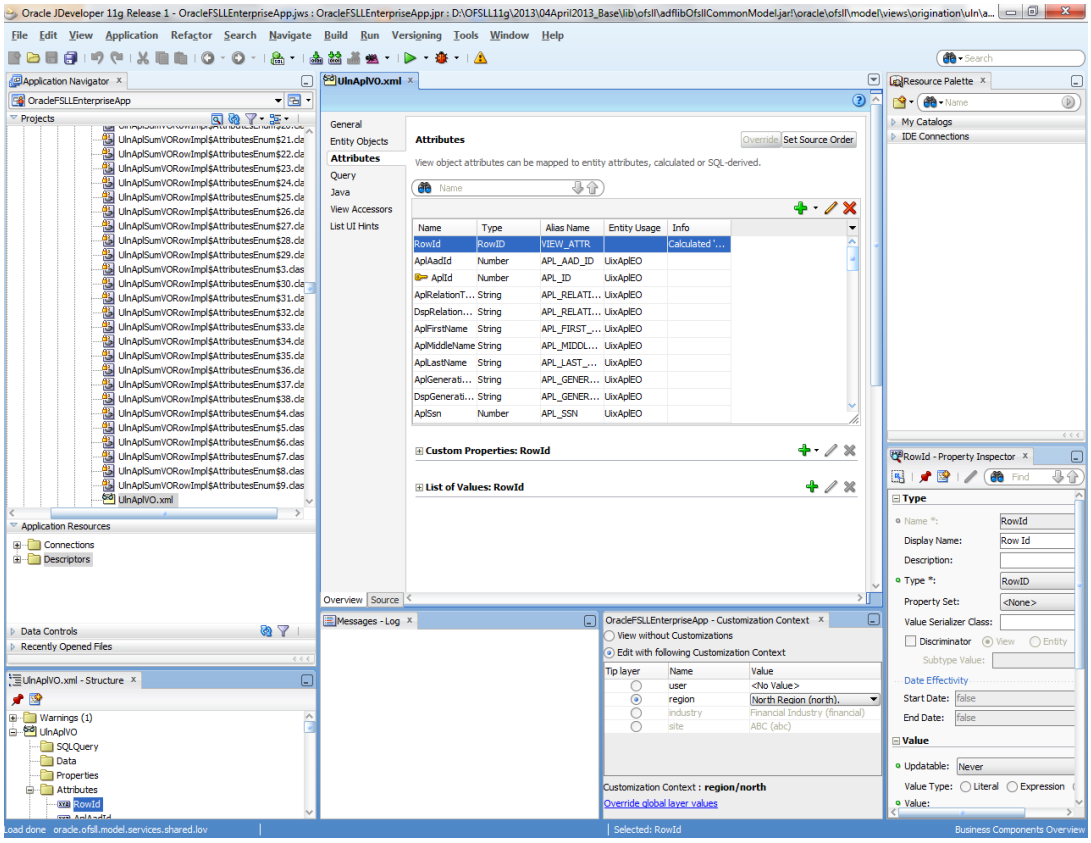
If you want your custom attributes to be displayed in the application, you must also customize the pages to display them.

Task: Edit Entity Objects

In JDeveloper, you 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, you 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



