Oracle® Banking Enterprise Originations

UI Extensibility Guide

Release 2.10.0.0.0

F29510-01

April 2020

Oracle Banking Enterprise Originations UI Extensibility Guide, Release 2.10.0.0.0

F29510-01

Contents

Preface **14**

Audience 14

Documentation Accessibility 14

Related Documents 14

Conventions 14

1 About This Guide **17**

2 Objective and Scope **19**

2.1 Overview 19

2.2 Objective and Scope 19

2.2.1 Extensibility Objective 19

2.3 Complementary Artefacts 19

2.4 Out of Scope 20

3 Overview of Use Cases **21**

3.1 Extensibility Use Cases 21

3.1.1 ADF Screen Customization Using UI Extensions 21

3.1.2 ADF Screen Customization Using MDS 22

3.1.3 Print Receipt Functionality 22

4 ADF Screen Customizations Using UI Extensions **25**

4.1 UI Extension Interface 27

4.2 Default UI Extension 28

4.3 UI Extension Executor 29

4.4 Extension Configuration 32

4.5 Customization Examples 33

4.5.1 Replacing skin 33

4.5.2 Changing the logo in the branding bar 35

4.5.3 Modifying fonts 35

4.5.4 Modifying images 36

4.5.5 Graphics 36

4.5.6 Adding a simple field to a product screen 36

4.5.7 Adding a complex field popup to a product screen (popup, table, tree, region, tf) 37

4.5.8 Removing an existing field from a product screen 37

4.5.9 Making certain product optional product fields mandatory or optional 37

4.5.10 Adding a new column to an existing product grid 37

4.5.11 Hiding columns from an existing product grid 38

4.5.12 Graying out certain columns from an existing product grid 39

4.5.13 Modifying properties of product table (rows or tablesummary) 39

4.5.14 Adding a new section to an existing product screen 39

4.5.15 Hiding a section from a product screen 40

4.5.16 Adding a new tab to an existing product screen made of tabs 40

4.5.17 Hiding a tab from a product screen made of multiple tabs 41

4.5.18 Adding new buttons or links 41

4.5.19 Overriding / Customizing the product behaviour on certain actions like button clicks or tab-outs 42

4.5.20 Overriding the product validation pattern 42

4.5.21 Overriding the product lengths (min/max) 42

4.5.22 Disable / Enable certain product fields 42

4.5.23 Change certain product fields to read-only either on load or based on certain conditions 42

4.5.24 Change label of existing product fields 42

4.5.25 DC validation 43

4.5.26 LOV Extension– LOV Delegate Pattern 43
4.6 Using the JSFF Utils 44
4.6.1 How to Use JSFF Utils 44
4.6.2 Sample JSFF Utils Code Snippet 44
5 ADF Screen Customizations Using MDS 47
5.1 Seeded Customization Concepts 47
5.2 Customization Layer 48
5.3 Customization Class 48
5.4 Enabling Application for Seeded Customization 50
5.5 Customization Project 53
5.6 Customization Role and Context 54
5.7 Customization Layer Use Cases 56
5.7.1 Adding a UI Table Component to the Screen 56
5.7.2 Approvals Framework 70
5.7.3 Override the product managedBean 111
6 Human Task Screen Extension 113
6.1 Introduction 113
6.2 Custom CSS Skin 113
6.2.1 Create New ADF Skin 113
6.2.2 Apply New Skin 114
7 Receipt Printing 117
7.1 Prerequisite 117
7.1.1 Identify Node Element for Attributes in Print Receipt Template 117
7.1.2 Receipt Format Template (.rtf) 119
7.2 Configuration 120
7.2.1 Parameter Configuration in the BROPConfig.properties 120

7.2.2 Configuration in the ReceiptPrintReports.properties 121

7.3 Implementation 121

7.3.1 Default Nodes 122

7.4 Special Scenarios 122

8 Extensibility Usage – OBP Localization Pack 125

8.1 Localization Implementation Architectural Change 126

8.2 Customizing UI Layer 127

8.2.1 JDeveloper and Project Customization 127

8.2.2 Generic Project Creation 133

8.2.3 MAR Creation 133

8.3 Source Maintenance and Build 141

8.3.1 Source Check-ins to SVN 141

8.3.2 .mar files Generated during Build 142

8.3.3 adf-config.xml 142

8.4 Packaging and Deployment of Localization Pack 142

9 Deployment Guideline 145

9.1 Customized Project Jars 145

9.2 Database Objects 145

9.3 Extensibility Deployment 145

List of Figures

Figure 3–1 ADF Screen Extensions 21

Figure 3–2 ADF Screen Customization 22

Figure 3–3 Print Receipt Functionality 23

Figure 4–1 UI Extension Pre Hook and Post Hook Taskflow 26

Figure 4–2 Save Method in IntegrableTaskflowHelper 27

Figure 4–3 Example of UI Extension 28

Figure 4–4 Example of Default UI Extension 29

Figure 4–5 UI Extension Executor Class Taskflow 30

Figure 4–6 Example of UI Extension Executor Class 31

Figure 4–7 Example of UI Extension Executor Class 32

Figure 4–8 Replacing skin 34

Figure 4–9 Replacing skin 34

Figure 4–10 Example: Replacing skin 35

Figure 4–11 Replacing the logo 35

Figure 4–12 Example: To modify images 36

Figure 4–13 Example: To add a simple field to a product screen 37

Figure 4–14 Example: To remove an existing field from a region 37

Figure 4–15 Example: To add a new column to an existing prouct grid 38

Figure 4–16 Example: To hide columns from an existing product grid 39

Figure 4–17 Example: To modify the properties of product table 39

Figure 4–18 Example: To add a new section to an existing product screen 40

Figure 4–19 Example: To add a new tab to existing product screen made of tabs 41

Figure 4–20 Example: To hide a tab from a product screen made of multiple tabs 41

Figure 4–21 Example: To add new buttons or links 42

Figure 4–22 Example: To override the product validation pattern 42
Figure 4–23 LOV Extension– LOV Delegate Pattern 43
Figure 4–24 Sample Code Snippet 44
Figure 4–25 Example of JSFF Utils 44
Figure 5–1 Customization Application View 47
Figure 5–2 CustomizationLayerValues.xml 48
Figure 5–3 Customization Class 49
Figure 5–4 Implementation for the abstract methods of CustomizationClass 50
Figure 5–5 Enable Seeded Customizations 51
Figure 5–6 Adding com.ofss.fc.demo.ui.OptionCC.jar 52
Figure 5–7 Adding com.ofss.fc.demo.ui.OptionCC.OptionCC 52
Figure 5–8 Adf-config.xml 53
Figure 5–9 Customization Developer 54
Figure 5–10 Selecting Always Prompt for Role Selection on Start Up 55
Figure 5–11 View Customization Context 56
Figure 5–12 Adding a UI Table Component - Party Search screen 57
Figure 5–13 Adding a UI Table Component - Related Party screen 57
Figure 5–14 Creating Binding Bean Class 59
Figure 5–15 Create Event Consumer Class 60
Figure 5–16 Creating Managed Bean 60
Figure 5–17 Create Data Control 61
Figure 5–18 Adding View Object Binding to Page Definition - Add Tree Binding 62
Figure 5–19 Adding View Object Binding to Page Definition - Update Root Data Source 63
Figure 5–20 Page Data Binding Definition - Insert Item 64
Figure 5–21 Page Data Binding Definition - Create Action Binding 65
Figure 5–22 Edit Event Map 66

Figure 5–23 Event Map Editor 67
Figure 5–24 Add UI Components to Screen 68
Figure 5–25 Application Navigator 69
Figure 5–26 Party Search 70
Figure 5–27 Contact Point Screen 71
Figure 5–28 Create Table 72
Figure 5–29 Create Java Project 72
Figure 5–30 Create Domain Objects 73
Figure 5–31 Create Interface 73
Figure 5–32 Create Class 74
Figure 5–33 Set OBP Plugin Preferences 74
Figure 5–34 Set OBP Plugin Preferences 75
Figure 5–35 Set OBP Pugin Prefernces 76
Figure 5–36 Create Application Service 77
Figure 5–37 Application Service Classes Generated 77
Figure 5–38 Modify Data Transfer Object (DTO) 78
Figure 5–39 Generate Service and Facade Layer Sources 79
Figure 5–40 Modify ContactExpiryApplicationServiceSpi.java 80
Figure 5–41 Modify ContactExpiryApplicationServiceSpi.java 81
Figure 5–42 Modify ContactExpiryApplicationServiceSpi.java 82
Figure 5–43 Java Packages 82
Figure 5–44 Export Java Project as JAR 83
Figure 5–45 Create ContactExpiry.hbm.xml 84
Figure 5–46 Configure hostapplicationlayer.properties 84
Figure 5–47 Configure ProxyFacadeConfig.properties 85
Figure 5–48 Configure JSONServiceMap.properties 85

Figure 5–49 Create Model Project 86
Figure 5–50 Create Model Project - Configure Java Settings 87
Figure 5–51 Create Application Module 88
Figure 5–52 Set Package and Name of Application Module 89
Figure 5–53 Summary of Application Module Created 90
Figure 5–54 Create View Object 91
Figure 5–55 View Attribute 92
Figure 5–56 Application Module 93
Figure 5–57 Create View Object - Summary 94
Figure 5–58 Create View Controller Project 95
Figure 5–59 Name your Project 96
Figure 5–60 Libraries and Classpath 97
Figure 5–61 Dependencies 97
Figure 5–62 Create Maintenance State Action Interface 98
Figure 5–63 Create Update State Action Class 99
Figure 5–64 Create Update State Action Class 100
Figure 5–65 DemoContactPoint.java displays the View Objects 101
Figure 5–66 DemoCreateContactPoint / DemoUpdateContactPoint 102
Figure 5–67 Create Contact Expiry DTO 102
Figure 5–68 Value Change Event Handler for the Expiry Date UI Component 103
Figure 5–69 Value Change Event Handlers for Existing UI Components 103
Figure 5–70 Method to fetch Screen Data using Contact Expiry Proxy Service 104
Figure 5–71 Create Managed Bean 104
Figure 5–72 Create Event Customer Class 105
Figure 5–73 Create Data Control 106
Figure 5–74 Generated contactPoint.jsff.xml 107

Figure 5–75 Add an attributeValues binding 107
Figure 5–76 Create Attribute Binding 108
Figure 5–77 Add a methodAction binding 108
Figure 5–78 Create Action Binding 109
Figure 5–79 Select the Event Consumer Method 110
Figure 5–80 Generated contactPoint.jsff.xml 110
Figure 5–81 PI041 - Contact Point Screen 111
Figure 6–1 Sample trinidad-skins.xml 113
Figure 6–2 Package Structure 114
Figure 6–3 Sample Data 114
Figure 6–4 Sample Implementation 115
Figure 7–1 Input Property Files 117
Figure 7–2 Build Path of Utility 118
Figure 7–3 Utility Execution 119
Figure 7–4 Excel Generation 119
Figure 7–5 Receipt Format Template 120
Figure 7–6 Receipt Print Reports 121
Figure 7–7 Sample of Print Receipt 122
Figure 8–1 Perfection Capture Screen 125
Figure 8–2 Localization Implementation Architectural Change 126
Figure 8–3 Package Structure 127
Figure 8–4 Customization of the JDeveloper 128
Figure 8–5 Customization of the JDeveloper 128
Figure 8–6 Configure Design Time Customization layer 129
Figure 8–7 Enabling Seeded Customization 130
Figure 8–8 Library and Class Path 131

Figure 8–9 MDS Configuration 132
Figure 8–10 MDS Configuration 133
Figure 8–11 MAR Creation 134
Figure 8–12 MAR Creation - Application Properties 135
Figure 8–13 MAR Creation - Create Deployment Profile 136
Figure 8–14 MAR Creation - MAR File Selection 137
Figure 8–15 MAR Creation - Enter Details 138
Figure 8–16 MAR Creation - ADF Library Customization 139
Figure 8–17 MAR Creation - Edit File 140
Figure 8–18 MAR Creation - Application Assembly 141
Figure 8–19 Package Deployment 143
Figure 9–1 Extensibility Deployment 146

List of Tables

Table 8–1 Path Structure 141

Preface

This guide explains customization and extension of Oracle Banking Enterprise Originations.

This preface contains the following topics:

- Audience
- Documentation Accessibility
- Related Documents
- Conventions

Audience

This guide is intended for the users of Oracle Banking Enterprise Originations.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/us/corporate/accessibility/index.html.

Access to Oracle Support

Oracle customers have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/us/corporate/accessibility/support/index.html#info or visit http://www.oracle.com/us/corporate/accessibility/support/index.html#trs if you are hearing impaired.

Related Documents

For more information, see the following documentation:

- For installation and configuration information, see the Oracle Banking Enterprise Originations Localization Installation Guide - Silent Installation guide.
- For a comprehensive overview of security, see the Oracle Banking Enterprise Originations Security Guide.
- For the complete list of licensed products and the third-party licenses included with the license, see the Oracle Banking Enterprise Originations Licensing Guide.
- For information related to setting up a bank or a branch, and other operational and administrative functions, see the Oracle Banking Enterprise Originations Administrator Guide.
- For information on the functionality and features, see the respective Oracle Banking Enterprise Originations Functional Overview documents.
- For recommendations of secure usage of extensible components, see the Oracle Banking Enterprise Originations Secure Development Guide.

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

1 About This Guide

This guide is applicable for the following products:

- Oracle Banking Platform
- Oracle Banking Enterprise Originations
- Oracle Banking Enterprise Default Management

References to Oracle Banking Platform or OBP in this guide apply to all the above mentioned products.

2 Objective and Scope

This chapter defines the objective and scope of this document.

2.1 Overview

Oracle Banking Platform (OBP) is designed to help banks respond strategically to today's business challenges, while also transforming their business models and processes to reduce operating costs and improve productivity across both front and back offices. It is a one-stop solution for a bank that seeks to leverage Oracle Fusion experience for its core banking operations, across its retail and corporate offerings.

OBP provides a unified yet scalable IT solution for a bank to manage its data and end-to-end business operations with an enriched user experience. It comprises pre-integrated enterprise applications leveraging and relying on the underlying Oracle Technology Stack to help reduce in-house integration and testing efforts.

2.2 Objective and Scope

Most product development can be accomplished through highly flexible system parameters and business rules. Further competitive differentiation can be achieved through IT configuration and extension support. In OBP, additional business logic required for certain services is not always a part of the core product functionality but could be a client requirement. For these purposes, extension points and customization support have been provided in the application code which can be implemented by the bank and / or by partners, wherein the existing business logic can be added with or overridden by customized business logic. This way the time consuming activity of custom coding to enable region specific, site specific or bank specific customizations can be minimized.