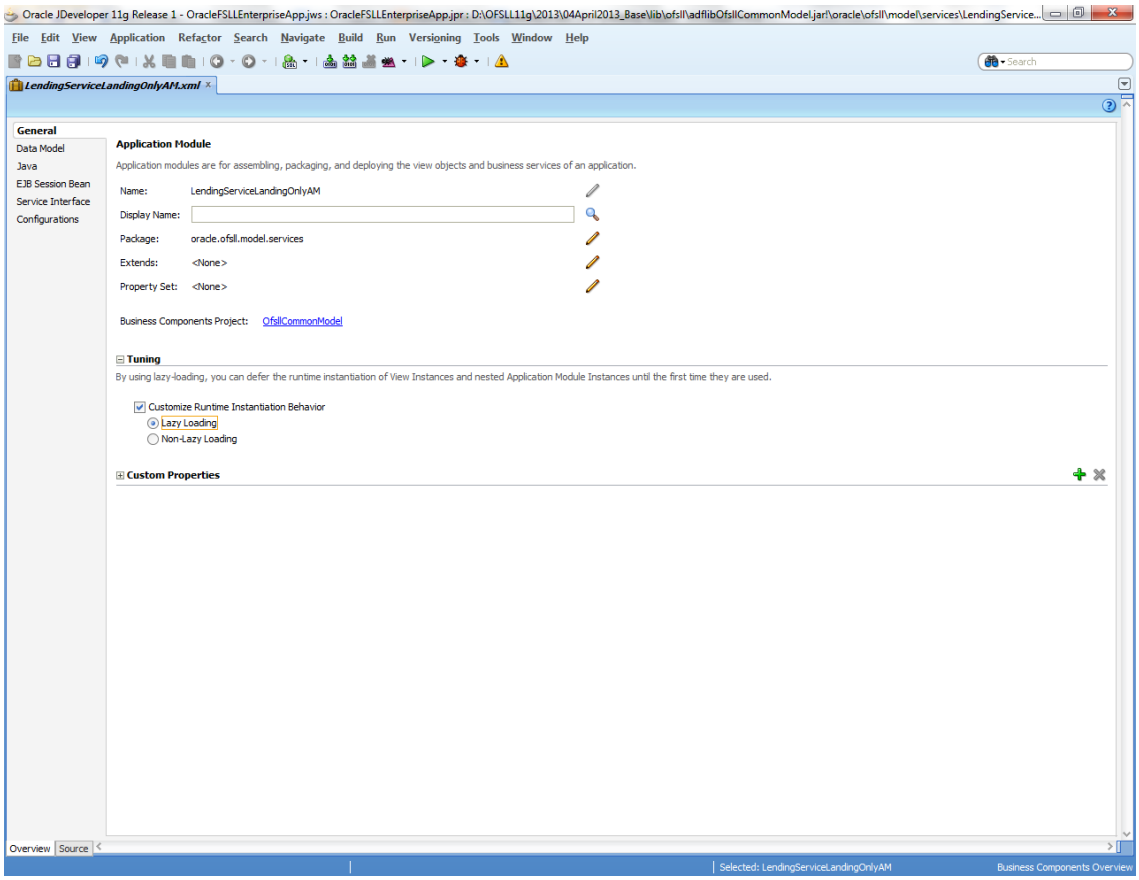
Task: Edit Validation

In JDeveloper, you edit declarative validation rules for entity objects and view objects using the overview editor. In the Application Navigator, right-click an entity object or view object, and choose **Open**. Then click the Business Rules navigator tab to view and edit the validation rules.

When implementing customizations on validation rules, you can add rules, modify the error message, and make rules more restrictive. But you should avoid removing rules or making rules less restrictive, because this can cause unpredictable results. Also, you can edit only declarative validation rules; programmatic validation rules cannot be customized.

Task: Edit Application Modules

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

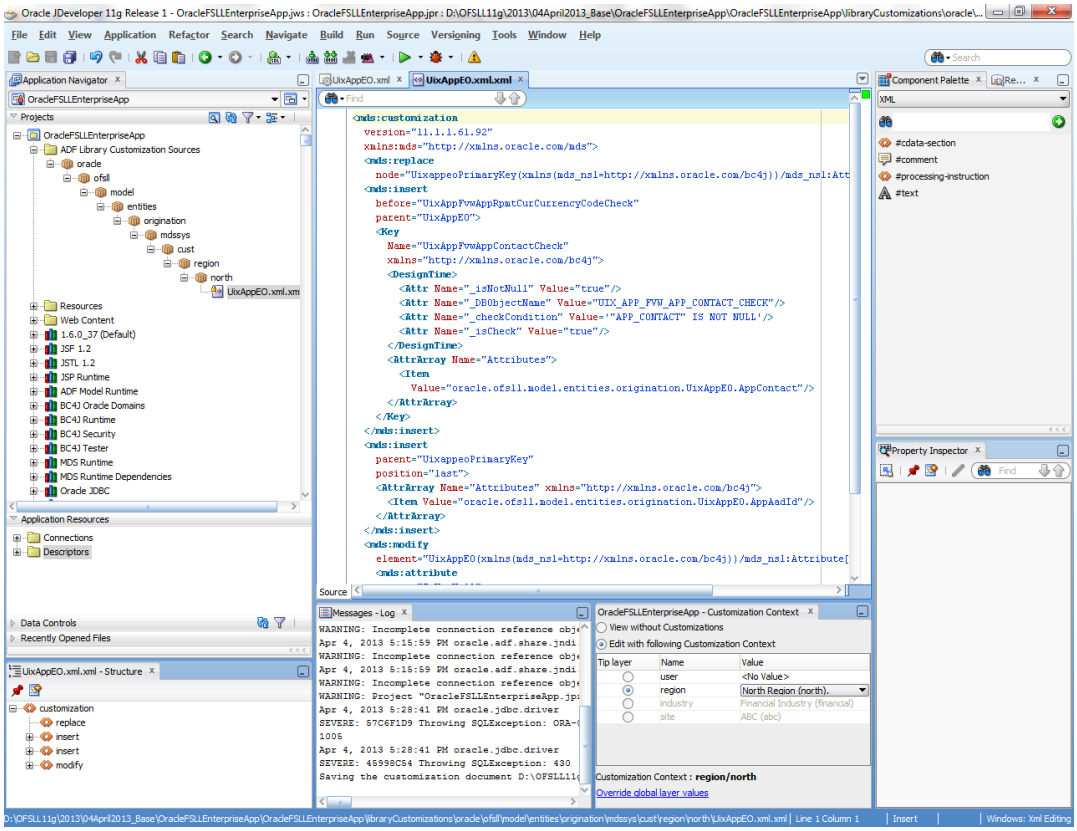


In JDeveloper, you can make the following kinds of customizations on an application module:

- 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 choose as shown below.