2.2.1 Extensibility Objective

The broad guiding principles with respect to providing extensibility in OBP are summarized below:

- Strategic intent for enabling customers and partners to extend the application.
- Internal development uses the same principles for client specific customizations.
- Localization packs
- Extensions by Oracle Consultants, Oracle Partners, Banks or Bank Partners.
- Extensions through the addition of new functionality or modification of existing functionality.
- Planned focus on this area of the application. Hence, separate budgets specifically for this.
- Standards based - OBP leverages standard tools and technology
- Leverage large development pool for standards based technology.
- Developer tool sets provided as part of JDeveloper and Eclipse for productivity.

2.3 Complementary Artefacts

The document is a developer's extensibility guide and does not intend to work as a replacement of the functional or technical specification, which would be the primary resource covering the following:

- OBP Zen training course
- OBP installation and configuration
- OBP parameterization as part of implementation
- Functional solution and product user guide

References to plugin indicate the eclipse based OBP development plugin for relevant version of OBP being extended. The plugin is not a product GA artefact and is a means to assist development. Hence, the same is not covered under product support.

2.4 Out of Scope

The scope of extensibility does not intend to suggest that OBP is forward compatible.

3 Overview of Use Cases

The use cases that are covered in this document shall enable the developer in applying the discipline of extensibility to OBP. While the overall support for customizations is complete in most respects, the same is not a replacement for implementing a disciplined, thoughtful and well-designed approach towards implementing extensions and customizations to the product.

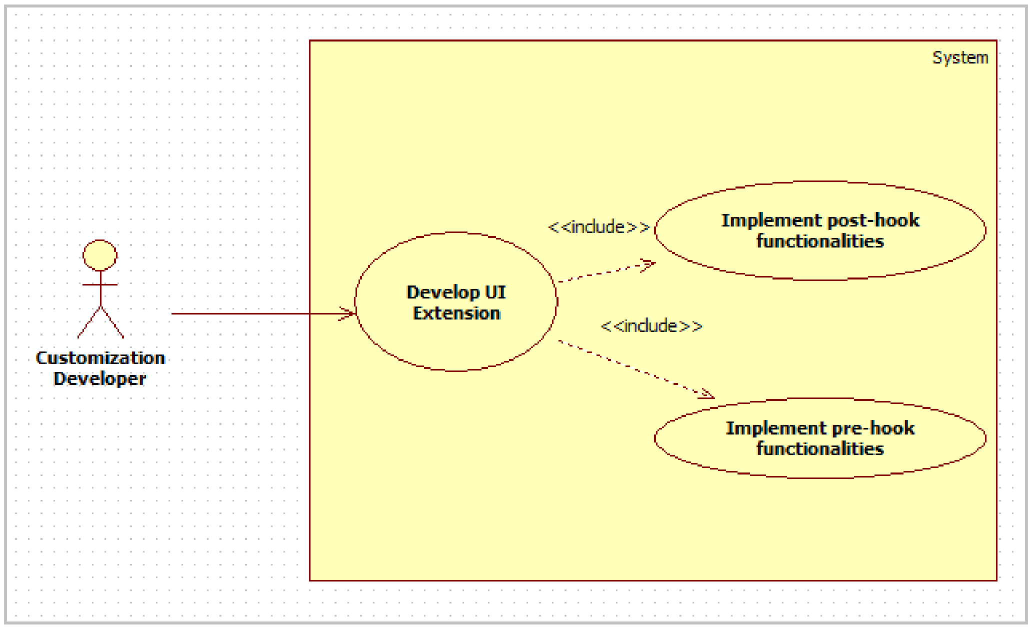
3.1 Extensibility Use Cases

This section gives an overview of the extensibility topics and customization use cases to be covered in this document. Each of these topics is detailed in the further sections.

3.1.1 ADF Screen Customization Using UI Extensions

In OBP, additional business logic or UI component changes might be required for certain ADF screen. This additional logic is not part of the core product functionality, but could be a client requirement. For this purpose, hooks have been provided in the application code wherein additional business logic can be added with custom business logic.

Figure 3–1 ADF Screen Extensions



Note

Screen changes can be implemented using the UI extensions or ADF Screen Customization. It is recommended to use the UI extensions where possible as migration path to higher release of OBP is easier.

UI Extension:

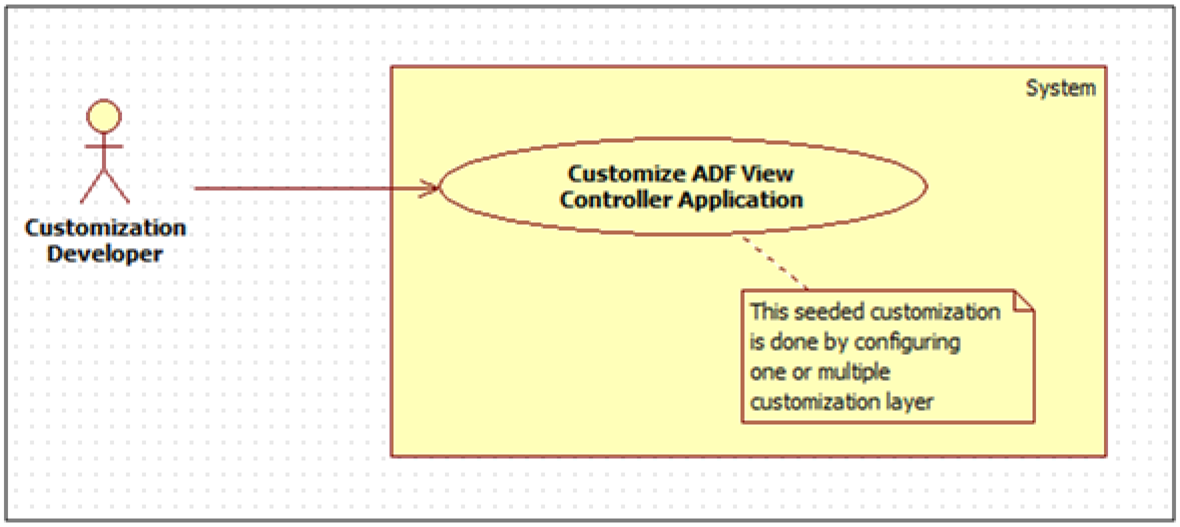
This hook resides in the ADF taskflow. This hook is present before as well as after the actual UI event execution. The additional business logic has to implement the interface **I<taskflow_name>UIExt** and extend and override the default implementation **Void<taskflow_name>UIExt** provided for the taskflow. Multiple implementations can be defined for a particular taskflow. The UI extensions executor invokes all the implementations defined for the particular taskflow both before and after the actual UI event execution.

3.1.2 ADF Screen Customization Using MDS

OBP application may need to be customized for certain additional requirements. However, since these additional requirements differ from client to client, and the base application functionality remains the same, the code to handle the additional requirements is kept separate from the code of the base application. For this purpose, Seeded Customizations (built using Oracle Meta-data Services framework) can be used to customize an application.

When designing seeded customizations for an application, one or more customization layers need to be specified. A customization layer is used to hold a set of customizations. A customization layer supports one or more customization layer value which specifies the set of customizations to apply at runtime.

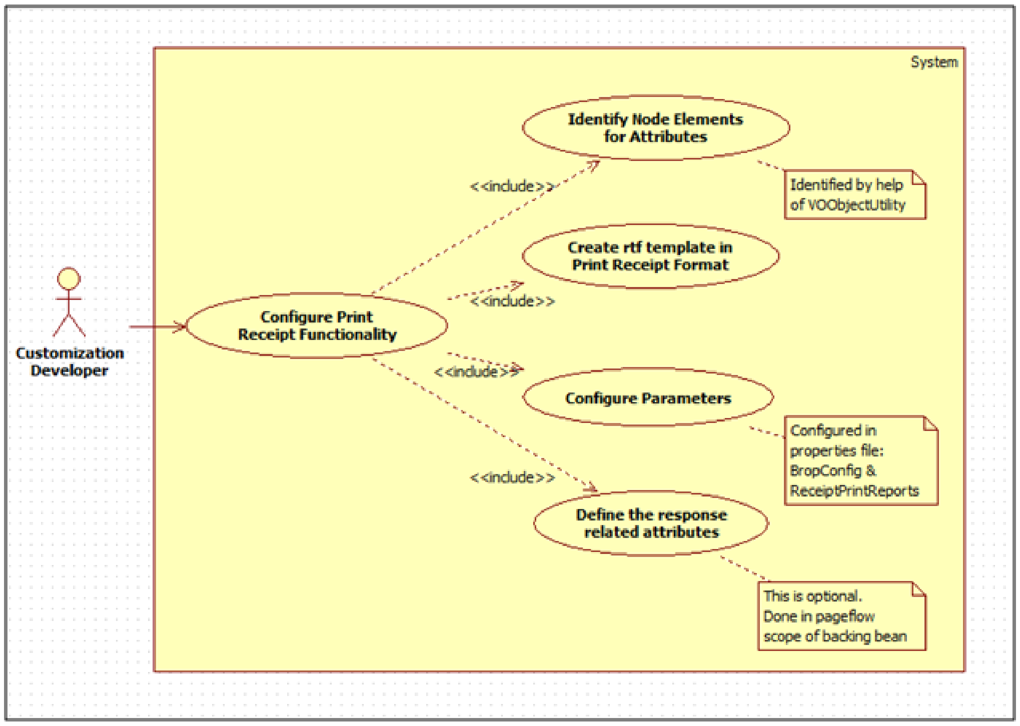
Figure 3–2 ADF Screen Customization


System
Customization Developer
Customize ADF View Controller Application
This seeded customization is done by configuring one or multiple customization layer

3.1.3 Print Receipt Functionality

OBP has many transaction screens in different modules where it is desired to print the receipt with different details about the transaction. This functionality provides the print receipt button on the top right corner of the screen which gets enabled on the completion of the transaction and can be used for printing of receipt of the transaction details.

Figure 3–3 Print Receipt Functionality


System
Customization Developer
Configure Print Receipt Functionality
<<include>>
Identify Node Elements for Attributes
Identified by help of VOObjectUtility
Create rtf template in Print Receipt Format
<<include>>
Configure Parameters
<<include>>
Configured in properties file: BropConfig & ReceiptPrintReports
<<include>>
Define the response related attributes
This is optional. Done in pageflow scope of backing bean

4 ADF Screen Customizations Using UI Extensions

This chapter describes how additional business logic can be added prior to (pre hook) and / or post the execution (post hook) of a particular UI event business logic on the UI side. Extension prior to a UI event execution may be required for the purposes of additional input validation, input manipulation, custom logging, and so on. A few examples in which the UI extensions in the form of pre and post hook are required are mentioned below.

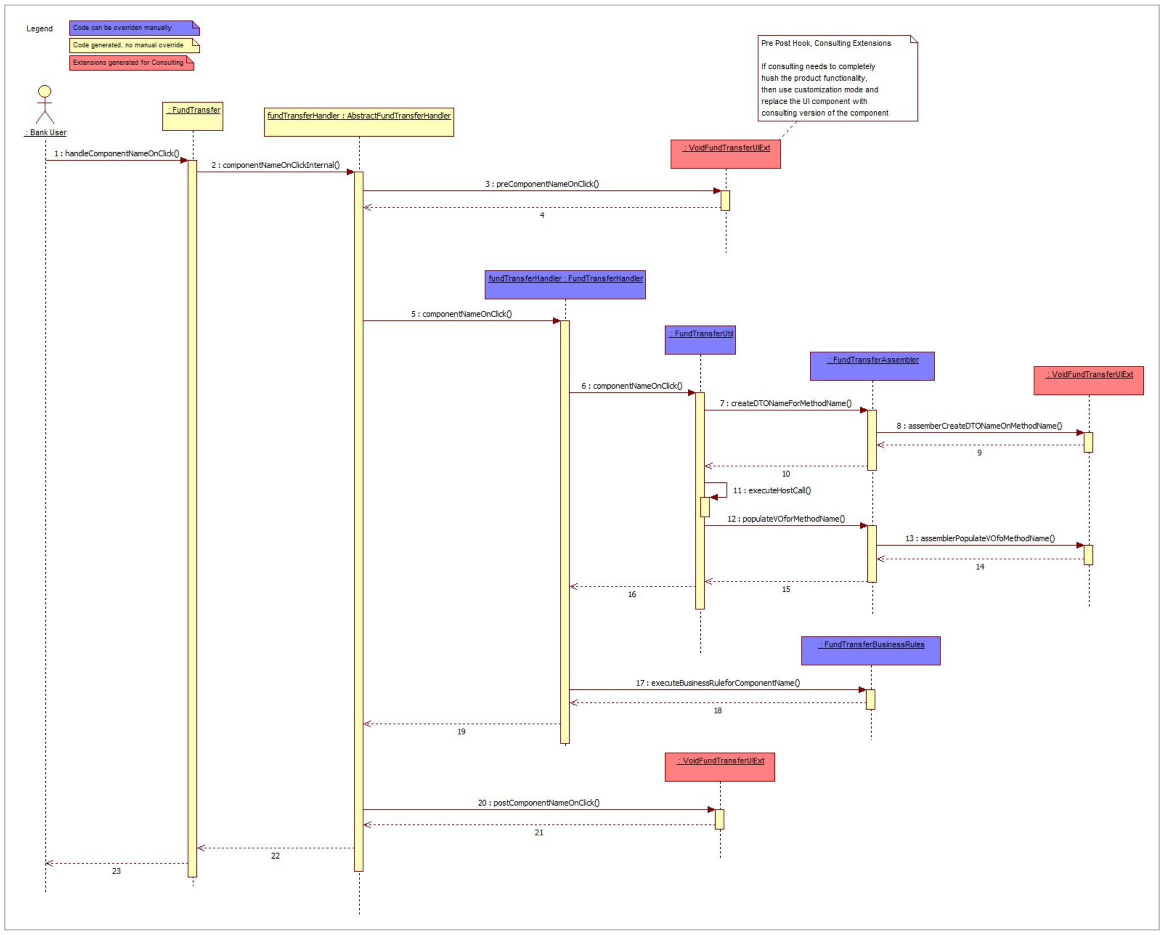
A UI extension in the form of a pre hook can be important in the following scenarios:

- Additional input validations
- Execution of business logic, which necessarily has to happen before going ahead with normal event execution
- Request manipulation prior to making host call

A UI extension in the form of a post hook can be important in the following scenarios:

- Output response manipulation
- Custom UI components rendering, changing to read only

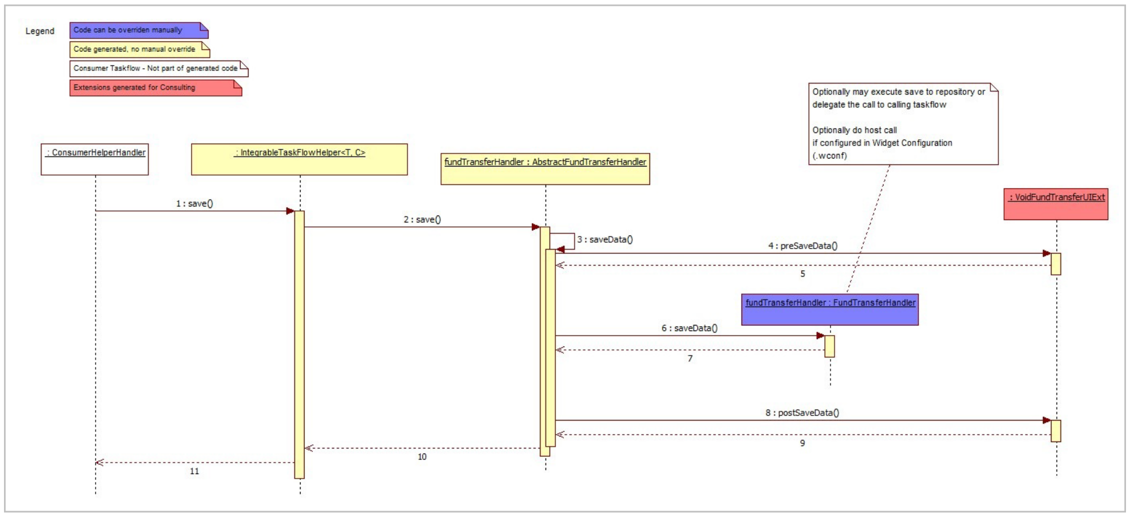
Figure 4–1 UI Extension Pre Hook and Post Hook Taskflow


Legend
Code can be overriden manually
Code generated, no manual override
Extensions generated for Consulting
Pre Post Hook, Consulting Extensions
If consulting needs to completely hush the product functionality, then use customization mode and replace the UI component with consulting version of the component
: Bank User
: FundTransfer
fundTransferHandler : AbstractFundTransferHandler
: VoidFundTransferUIExt
1 : handleComponentNameOnClick()
2 : componentNameOnClickInternal()
3 : preComponentNameOnClick()
4
fundTransferHandler : FundTransferHandler
5 : componentNameOnClick()
: FundTransferUtil
: FundTransferAssembler
: VoidFundTransferUIExt
6 : componentNameOnClick()
7 : createDTONameForMethodName()
8 : assemberCreateDTONameOnMethodName()
9
10
11 : executeHostCall()
12 : populateVOforMethodName()
13 : assemblerPopulateVOforMethodName()
14
15
16
: FundTransferBusinessRules
17 : executeBusinessRuleforComponentName()
18
19
: VoidFundTransferUIExt
20 : postComponentNameOnClick()
21
22
23

The pre hook is provided after the invocation of *UIevent* call inside the Abstract Taskflow Handler. The extension method is provided with the ADF event and the Taskflow Handler Instance as parameters. The handler instance may be required in such cases where the VO attributes or the UI components need to be accessed as a part of the customization.

The post hook is provided after the event business logic. Similar parameters are provided in the post extension. Hooks are provided in handler and assembler methods, for taskflows using the Integrable Taskflow framework. Hooks are provided in backing bean methods for all other taskflows.

Figure 4–2 Save Method in IntegrableTaskflowHelper


Legend
Code can be overriden manually
Code generated, no manual override
Consumer Taskflow - Not part of generated code
Extensions generated for Consulting
Optionally may execute save to repository or delegate the call to calling taskflow
Optionally do host call if configured in Widget Configuration (.wconf)
: ConsumerHelperHandler
: IntegrableTaskFlowHelper<T, C>
fundTransferHandler : AbstractFundTransferHandler
: VoidFundTransferUIExt
fundTransferHandler : FundTransferHandler
1 : save()
2 : save()
3 : saveData()
4 : preSaveData()
5
6 : saveData()
7
8 : postSaveData()
9
10
11

For taskflows implementing the ADF Integrable Taskflow Framework, the pre and post hooks are provided for the common Integrable taskflow helper methods. Refer to the above sequence diagram for the Save method in IntegrableTaskflowHelper.

The following sections detail the important concepts which should be understood for extending in this UI layer.

4.1 UI Extension Interface

The OBP ADF Taskflow Generator generates an interface for the extensions of a particular taskflow. The interface name is of the form **I<Taskflow_Name>UIExt**. This interface has a pair of pre and post method definitions for each public method present in the Abstract Taskflow Handler and the Integrable Taskflow Helper. The signatures of these methods are:

```
public void pre<Method_Name>(<Method_Parameters>) throws
FatalException;
public void post<Method_Name>(<Method_Parameters>) throws
FatalException;
```

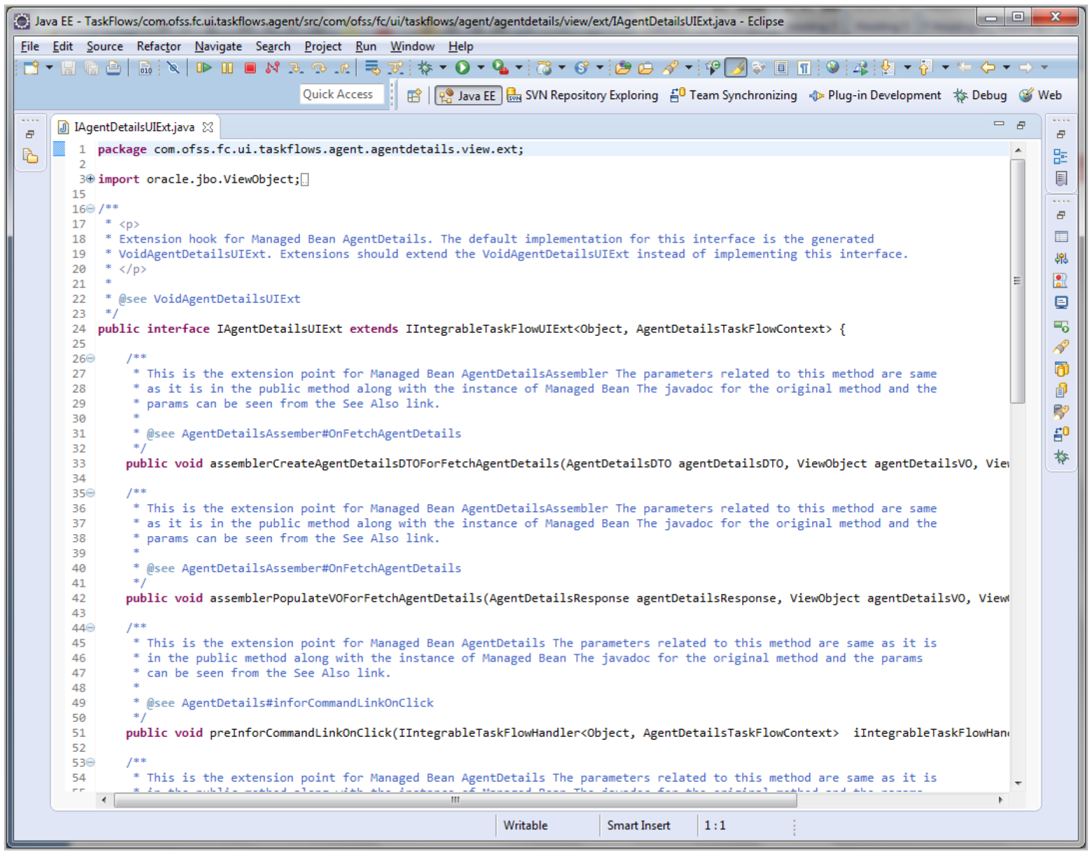
A single method is provided for Integrable Assembler. The signature as below:

```
public void assembler<Method_Name>(<Method_Parameters>) throws
FatalException;
```

A UI extension class has to implement this interface. The pre method of the extension is executed before the actual method and the post method of the extension is executed after the method.

The return type for certain methods are boolean (for example, public boolean preValidateData).

Figure 4–3 Example of UI Extension


Java EE - TaskFlows/com.ofss.fc.ui.taskflows.agent/src/com/ofss/fc/ui/taskflows/agent/agentdetails/view/ext/IAgentDetailsUIExt.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
SVN Repository Exploring
Team Synchronizing
Plug-in Development
Debug
Web
IAgentDetailsUIExt.java
Writable
Smart Insert
1 : 1

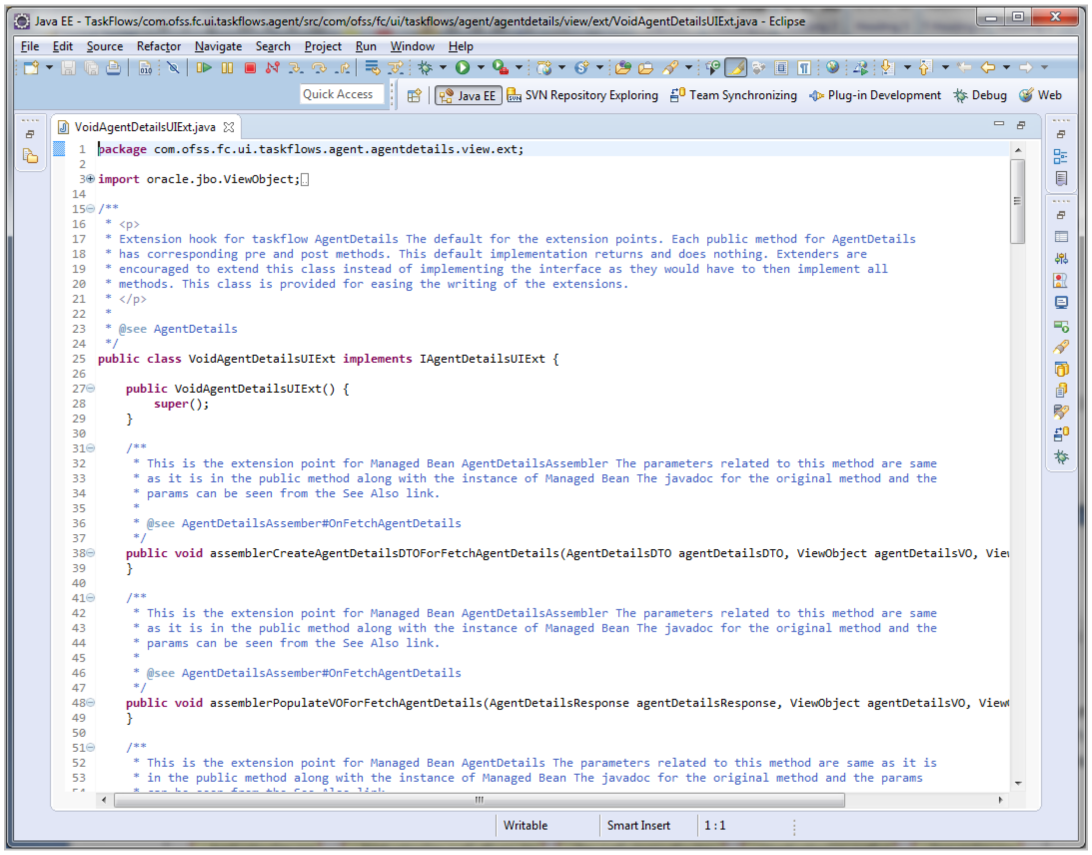
```
package com.ofss.fc.ui.taskflows.agent.agentdetails.view.ext;

import oracle.jbo.ViewObject;

/**
 * <p>
 * Extension hook for Managed Bean AgentDetails. The default implementation for this interface is the generated
 * VoidAgentDetailsUIExt. Extensions should extend the VoidAgentDetailsUIExt instead of implementing this interface.
 * </p>
 *
 * @see VoidAgentDetailsUIExt
 */
public interface IAgentDetailsUIExt extends IIntegrableTaskFlowUIExt<Object, AgentDetailsTaskFlowContext> {

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerCreateAgentDetailsDTOForFetchAgentDetails(AgentDetailsDTO agentDetailsDTO, ViewObject agentDetailsVO, Vie

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerPopulateVOForFetchAgentDetails(AgentDetailsResponse agentDetailsResponse, ViewObject agentDetailsVO, View

    /**
     * This is the extension point for Managed Bean AgentDetails The parameters related to this method are same as it is
     * in the public method along with the instance of Managed Bean The javadoc for the original method and the params
     * can be seen from the See Also link.
     *
     * @see AgentDetails#inforCommandLinkOnClick
     */
    public void preInforCommandLinkOnClick(IIntegrableTaskFlowHandler<Object, AgentDetailsTaskFlowContext>  iIntegrableTaskFlowHan

    /**
     * This is the extension point for Managed Bean AgentDetails The parameters related to this method are same as it is
```

4.2 Default UI Extension

The OBP plug-in generates a default extension for a particular taskflow in the form of the class **Void<Taskflow_Name>UIExt**. This class implements the aforementioned UI extension interface without any business logic, that is, the implemented methods are empty.

The default extension is a useful and convenient mechanism to implement the pre and / or post extension hooks for specific methods of a taskflow. Instead of implementing the entire interface, one should extend the default extension class and override only the required methods with the additional business logic. Product developers DO NOT implement any logic, including product extension logic, inside the default extension classes. This is because the classes are auto-generated, reserved for product use, and get overwritten as a part of a bulk generation process.

Figure 4–4 Example of Default UI Extension


Java EE - TaskFlows/com.ofss.fc.ui.taskflows.agent/src/com/ofss/fc/ui/taskflows/agent/agentdetails/view/ext/VoidAgentDetailsUIExt.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
SVN Repository Exploring
Team Synchronizing
Plug-in Development
Debug
Web
VoidAgentDetailsUIExt.java
Writable
Smart Insert
1 : 1

```
package com.ofss.fc.ui.taskflows.agent.agentdetails.view.ext;

import oracle.jbo.ViewObject;

/**
 * <p>
 * Extension hook for taskflow AgentDetails The default for the extension points. Each public method for AgentDetails
 * has corresponding pre and post methods. This default implementation returns and does nothing. Extenders are
 * encouraged to extend this class instead of implementing the interface as they would have to then implement all
 * methods. This class is provided for easing the writing of the extensions.
 * </p>
 *
 * @see AgentDetails
 */
public class VoidAgentDetailsUIExt implements IAgentDetailsUIExt {

    public VoidAgentDetailsUIExt() {
        super();
    }

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerCreateAgentDetailsDTOForFetchAgentDetails(AgentDetailsDTO agentDetailsDTO, ViewObject agentDetailsVO, Vie
    }

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerPopulateVOForFetchAgentDetails(AgentDetailsResponse agentDetailsResponse, ViewObject agentDetailsVO, View
    }

    /**
     * This is the extension point for Managed Bean AgentDetails The parameters related to this method are same as it is
     * in the public method along with the instance of Managed Bean The javadoc for the original method and the params
```

4.3 UI Extension Executor

The OBP plug-in for Eclipse generates a UI extension executor interface and an implementation for the executor interface. The naming convention for the generated executor classes which enable "extension chaining" is shown below:

```
Interface : I<Taskflow_Name>UIExtExecutor
Implementation : < Taskflow_Name >UIExtExecutor
```

The UI extension executor class, on load, creates an instance each of all the extensions defined in the UI extensions configuration. If no extensions are defined for a particular service, the executor creates an instance of the default extension for the taskflow. The executor also has a pair of pre and post methods for each method. These methods in turn call the corresponding methods of all the extension classes defined for the taskflow.