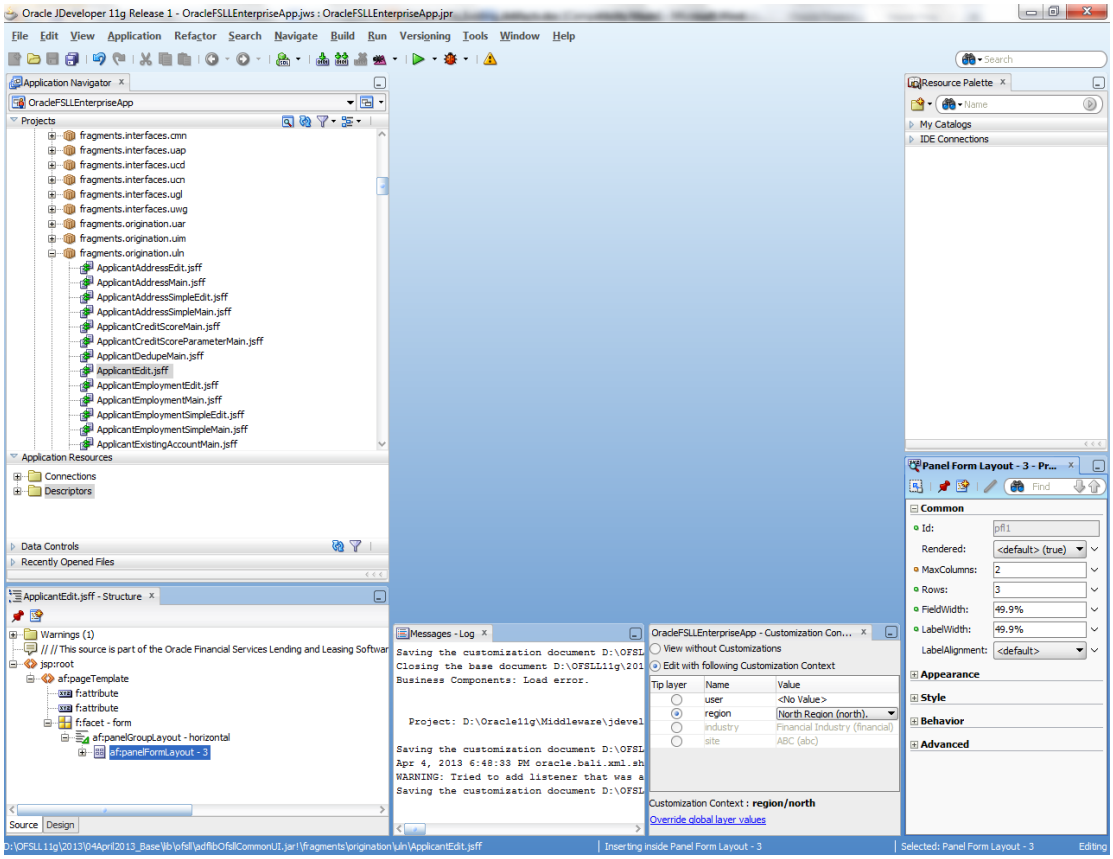


4.7 Editing Pages

You can use JDeveloper to implement customizations on the pages that are used in your application. When editing a page in JDeveloper, you must launch JDeveloper in the Customization role.

Task: Edit Pages

In JDeveloper, you use the visual editor to implement customizations on existing pages. In the Application Navigator, right-click the page you want to customize, and choose **Open**. The page is displayed in the visual editor (accessed by clicking the Design tab). Then you can edit the page as you typically would using this editor



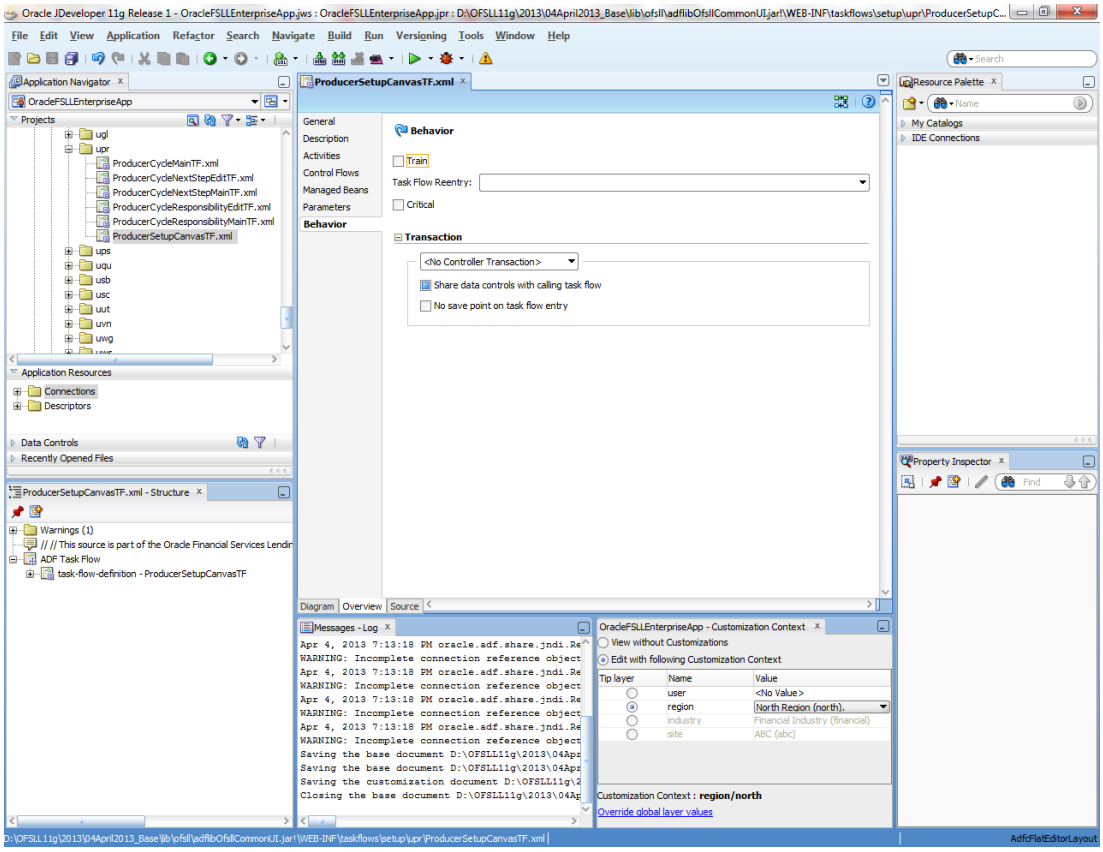
4.8 Editing Task Flows

You can use JDeveloper to implement customizations on the task flows that are used in your 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 you modify a bounded task flow, the changes apply to any ADF region that uses the task flow.

Task: Edit Task Flows

In JDeveloper, you use the task flow diagram editor to implement customizations on existing task flows. In the Application Navigator, right-click the task flow you want to customize, and choose **Open**. The page is displayed in the diagram editor, where you can make changes to the existing activities and control flow cases, or create new custom ones.



4.9 Creating Custom Business Components

You can use JDeveloper to extend your application by creating custom business components. When creating custom business components in JDeveloper, you must launch JDeveloper in the **Default** role. This role is used for creating new custom objects that you want to add to the application. You can use the same workspace that you created for customization. Then after you have created the custom business components, you 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 your business entities.

In JDeveloper, you can create entity objects using the Create Entity Object wizard, which you can launch from the New Gallery. In the Application Navigator, right-click the project you want to add the entity object to, 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, you can create view objects using the Create View Object wizard, which you can launch from the New Gallery. In the Application Navigator, right-click the project you want to add the view object to, 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, you can create application modules using the Create Application Module wizard, which you can launch from the New Gallery. In the Application Navigator, right-click the project you want to add the application module to, 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, you can create declarative validation rules 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 you want to provide validation for, click the Create new validator icon, and use the Add Validation Rule dialog to configure the rule.

4.10 Creating Custom Task Flows

You can use JDeveloper to create custom task flows that you can include in your 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

You can create a custom task flow in JDeveloper using the New Gallery, and then define its activities using the task flow diagram editor. In the Application Navigator, right-click the project you want to add the task flow to, 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, you'll specify the details about the type of task flow you want to create. When you click **OK**, the task flow is created and displayed in the diagram editor.

4.11 Creating Custom Pages

You can use JDeveloper to create custom pages that you can include in your application. When creating custom pages in JDeveloper, you must launch JDeveloper in the **Default** role.

When creating the page (or dropping a view activity onto a task flow), you can create the page 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 what you use when you want to use task flows as regions on a page. When you modify a JSF page fragment, the JSF pages that consume the page fragment are automatically updated.