Figure 4–5 UI Extension Executor Class Taskflow

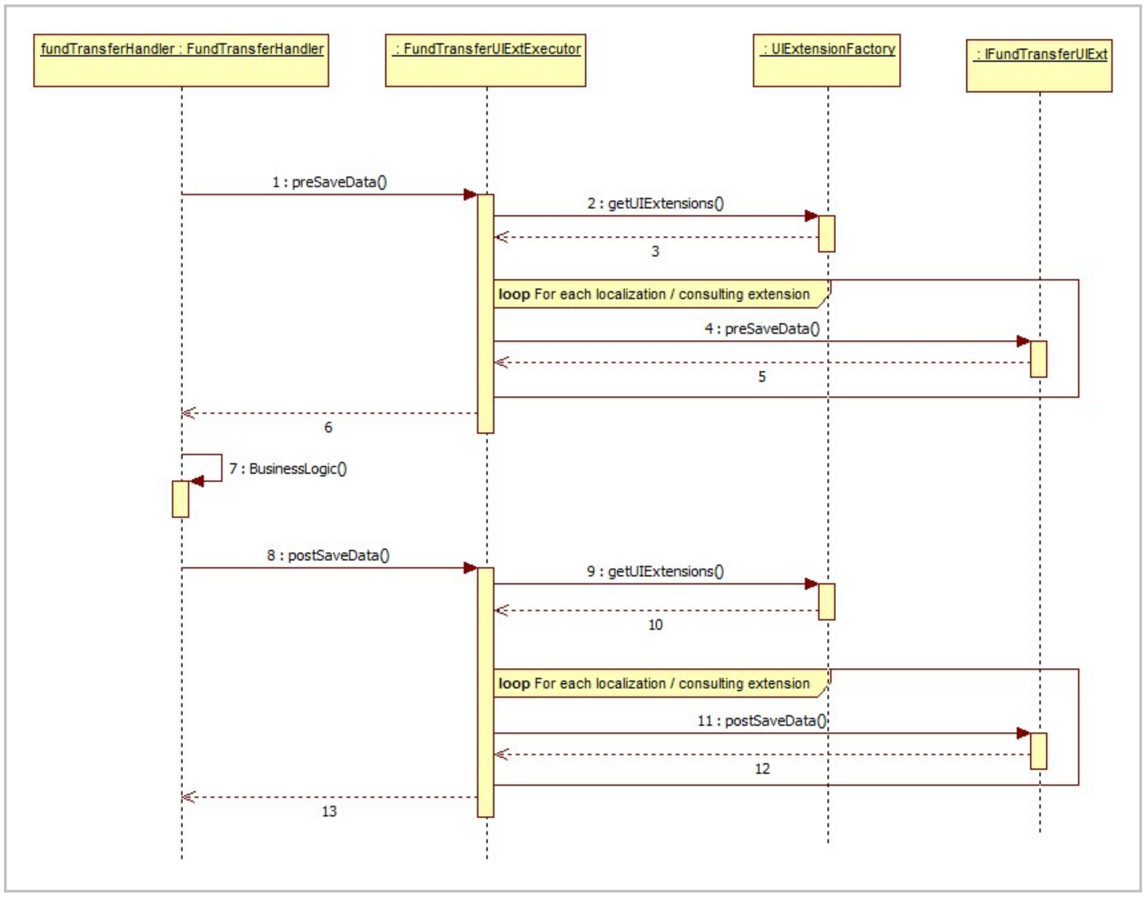
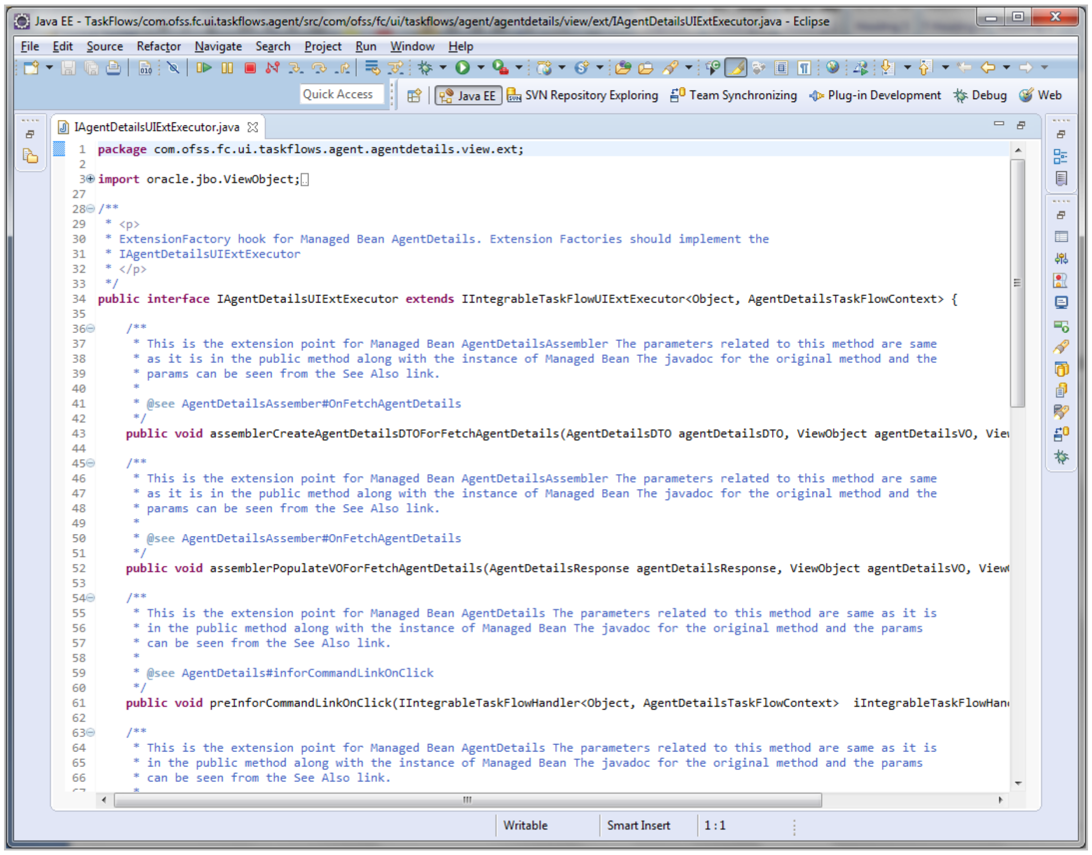
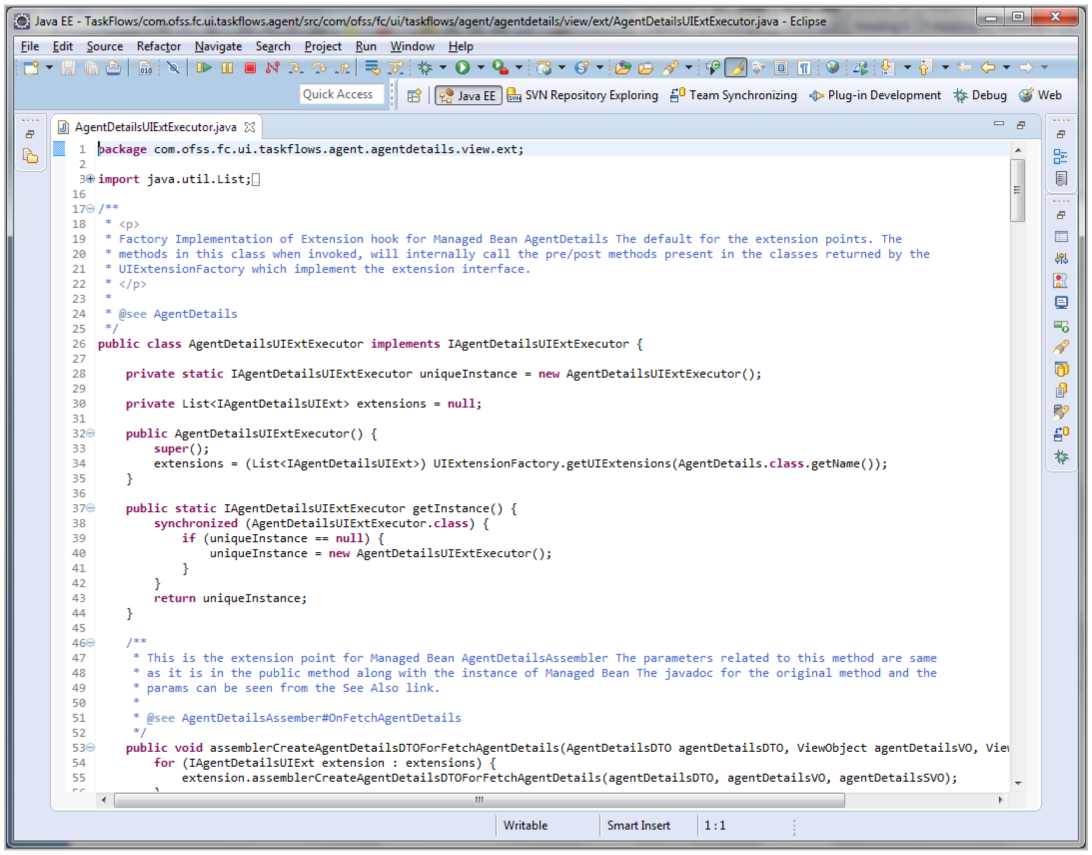


Figure 4–6 Example of UI Extension Executor Class


Java EE - TaskFlows/com.ofss.fc.ui.taskflows.agent/src/com/ofss/fc/ui/taskflows/agent/agentdetails/view/ext/IAgentDetailsUIExtExecutor.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
SVN Repository Exploring
Team Synchronizing
Plug-in Development
Debug
Web
IAgentDetailsUIExtExecutor.java
Writable
Smart Insert
1 : 1

```
package com.ofss.fc.ui.taskflows.agent.agentdetails.view.ext;

import oracle.jbo.ViewObject;

/**
 * <p>
 * ExtensionFactory hook for Managed Bean AgentDetails. Extension Factories should implement the
 * IAgentDetailsUIExtExecutor
 * </p>
 */
public interface IAgentDetailsUIExtExecutor extends IIntegrableTaskFlowUIExtExecutor<Object, AgentDetailsTaskFlowContext> {

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerCreateAgentDetailsDTOForFetchAgentDetails(AgentDetailsDTO agentDetailsDTO, ViewObject agentDetailsVO, Vie

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerPopulateVOForFetchAgentDetails(AgentDetailsResponse agentDetailsResponse, ViewObject agentDetailsVO, View

    /**
     * This is the extension point for Managed Bean AgentDetails The parameters related to this method are same as it is
     * in the public method along with the instance of Managed Bean The javadoc for the original method and the params
     * can be seen from the See Also link.
     *
     * @see AgentDetails#inforCommandLinkOnClick
     */
    public void preInforCommandLinkOnClick(IIntegrableTaskFlowHandler<Object, AgentDetailsTaskFlowContext>  iIntegrableTaskFlowHan

    /**
     * This is the extension point for Managed Bean AgentDetails The parameters related to this method are same as it is
     * in the public method along with the instance of Managed Bean The javadoc for the original method and the params
     * can be seen from the See Also link.
```

Figure 4–7 Example of UI Extension Executor Class


Java EE - TaskFlows/com.ofss.fc.ui.taskflows.agent/src/com/ofss/fc/ui/taskflows/agent/agentdetails/view/ext/AgentDetailsUIExtExecutor.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Java EE
SVN Repository Exploring
Team Synchronizing
Plug-in Development
Debug
Web
AgentDetailsUIExtExecutor.java
Writable
Smart Insert
1:1

```
package com.ofss.fc.ui.taskflows.agent.agentdetails.view.ext;

import java.util.List;

/**
 * <p>
 * Factory Implementation of Extension hook for Managed Bean AgentDetails The default for the extension points. The
 * methods in this class when invoked, will internally call the pre/post methods present in the classes returned by the
 * UIExtensionFactory which implement the extension interface.
 * </p>
 *
 * @see AgentDetails
 */
public class AgentDetailsUIExtExecutor implements IAgentDetailsUIExtExecutor {

    private static IAgentDetailsUIExtExecutor uniqueInstance = new AgentDetailsUIExtExecutor();

    private List<IAgentDetailsUIExt> extensions = null;

    public AgentDetailsUIExtExecutor() {
        super();
        extensions = (List<IAgentDetailsUIExt>) UIExtensionFactory.getUIExtensions(AgentDetails.class.getName());
    }

    public static IAgentDetailsUIExtExecutor getInstance() {
        synchronized (AgentDetailsUIExtExecutor.class) {
            if (uniqueInstance == null) {
                uniqueInstance = new AgentDetailsUIExtExecutor();
            }
        }
        return uniqueInstance;
    }

    /**
     * This is the extension point for Managed Bean AgentDetailsAssembler The parameters related to this method are same
     * as it is in the public method along with the instance of Managed Bean The javadoc for the original method and the
     * params can be seen from the See Also link.
     *
     * @see AgentDetailsAssember#OnFetchAgentDetails
     */
    public void assemblerCreateAgentDetailsDTOForFetchAgentDetails(AgentDetailsDTO agentDetailsDTO, ViewObject agentDetailsVO, Vie
        for (IAgentDetailsUIExt extension : extensions) {
            extension.assemblerCreateAgentDetailsDTOForFetchAgentDetails(agentDetailsDTO, agentDetailsVO, agentDetailsSVO);
```

4.4 Extension Configuration

The extension classes that implement the extension interface are mapped to the taskflow with the help of seed data in **FLX_FW_CONFIG_ALL_B**.

Following is a sample implementation.