After extending your application with custom pages, you will need 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

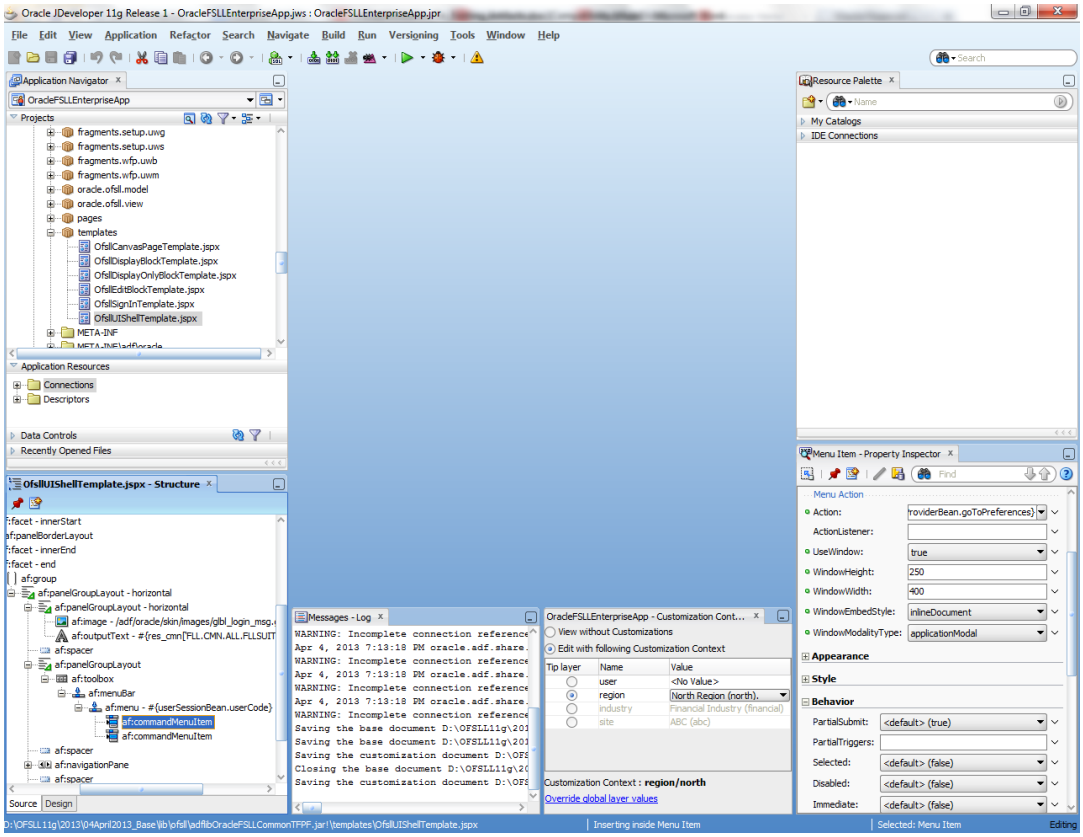
In JDeveloper, you can create pages either by double-clicking a view activity in a task flow or by using the New Gallery. In the Application Navigator, right-click the project you want to add the page to, and choose **New**. Then in the New Gallery, expand Web Tier, and click JSF/Facelets. Then select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

Task: Add a Custom Page to a Task Flow

If you created the page by double-clicking a view activity in a task flow, it is already added to the task flow. If you created it using the New Gallery, you can add it to a task flow by dragging the page from the Application Navigator and dropping it in the task flow diagram editor. Then you can 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, import the *adflibOracleFSLLCommonTFPF.jar* and drill down to the file */templates/OfsllUiShellTemplate.jspx*. This is the UI Shell template, which you can customize as necessary.