Single Extension Class

```
insert into
FLX_FW_CONFIG_ALL_B(CATEGORY_ID,PROP_ID,PROP_VALUE,PROP_
COMMENTS,OBJECT_VERSION_NUMBER,CREATED_BY,CREATION_DATE,LAST_
UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS_FLAG,FACTORY_SHIPPED_
FLAG)
values
('UIExtensions','com.ofss.fc.ui.taskflows.account.accountholderpre
ferencesetup.view.backing.AccountHolderPreferenceSetup','com.ofss.
fc.lz.au.ui.taskflows.account.accountholderpreferencesetup.view.ex
t.RegionalAccountHolderPreferenceSetupUIExt','',1,'ofssuser',SYSDA
TE,'ofssuser',SYSDATE,'A','y');
```

Multiple Extension Classes

```
insert into
```

```
FLX_FW_CONFIG_ALL_B(CATEGORY_ID,PROP_ID,PROP_VALUE,PROP_
COMMENTS,OBJECT_VERSION_NUMBER,CREATED_BY,CREATION_DATE,LAST_
UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS_FLAG,FACTORY_SHIPPED_
FLAG)
values
('UIExtensions','com.ofss.fc.ui.taskflows.account.accountholderpre
ferencesetup.view.backing.AccountHolderPreferenceSetup','com.ofss.
fc.lz.au.ui.taskflows.account.accountholderpreferencesetup.view.ex
t.RegionalAccountHolderPreferenceSetupUIExt','',1,'ofssuser',SYSDA
TE,'ofssuser',SYSDATE,'A','y');
insert into
FLX_FW_CONFIG_ALL_B(CATEGORY_ID,PROP_ID,PROP_VALUE,PROP_
COMMENTS,OBJECT_VERSION_NUMBER,CREATED_BY,CREATION_DATE,LAST_
UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS_FLAG,FACTORY_SHIPPED_
FLAG)
values
('UIExtensions','com.ofss.fc.ui.taskflows.account.accountholderpre
ferencesetup.view.backing.AccountHolderPreferenceSetup','com.ofss.
fc.lz.au.ui.taskflows.account.accountholderpreferencesetup.view.ex
t.RegionalAccountHolderPreferenceSetupUIExtForUseCase1','',1,'ofss
user',SYSDATE,'ofssuser',SYSDATE,'A','y');
insert into
FLX_FW_CONFIG_ALL_B(CATEGORY_ID,PROP_ID,PROP_VALUE,PROP_
COMMENTS,OBJECT_VERSION_NUMBER,CREATED_BY,CREATION_DATE,LAST_
UPDATED_BY,LAST_UPDATED_DATE,OBJECT_STATUS_FLAG,FACTORY_SHIPPED_
FLAG)
values
('UIExtensions','com.ofss.fc.ui.taskflows.account.accountholderpre
ferencesetup.view.backing.AccountHolderPreferenceSetup','com.ofss.
fc.lz.au.ui.taskflows.account.accountholderpreferencesetup.view.ex
t.RegionalAccountHolderPreferenceSetupUIExtForUseCase2','',1,'ofss
user',SYSDATE,'ofssuser',SYSDATE,'A','y');
```

It is possible to configure multiple implementations of pre or post extensions for a taskflow in this layer. This is achieved with the help of the extension executor. It has the capability to loop through a set of extension implementations, which conform to the extension interface supported by the taskflow.

4.5 Customization Examples

Following are some examples of customization.

4.5.1 Replacing skin

Colours are maintained as a variable in the css lib files of the respective modules. Skin can be replaced to change the colours.

Replace skin: inside preCustomBranding()

@Override

public void preCustomBranding(Main main) {

/*setting skin */

FacesContext fc = FacesContext.getCurrentInstance();

ELContext elc = fc.getELContext();

String skinId = "skyros";

ExpressionFactory exprFact = fc.getApplication().getExpressionFactory();

ValueExpression ve = exprFact.createValueExpression(elc, "#{sessionScope.skinFamily}", Object.class);

ve.setValue(elc, skinId);

/* setting fonts */

main.setFontPath("/css/lato.css");

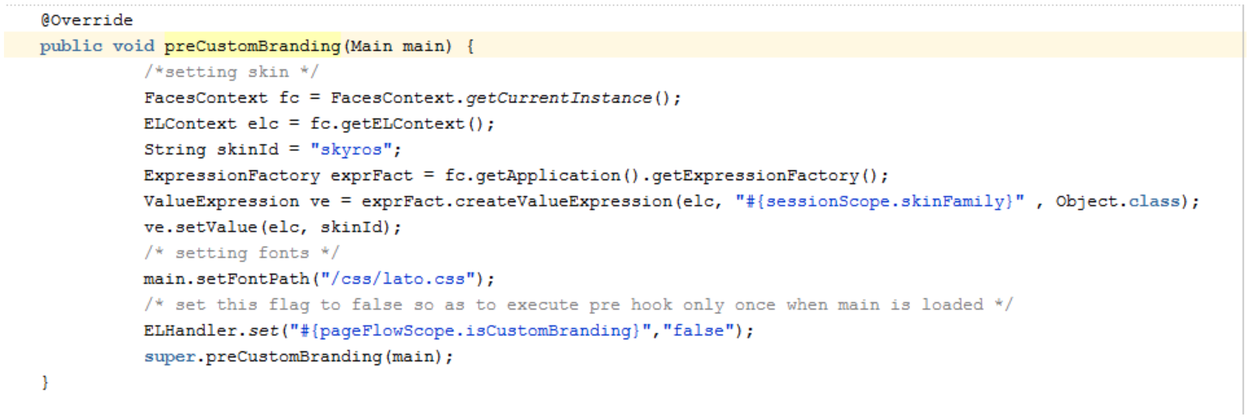
/* set this flag to false so as to execute pre hook only once when main is loaded */

ELHandler.set("#{pageFlowScope.isCustomBranding}","false");

super.preCustomBranding(main);

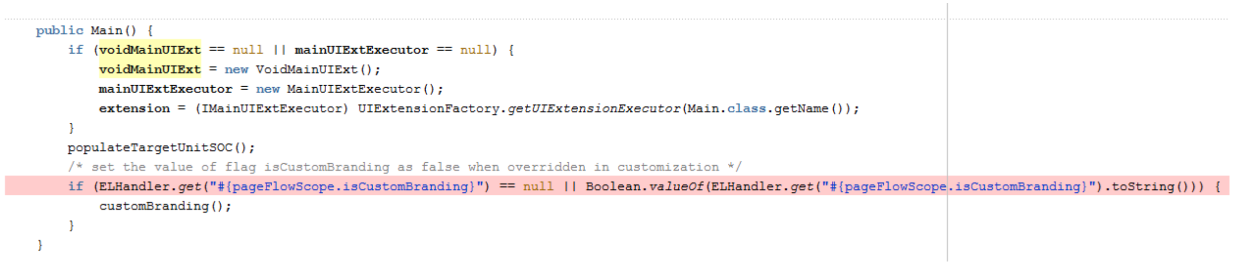
}

Figure 4–8 Replacing skin



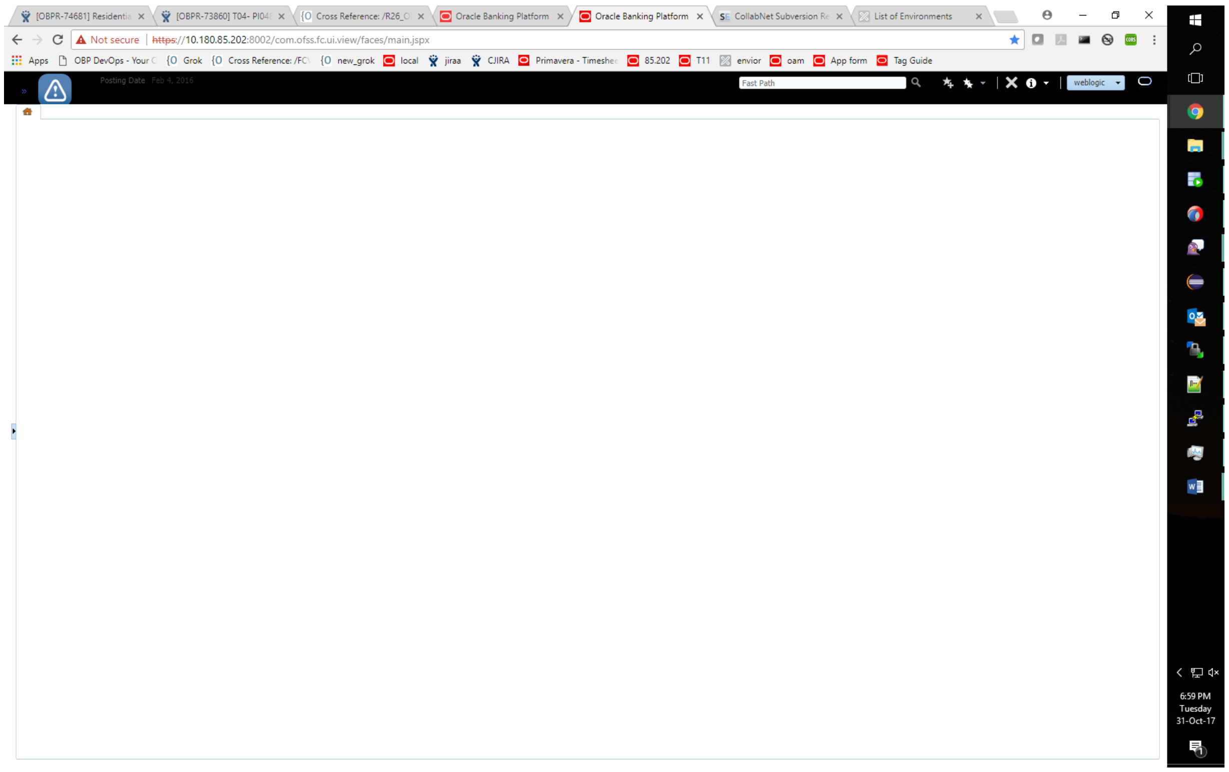
```
    @Override
    public void preCustomBranding(Main main) {
            /*setting skin */
            FacesContext fc = FacesContext.getCurrentInstance();
            ELContext elc = fc.getELContext();
            String skinId = "skyros";
            ExpressionFactory exprFact = fc.getApplication().getExpressionFactory();
            ValueExpression ve = exprFact.createValueExpression(elc, "#{sessionScope.skinFamily}" , Object.class);
            ve.setValue(elc, skinId);
            /* setting fonts */
            main.setFontPath("/css/lato.css");
            /* set this flag to false so as to execute pre hook only once when main is loaded */
            ELHandler.set("#{pageFlowScope.isCustomBranding}","false");
            super.preCustomBranding(main);
    }
```

Figure 4–9 Replacing skin



```
    public Main() {
        if (voidMainUIExt == null || mainUIExtExecutor == null) {
            voidMainUIExt = new VoidMainUIExt();
            mainUIExtExecutor = new MainUIExtExecutor();
            extension = (IMainUIExtExecutor) UIExtensionFactory.getUIExtensionExecutor(Main.class.getName());
        }
        populateTargetUnitSOC();
        /* set the value of flag isCustomBranding as false when overridden in customization */
        if (ELHandler.get("#{pageFlowScope.isCustomBranding}") == null || Boolean.valueOf(ELHandler.get("#{pageFlowScope.isCustomBranding}").toString())) {
            customBranding();
        }
    }
```

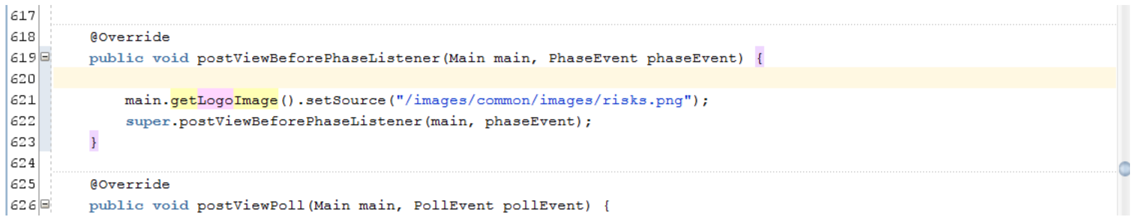
Figure 4–10 Example: Replacing skin



4.5.2 Changing the logo in the branding bar

Given the multi-brand nature, the ability is provided to display appropriate brand in OBP. For example, Westpac, St George, Bank SA & Bank of Melbourne. Logos are given in the jspx/jsff files in the current code 'Oracle' logo is maintained in "main.jspx" file. To replace a logo, refer to the following screen shot.

Figure 4–11 Replacing the logo



```
@Override
public void postViewBeforePhaseListener(Main main, PhaseEvent phaseEvent) {

    main.getLogoImage().setSource("/images/common/images/risks.png");
    super.postViewBeforePhaseListener(main, phaseEvent);
}

@Override
public void postViewPoll(Main main, PollEvent pollEvent) {
```

4.5.3 Modifying fonts

Font-family is maintained as a variable and inherited the variable in the mixins which are used to style the various ADF components. Hence if changed the variable's value font will change.

Variable for href to be maintained in backing bean and this variable will be overridden in customization.

Example:

Main.jspx has a placeholder:

href="${pageContext.request.contextPath}${Main.fontPath}"

Main.java holds the path of variable

private String fontPath = "/css/roboto.css";

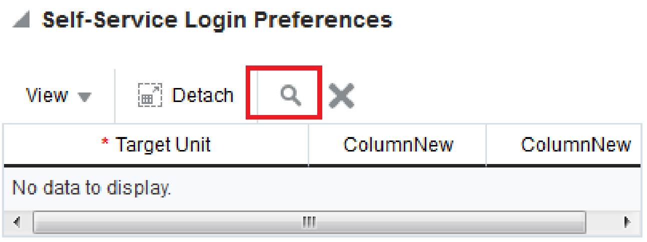
4.5.4 Modifying images

Images are maintained in the jsff as well as in css files.

Many ADF components provide provisions to give icons for different states of the components.

Example: set the icon and hover icon attribute

Figure 4–12 Example: To modify images



this.getButtonAdd().setIcon("/images/common/search/search_16_ena.png");

this.getButtonAdd().setHoverIcon("/images/common/search/search_16_ena.png");

Also wherever component's image can be replaced using css by applying it to the particular selector if component exposes the selector. Same as graphics.

4.5.5 Graphics

Graphics include buttons, warnings, and so on.