4.13 Customizing Security for ADF Application Artifacts

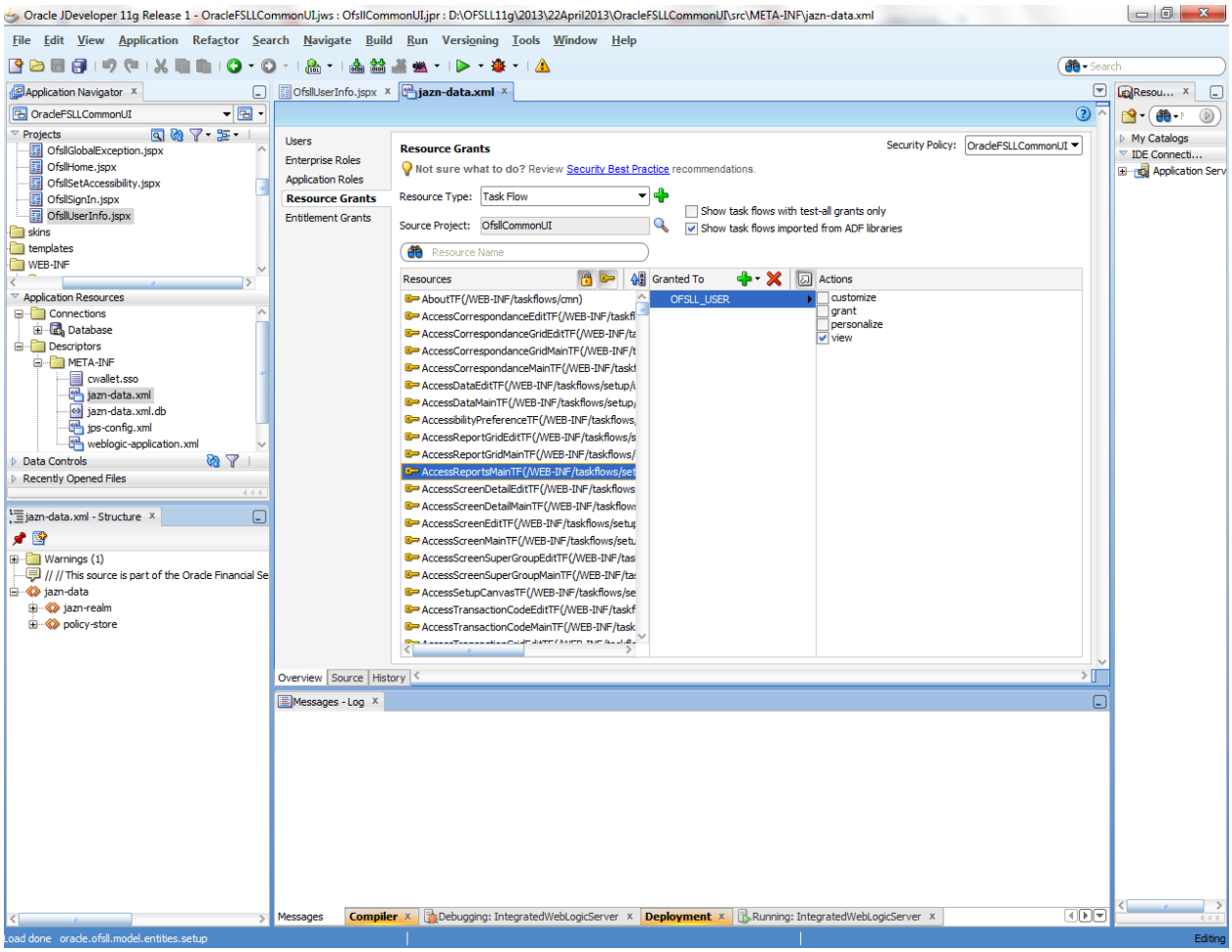
This chapter describes how to customize security for custom and extended business objects and related custom and extended application artifacts defined by Oracle Application Development Framework (Oracle ADF) in Oracle Fusion applications using Oracle JDeveloper.

This is applicable only for newly added Taskflows, Fragments, Command Buttons and Tabs.

Following Steps are to be followed to add the ADF Security:

1. Open JDeveloper in **Default** role and open the *OracleFSLLEnterpriseApp.jws* project

2. Edit *jazn-data.xml* file and select the *Resource* grants from Overview Editor



3. Select the option to import taskflows from ADF libraries

4. Enable the permissions for the newly added Taskflows.

Note

Tab permissions are to be set against *TabItemPermission* Resource Type.

Command buttons permissions are to be set against *ButtonItemPermission* Resource Type.

Ensure to follow the naming convention as per base resource keys.

4.14 Deploying ADF Customizations and Extensions

After you have customized existing artifacts, you can use JDeveloper 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 you implement. You can use this profile to package your customizations for deployment.

When you package customizations 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 you extend your application with new custom artifacts, you can use JDeveloper to package them into an ADF Library JAR and place them into the proper location within the application directory structure.

Task: Deploy the Customizations

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

When you deploy customizations 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 you have extended your application with new custom artifacts (or you are supplied with new artifacts), you must package these artifacts 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