Button styles are maintained in the respective css files. (Handled through Section 4.5.1 Replacing skin)

Warning text is maintained in the resource bundle (". properties") files: Replace the properties file in config/resources/taskflows/module

SamplePath : config/resources/taskflows/BankPolicyDefinition_en.properties

4.5.6 Adding a simple field to a product screen

Example: Adding input text to a panel form layout

/* create component and set relevant properties */

RichInputText ui = new RichInputText();

ui.setId("rit1");

ui.setLabel("Input text");

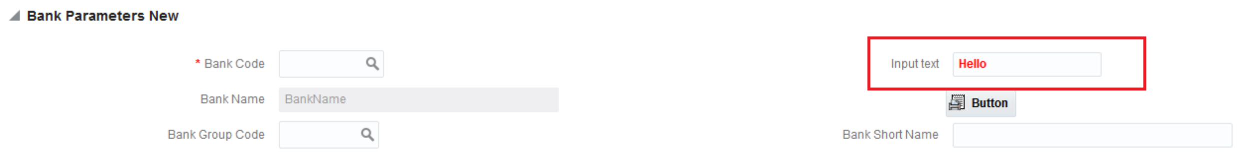
ui.setValue("Hello");

ui.setContentStyle("font-weight:bold;color:red");

/*add the newly created component to existing form */

this.getPfl1().getChildren().add(ui);

Figure 4–13 Example: To add a simple field to a product screen



4.5.7 Adding a complex field popup to a product screen (popup, table, tree, region, tf)

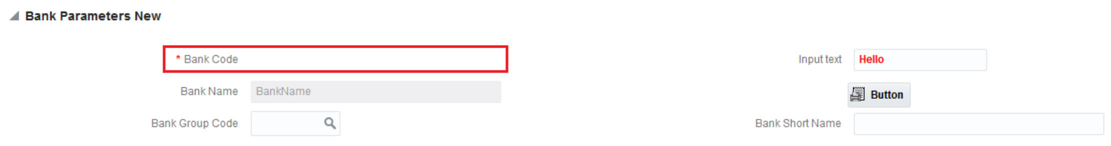
/*Handle via making mds changes*/

4.5.8 Removing an existing field from a product screen

Example: Hiding an LOV from panel form layout

this.getBankCodeLOV().setVisible(false);

Figure 4–14 Example: To remove an existing field from a region



4.5.9 Making certain product optional product fields mandatory or optional

Example: Setting bank name to required

this.getBankName().setRequired(true);

4.5.10 Adding a new column to an existing product grid

Example: Adding a new column to a table in CS26

/* create new column component */

RichColumn ui1 = new RichColumn();

ui1.setHeaderText("ColumnNew");

ui1.setId("col3");

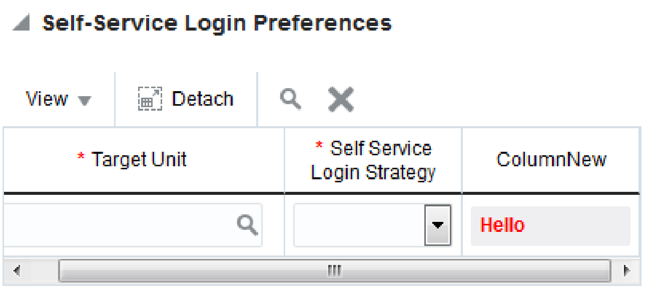
ui1.setAlign("center");

ui1.setRowHeader("unstyled");

/* get the table from bindings where column needs to be added */

```
getT1().getChildren().add(ui1);
AdfFacesContext.getCurrentInstance().addPartialTarget(getT1());

/* set the value in column 3 as required on any event */

RichInputText ui11 = new RichInputText();
ui11.setId("rit1");
ui11.setLabel("Input text");
ui11.setValue("Hello");
ui11.setContentStyle("font-weight:bold;color:red");
ui11.setReadOnly(false);
getT1().getChildren().get(3).getId();
getT1().getRowIndex();
getT1().getChildren().get(3).getChildren().add(ui11);
AdfFacesContext.getCurrentInstance().addPartialTarget(getT1());
```

Figure 4–15 Example: To add a new column to an existing prouct grid



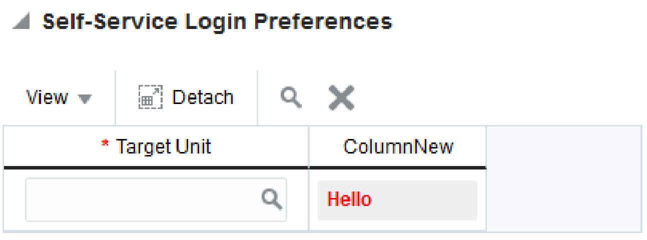
* Target Unit	* Self Service Login Strategy	ColumnNew
		Hello

4.5.11 Hiding columns from an existing product grid

Example: Hiding an existing column from a table in CS26

```
/* get the corresponding column and set its rendered property to false */

this.getT1().getChildren().get(1).setRendered(false);
```

Figure 4–16 Example: To hide columns from an existing product grid



4.5.12 Graying out certain columns from an existing product grid

/* disabling the component that was set inside the column */

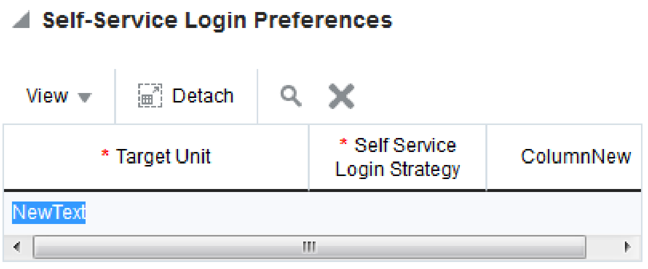
this. ui11.setDisabled(true);

4.5.13 Modifying properties of product table (rows or tablesummary)

this.getT1().setEmptyText("NewText");

in case where properties are picked up via RB the file itself can be replaced in customization

Figure 4–17 Example: To modify the properties of product table



4.5.14 Adding a new section to an existing product screen

/* create a new panel form layout */

RichPanelFormLayout pfl111 = new RichPanelFormLayout();

pfl111.setId("pfl111");

pfl111.setMaxColumns(2);

pfl111.setRows(1);

pfl111.setFieldWidth("60%");

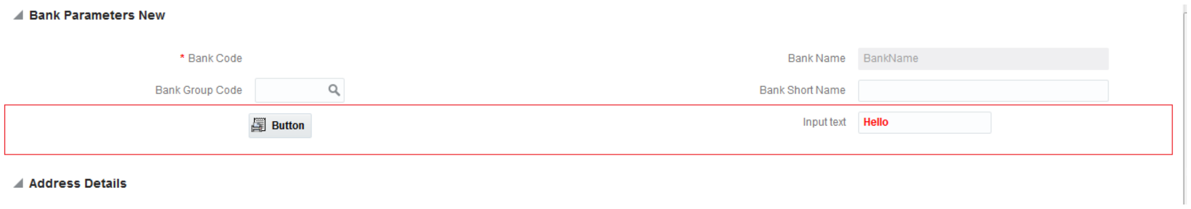
pfl111.setLabelWidth("40%");

getPb1().getChildren().add(pfl111);

AdfFacesContext.getCurrentInstance().addPartialTarget(getPb1());

```
/* create components to be added to that section */

RichInputText ui = new RichInputText();
ui.setId("rit1");
ui.setLabel("Input text");
ui.setValue("Hello");
ui.setContentStyle("font-weight:bold;color:red");
getPfl1().getChildren().add(ui);
AdfFacesContext.getCurrentInstance().addPartialTarget(getPfl1());
RichCommandButton ui2 = new RichCommandButton();
ui2.setId("ch1");
ui2.setText("Button");
ui2.setInlineStyle("font-weight:bold;");
ui2.setIcon("/images/common/search/search_16_ena.png");
ui2.setIcon("/images/common/print/printreciept_16_ena.png");

/*add new components to the new section */

getPb1().getChildren().get(2).getChildren().add(ui2);
getPb1().getChildren().get(2).getChildren().add(ui);
```

Figure 4–18 Example: To add a new section to an existing product screen



4.5.15 Hiding a section from a product screen

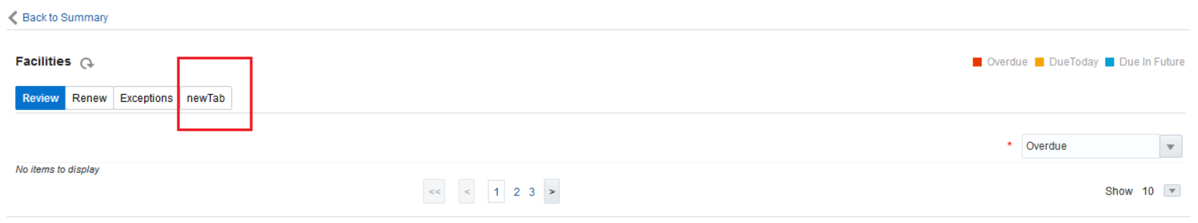
```
/* Hiding all components inside the panel form layout */

this.getPfl1().setVisible(false);
```

4.5.16 Adding a new tab to an existing product screen made of tabs

```
/* create a new tab and add its relevant properties */
```

```
RichCommandNavigationItem ui2 = new RichCommandNavigationItem();
ui2.setId("newTab");
ui2.setSelected(false);
ui2.setText("newTab");

/* add it to the navigation pane */

this.getNp1().getChildren().add(ui2);
```

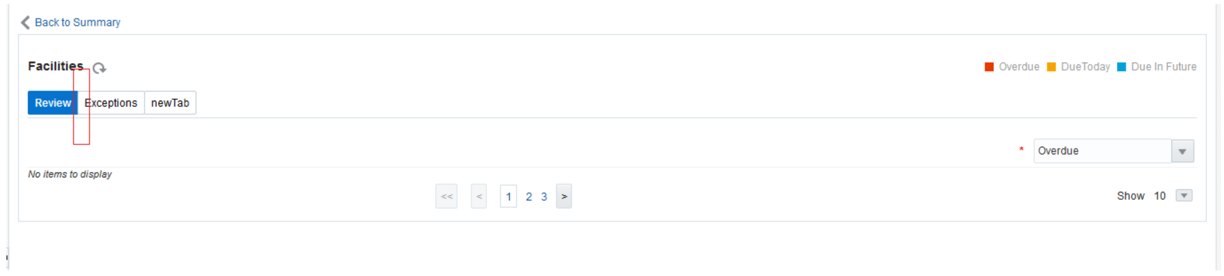
Figure 4–19 Example: To add a new tab to existing product screen made of tabs



4.5.17 Hiding a tab from a product screen made of multiple tabs

```
this.getNp1().getChildren().get(1).setRendered(false);
```

Figure 4–20 Example: To hide a tab from a product screen made of multiple tabs



4.5.18 Adding new buttons or links

This approach will not work for "Approvals" and "UI level security"

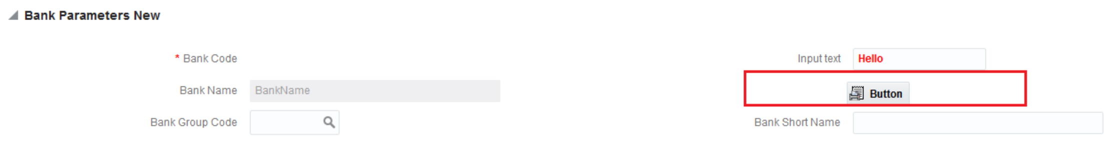
```
/* Create a new command button and set its relevant properties*/
RichCommandButton ui2 = new RichCommandButton();
ui2.setId("ch1");
ui2.setText("Button");
ui2.setInlineStyle("font-weight:bold;");
ui2.setIcon("/images/common/search/search_16_ena.png");
```

/* add it to the relevant panel component */

this.getPfl1().getChildren().add(ui2);

AdfFacesContext.getCurrentInstance().addPartialTarget(getPfl1());

Figure 4–21 Example: To add new buttons or links



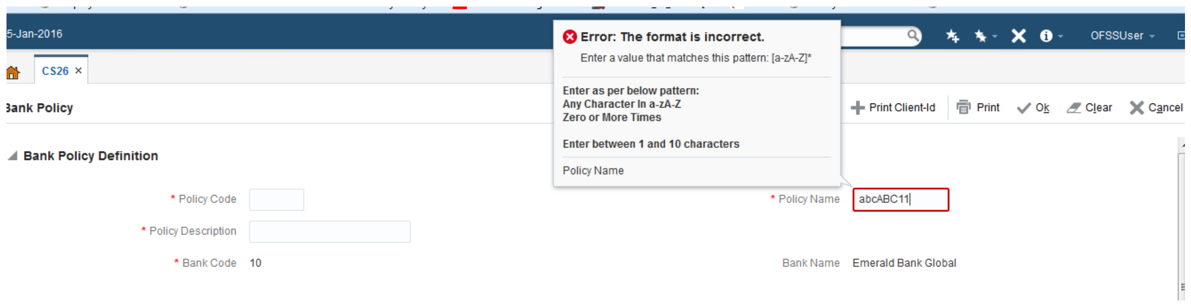
4.5.19 Overriding / Customizing the product behaviour on certain actions like button clicks or tab-outs

/* need to create a new link programmatically and link the action listener method to it */

4.5.20 Overriding the product validation pattern

this.getPolicyName().setPattern("[a-zA-Z]*");

Figure 4–22 Example: To override the product validation pattern


Error: The format is incorrect.
Enter a value that matches this pattern: [a-zA-Z]*

Enter as per below pattern:
Any Character In a-zA-Z
Zero or More Times

Enter between 1 and 10 characters

Policy Name

4.5.21 Overriding the product lengths (min/max)

this.getPolicyName().setMaxLength("10");

4.5.22 Disable / Enable certain product fields

this.getBankName().setDisabled(true);

4.5.23 Change certain product fields to read-only either on load or based on certain conditions

this.getBankName().setReadOnly(true);

4.5.24 Change label of existing product fields

this.getBankName().setValue("BankName");

4.5.25 DC validation

The text for error message comes from "CommonValidationMessages_en.properties" and this file can be replaced in customization. However the values for Min and Max length inside the message can be overridden.

4.5.26 LOV Extension– LOV Delegate Pattern

- **Consulting use case:**
  - Display the list of accounts of the logged in user from a third party system.
- **Implementation:**
  - Re-use "LOVDelegate" framework
  - Override the existing implementation in HostQueries.xml with a <service> tag while the existing product implementation is present conditionally.
  - <Service> tag in-turns points to a new LOVDelegate class which implements the ILOVDelegate interface.
  - The entire custom implementation to fetch external records will be present in the LOVDelegate class.
  - Conditionally invoke the consulting implementation or product implementation based on the requirements.
- **Key Benefits:**
  - Easy to plug-in with minimal changes. Host layer only impacted with no impact to the presentation layer.
  - Query can be overridden in a very sophisticated way with the use of <Service> tag.
  - Plug-in-play and can be easily turn off if required.
- **Visual representation is given below:**

Figure 4–23 LOV Extension– LOV Delegate Pattern

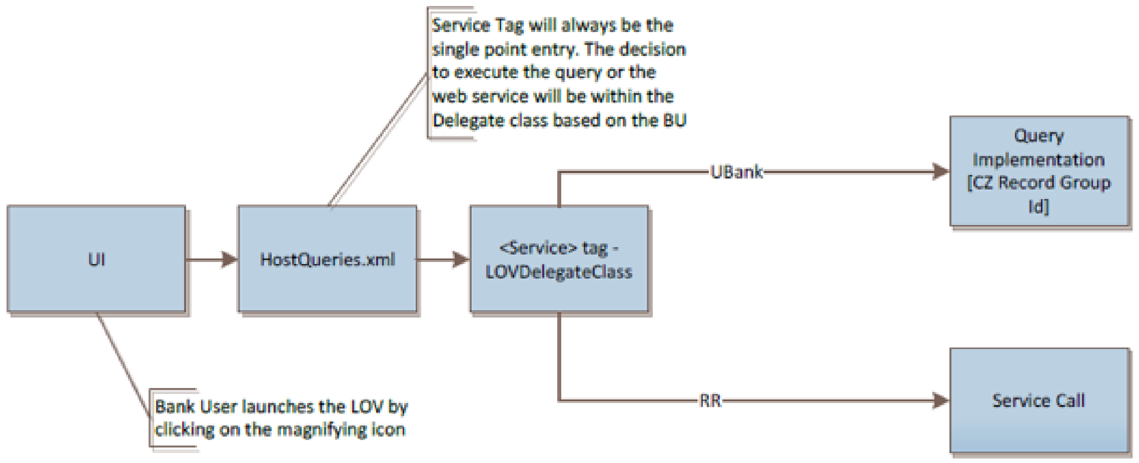
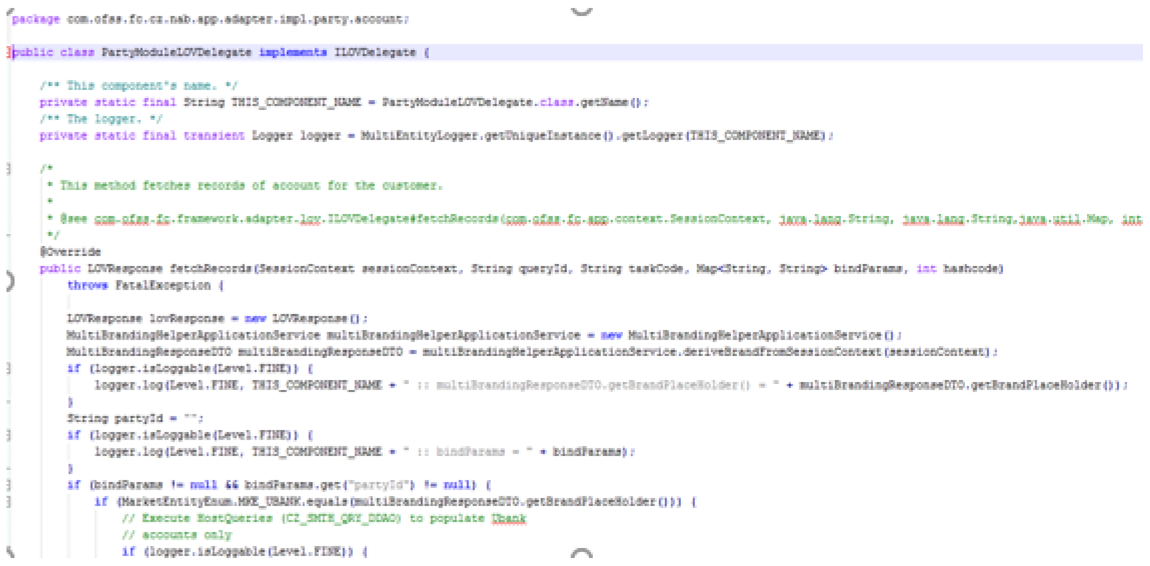

Service Tag will always be the single point entry. The decision to execute the query or the web service will be within the Delegate class based on the BU
UI
HostQueries.xml
<Service> tag - LOVDelegateClass
UBank
Query Implementation [CZ Record Group Id]
RR
Service Call
Bank User launches the LOV by clicking on the magnifying icon

Figure 4–24 Sample Code Snippet



```
package com.ofss.fc.cz.nab.app.adapter.impl.party.account;

public class PartyModuleLOVDelegate implements ILOVDelegate {

    /** This component's name. */
    private static final String THIS_COMPONENT_NAME = PartyModuleLOVDelegate.class.getName();
    /** The logger. */
    private static final transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(THIS_COMPONENT_NAME);

    /*
     * This method fetches records of account for the customer.
     *
     * @see com.ofss.fc.framework.adapter.lov.ILOVDelegate#fetchRecords(com.ofss.fc.app.context.SessionContext, java.lang.String, java.lang.String,java.util.Map, int
     */
    @Override
    public LOVResponse fetchRecords(SessionContext sessionContext, String queryId, String taskCode, Map<String, String> bindParams, int hashcode)
        throws FatalException {

        LOVResponse lovResponse = new LOVResponse();
        MultiBrandingHelperApplicationService multiBrandingHelperApplicationService = new MultiBrandingHelperApplicationService();
        MultiBrandingResponseDTO multiBrandingResponseDTO = multiBrandingHelperApplicationService.deriveBrandFromSessionContext(sessionContext);
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, THIS_COMPONENT_NAME + " :: multiBrandingResponseDTO.getBrandPlaceHolder() = " + multiBrandingResponseDTO.getBrandPlaceHolder());
        }
        String partyId = "";
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, THIS_COMPONENT_NAME + " :: bindParams = " + bindParams);
        }
        if (bindParams != null && bindParams.get("partyId") != null) {
            if (MarketEntityEnum.MKE_UBANK.equals(multiBrandingResponseDTO.getBrandPlaceHolder())) {
                // Execute HostQueries (CZ_SMTH_QRY_DDAO) to populate Ubank
                // accounts only
                if (logger.isLoggable(Level.FINE)) {
```

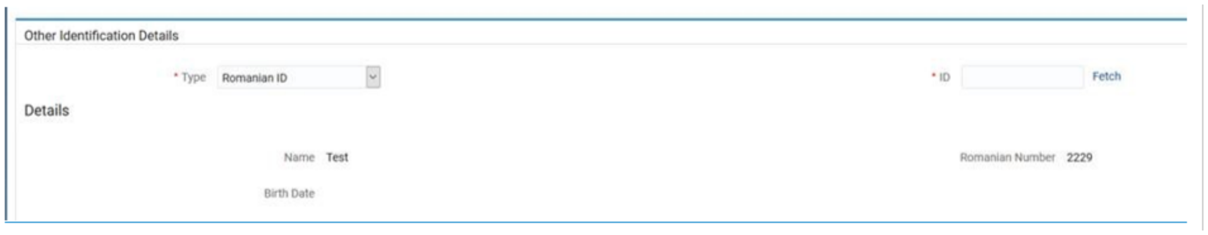
4.6 Using the JSFF Utils

4.6.1 How to Use JSFF Utils

Following is the example to use JSFFUtils:

```
JSFUtils.insertPanelHeader
("rphRomanianDetails","Details",parentId,uiComponent);
JSFUtils.insertRichPanelFormLayoutEnd
("rpfRomanian1","60%","40%",2,1,"rphRomanianDetails",uiComponent);
```

Figure 4–25 Example of JSFF Utils



4.6.2 Sample JSFF Utils Code Snippet

```
/**
*This method adds af:PanelFormLayout ADF component at the end of
the given parent.
* @param id sets id attribute on the af:PanelFormLayout.Type
String.
* @param fieldWidth sets fieldWidth attribute on the
af:PanelFormLayout.Type String.
* @param labelWidth sets labelWidth attribute on the
af:PanelFormLayout.Type String.
```

```
* @param maxColumns sets maxColumns attribute on the
af:PanelFormLayout.Type integer.
* @param row sets row attribute on the af:PanelFormLayout.Type
integer.
* @param parentId is the id of the immediate parent component where
af:PanelFormLayout need to be appended.Type String
* @param superParent is the component where parentId is placed.Type
UIComponent
*/
public static void insertRichPanelFormLayoutEnd(String id,String
fieldWidth, String labelWidth, int maxColumns,int row, String
parentId, UIComponent superParent) {
UIComponent uiComponentPGL = superParent.findComponent(parentId);
RichPanelFormLayout richPanelFormLayout = new RichPanelFormLayout
();
richPanelFormLayout.setId(id);
richPanelFormLayout.setFieldWidth(fieldWidth);
richPanelFormLayout.setLabelWidth(labelWidth);
richPanelFormLayout.setMaxColumns(maxColumns);
richPanelFormLayout.setRows(row);
uiComponentPGL.getChildren().add(richPanelFormLayout);
}
*This method adds af:panelHeader ADF component at the end of the
given parent.
* @param id sets id attribute on the af:panelHeader.Type String.
* @param text sets text attribute on the af:panelHeader.Type
String.
* @param parentId is the id of the immediate parent component where
af:panelHeader need to be appended.Type String
* @param superParent is the component where parentId is placed.Type
UIComponent
*/
public static void insertPanelHeader(String id,String text,String
parentId,UIComponent superParent){
UIComponent uiComponentParent = superParent.findComponent
(parentId);
RichPanelHeader richPanelHeader = new RichPanelHeader();
richPanelHeader.setId(id);
richPanelHeader.setText(text);
uiComponentParent.getChildren().add(richPanelHeader);
}
```

5 ADF Screen Customizations Using MDS

OBP provides the extensibility to an application for customizing certain additional requirements of a client. However, since these additional requirements differ from client to client, and the base application functionality remains the same, the code to handle the additional requirements should be kept separate from the code of the base application. For this purpose, **Seeded Customizations** (built on the Oracle Metadata Services framework) can be used to customize an application.

Note

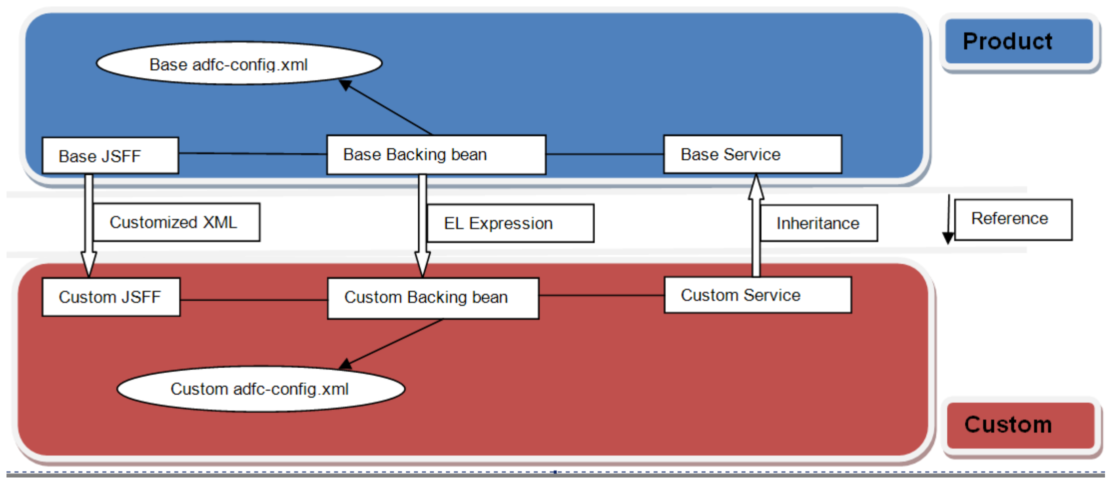
It is recommended to use ADF screen extensions for UI changes instead of mds where ever possible as it is easier to upgrade to new version of product.

5.1 Seeded Customization Concepts

When designing seeded customizations for an application, one or more customization layers need to be specified. A customization layer is used to hold a set of customizations. A customization layer supports one or more customization layer value which specifies which set of customizations to apply at runtime.

Custom Application View can be represented as follows:

Figure 5–1 Customization Application View



Oracle JDeveloper 11g includes a special role for designing customizations for each customization layer and layer value called the Customization Developer Role.

The following section explains the details about the Oracle JDeveloper customization mode as well as customizing and extending of the ADF application artifact. The detailed documentation for customizing and extending ADF Application Artifacts is also available at the Oracle website:

http://docs.oracle.com/cd/E25178_01/fusionapps.1111/e16691/ext_busobjedit.htm

5.2 Customization Layer

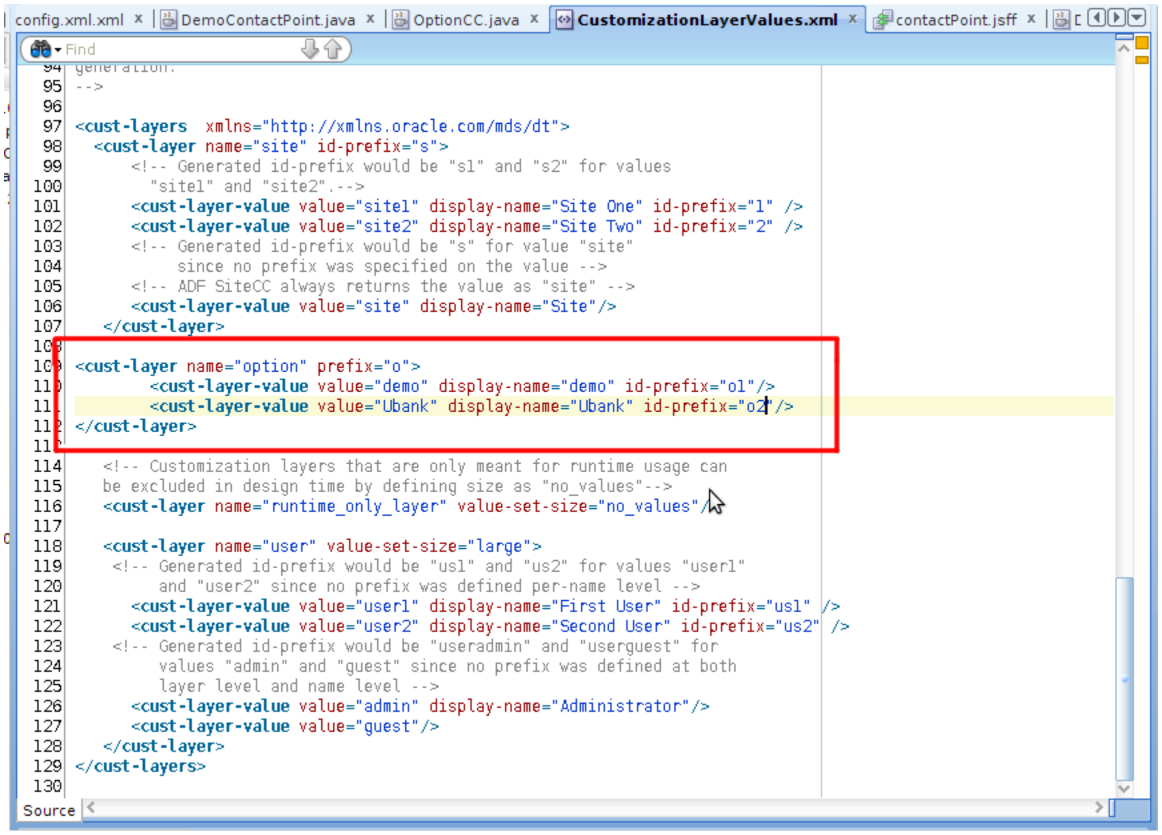
To customize an application, you must specify the customization layers and their values in the CustomizationLayerValues.xml file, so that they are recognized by JDeveloper.