4.15 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

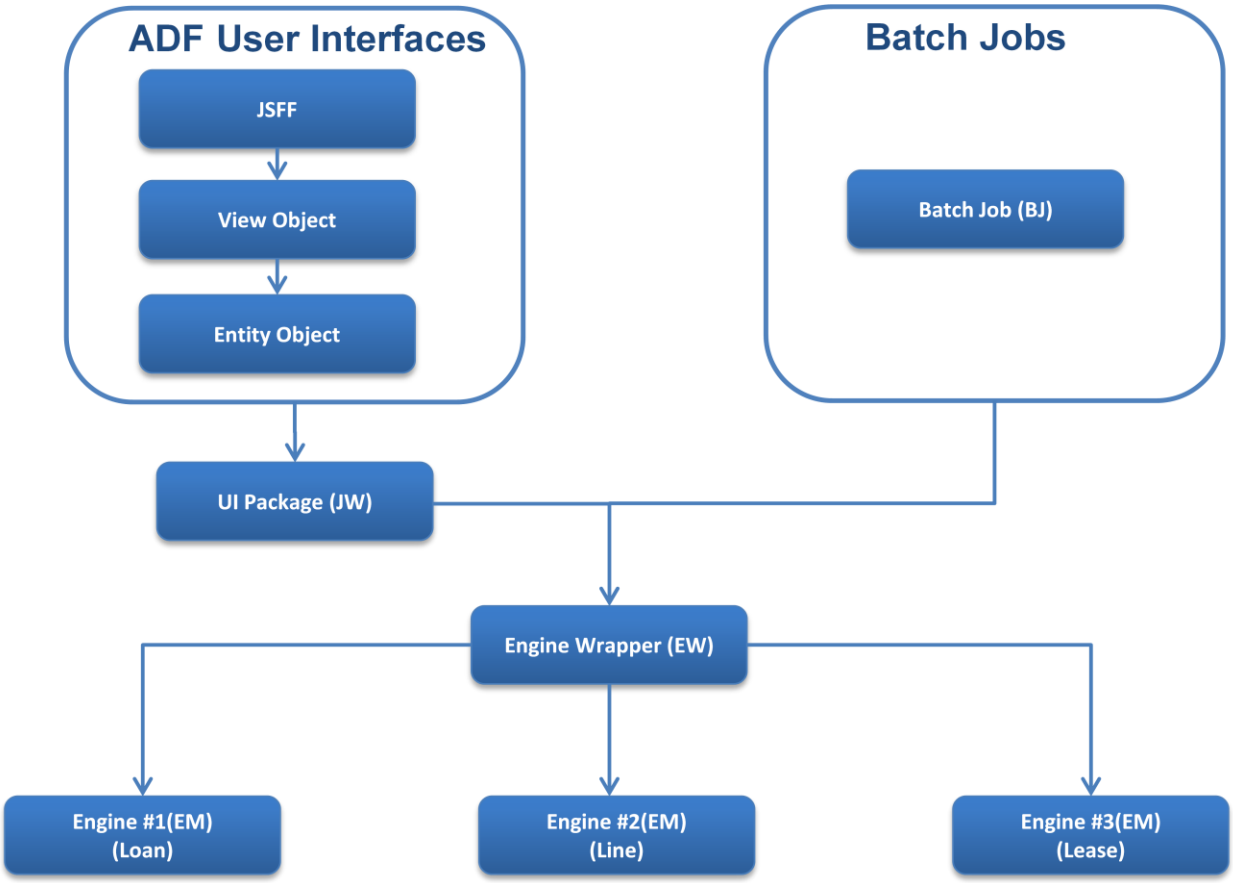
In *Customization Developer* role, the project creates the MAR deployment profile for customization deployment.

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.

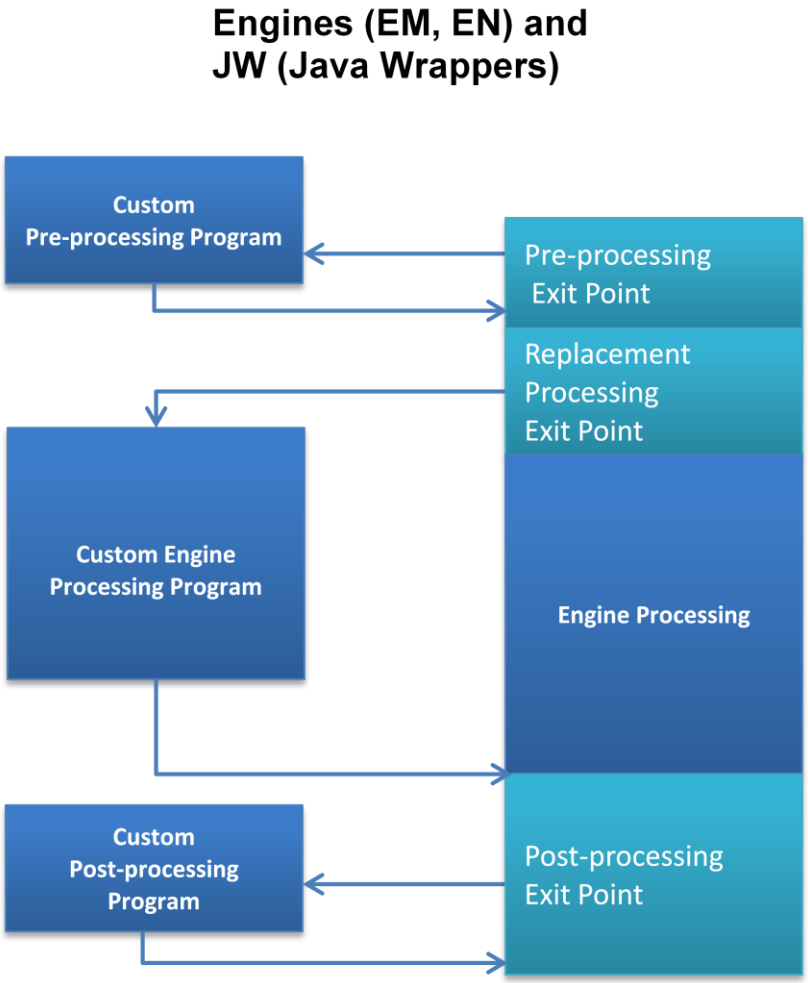


UI Java Wrapper (U*JW)

If you want to customize the java wrapper engine, follow the following steps:

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 your customized engine library and call it in the java wrapper Exit Points Package (EX).



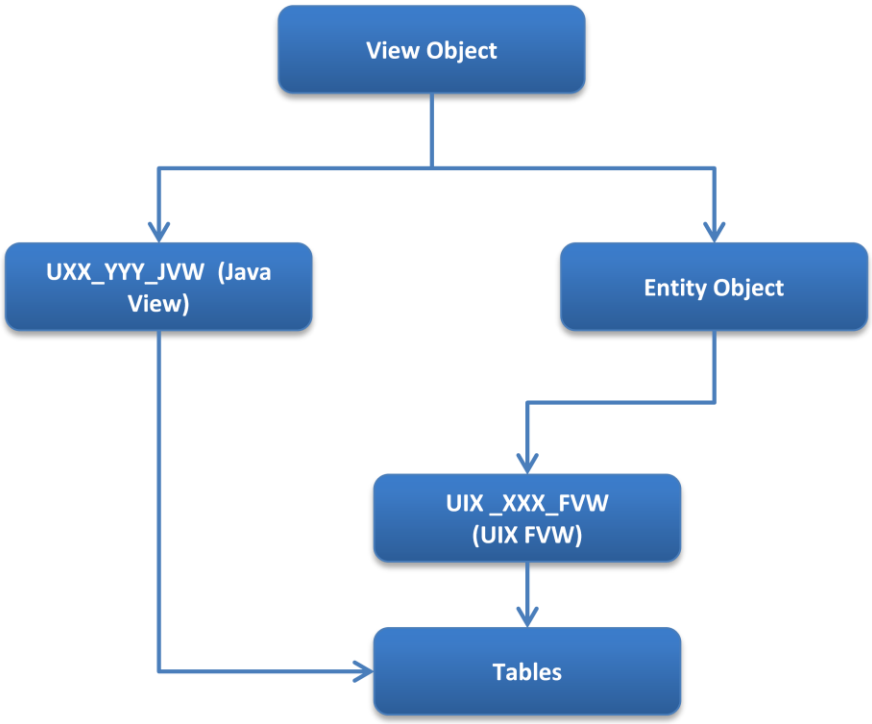
Business Logic Engine (Process)

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



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
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

Batch Job (BJ)

Batch Job **cannot** be customized, you will have to develop a new batch job.

Engine Wrapper (EW)

Engine Wrapper **cannot** be customized.

Main Engine (EM)

If you want to customize the main engine, follow the following steps:

- 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).

Engine Function (EN)

If you want to customize an engine function, follow the following steps:

- 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).

Engine View

If you want to customize an Engine View (EVW), follow the following steps:

- 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.

Common Features

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

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. We recommend you identify all the new customized seed data records with a customer identifier in the primary key.

Demo

- Oracle Financial Software Services Ltd. will provide the data first time for ease of setup customer can change/update this data

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

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

All seed data is stored in files and checked in the version control systems.

Seed data is send as merged statements in patch for changed (added or modified) data.

Developer's Tips

Suppose you want to customize account number generation that is different from what OFSLL generates. Requirements are 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 was to show the usage with replacement exit point and similar way “before” and “after” exit points can be used to extend the business logic functions.

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

6. BI Publisher Report

If you want to customize a report, follow the following steps:

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

7. Naming Convention for Customized Objects

Object	Naming Convention	Comment
New Table	<table_name>_xyz	Same Column Naming Convention
New View	<view_name>_xyz	Same 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>**	

Object	Naming Convention	Comment
New Report File Name	xyz<report_name>	
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 the following guidelines:

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.

Extensibility Guide
April [2013]
Version 14.0.0.0.0

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/