For example, you can create a customization layer with the name **option** and values **demo** and another bank name.

To create the customization layer, follow these steps:

1. From the main menu, choose the **File -> Open** option. Locate and open the file CustomizationLayerValues.xml which is found in the <JDEVELOPER_HOME>/jdeveloper/jdev directory. In the XML editor, add the entry for a new customization layer and values as shown in the following image.

Figure 5–2 CustomizationLayerValues.xml



```
generation.
-->

<cust-layers  xmlns="http://xmlns.oracle.com/mds/dt">
  <cust-layer name="site" id-prefix="s">
      <!-- Generated id-prefix would be "s1" and "s2" for values
        "site1" and "site2".-->
      <cust-layer-value value="site1" display-name="Site One" id-prefix="1" />
      <cust-layer-value value="site2" display-name="Site Two" id-prefix="2" />
      <!-- Generated id-prefix would be "s" for value "site"
           since no prefix was specified on the value -->
      <!-- ADF SiteCC always returns the value as "site" -->
      <cust-layer-value value="site" display-name="Site"/>
   </cust-layer>

<cust-layer name="option" prefix="o">
        <cust-layer-value value="demo" display-name="demo" id-prefix="o1"/>
        <cust-layer-value value="Ubank" display-name="Ubank" id-prefix="o2"/>
</cust-layer>

   <!-- Customization layers that are only meant for runtime usage can
   be excluded in design time by defining size as "no_values"-->
   <cust-layer name="runtime_only_layer" value-set-size="no_values"/>

   <cust-layer name="user" value-set-size="large">
    <!-- Generated id-prefix would be "us1" and "us2" for values "user1"
         and "user2" since no prefix was defined per-name level -->
      <cust-layer-value value="user1" display-name="First User" id-prefix="us1" />
      <cust-layer-value value="user2" display-name="Second User" id-prefix="us2" />
    <!-- Generated id-prefix would be "useradmin" and "userguest" for
         values "admin" and "guest" since no prefix was defined at both
         layer level and name level -->
      <cust-layer-value value="admin" display-name="Administrator"/>
      <cust-layer-value value="guest"/>
   </cust-layer>
</cust-layers>
```

2. Save and close the file.

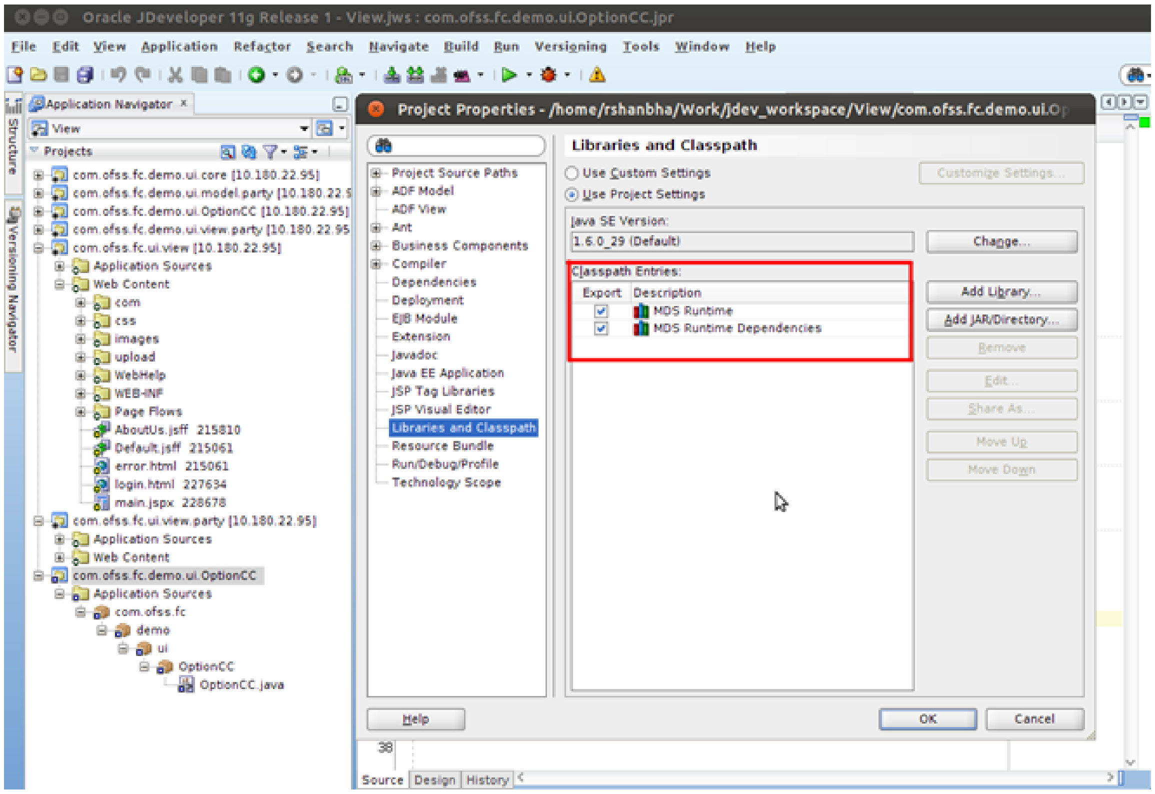
5.3 Customization Class

Before customizing an application, a customization class needs to be created. This class represents the interface that the *Oracle Metadata Services* framework uses to identify the customization layer that should be applied to the application's base metadata.

To create a customization class, follow these steps:

1. From the main menu, choose **File -> New**.
2. Create a generic project and give a name *(com.ofss.fc.demo.ui.OptionCC)* to the project.
3. Go to **Project Properties** for this project and add the required **MDS** libraries in the classpath of the project.

Figure 5–3 Customization Class



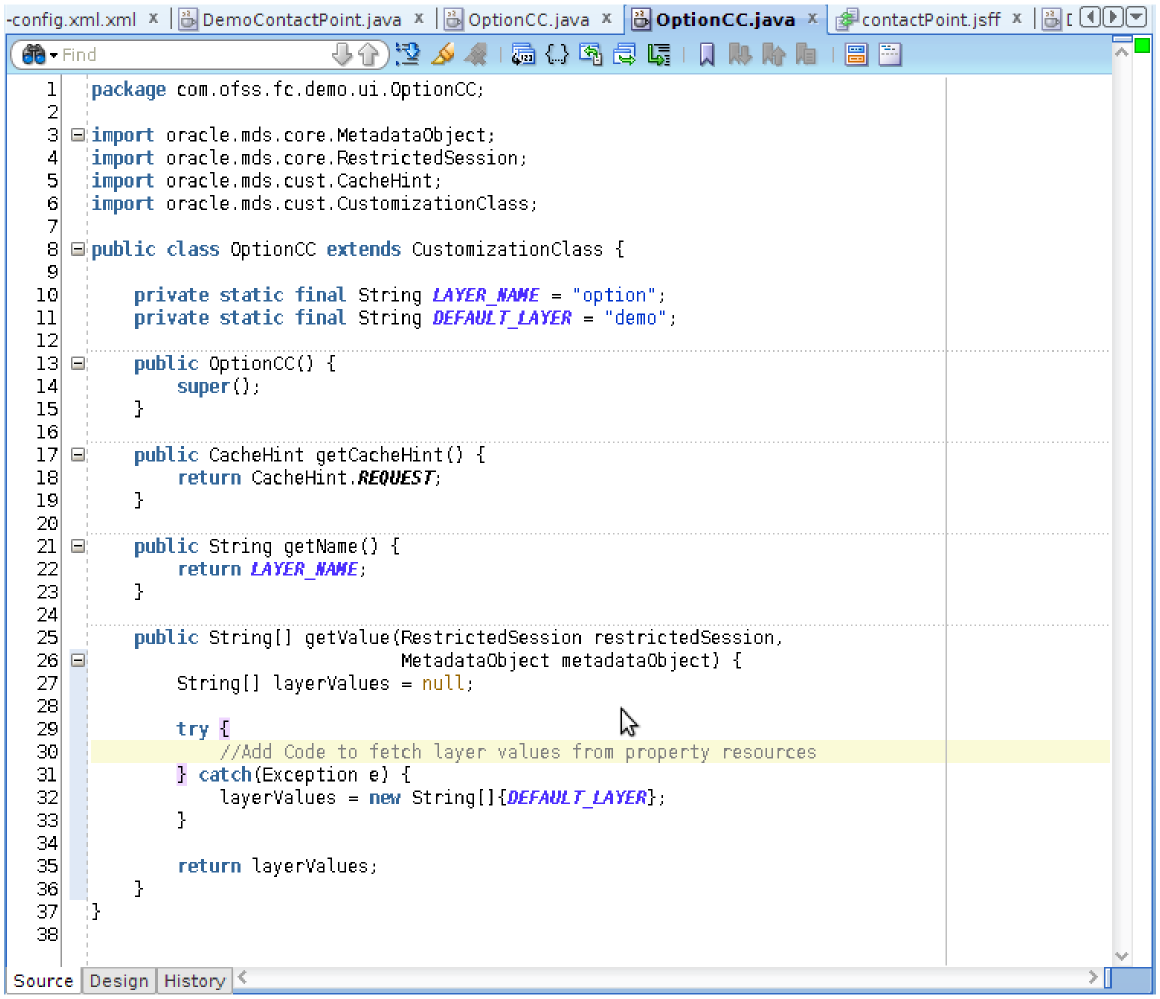
4. Create the customization class in this project. The customization class **must** extend the *oracle.mds.cust.CustomizationClass* abstract class.

   Following are the abstract methods of the CustomizationClass:

   - getCacheHint() - This method will return the information about whether the customization layer is applicable to all users, a set of users, a specific HTTP request or a single user.
   - getName() - This method will return the name of the customization layer.
   - getValue() - This method will return the customization layer value at runtime.

   The screenshot below depicts an implementation for the methods:

Figure 5–4 Implementation for the abstract methods of CustomizationClass



```
package com.ofss.fc.demo.ui.OptionCC;

import oracle.mds.core.MetadataObject;
import oracle.mds.core.RestrictedSession;
import oracle.mds.cust.CacheHint;
import oracle.mds.cust.CustomizationClass;

public class OptionCC extends CustomizationClass {

    private static final String LAYER_NAME = "option";
    private static final String DEFAULT_LAYER = "demo";

    public OptionCC() {
        super();
    }

    public CacheHint getCacheHint() {
        return CacheHint.REQUEST;
    }

    public String getName() {
        return LAYER_NAME;
    }

    public String[] getValue(RestrictedSession restrictedSession,
                             MetadataObject metadataObject) {
        String[] layerValues = null;

        try {
            //Add Code to fetch layer values from property resources
        } catch(Exception e) {
            layerValues = new String[]{DEFAULT_LAYER};
        }

        return layerValues;
    }
}
```

5. Build this class and deploy the project as a JAR file (com.ofss.fc.demo.ui.OptionCC.jar). This JAR file should only contain the customization class.
6. Place this JAR file in the location <JDEVELOPER_HOME>/jdeveloper/jdev/lib/patches so that the customization class is available in the classpath of JDeveloper.

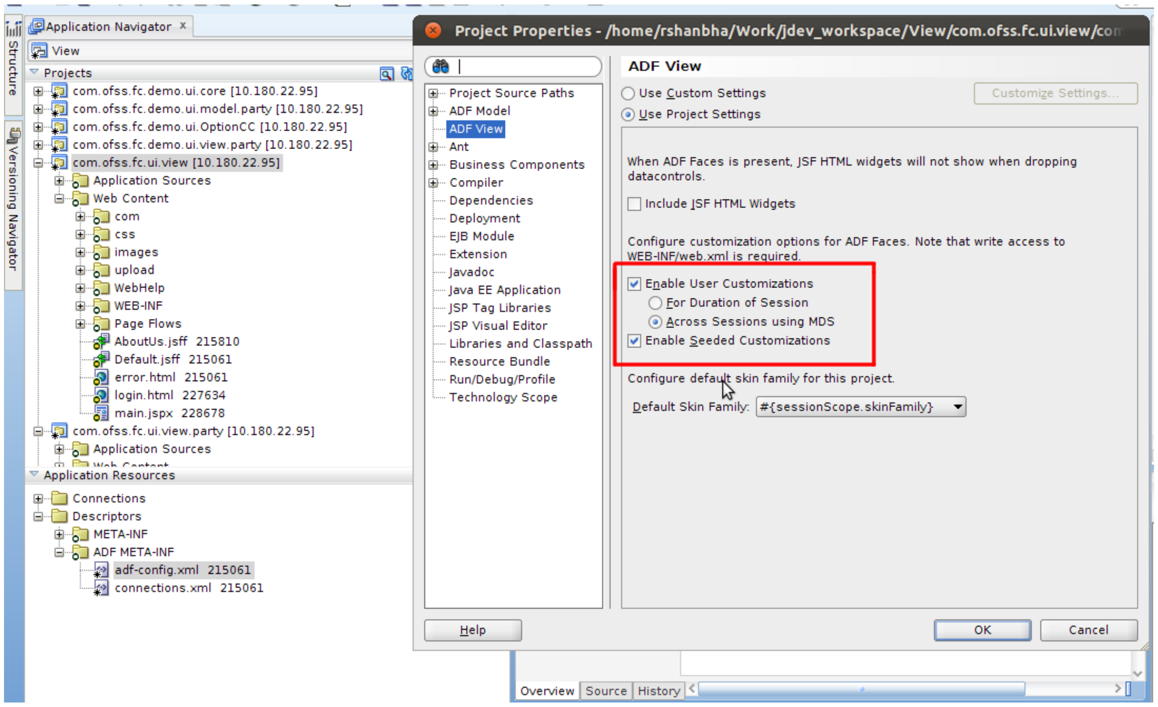
5.4 Enabling Application for Seeded Customization

Seeded customization of an application is the process of taking a generalized application and making modifications to suit the needs of a particular group. The generalized application first needs to be enabled for seeded customization before any customizations can be done on the application.

To enable seeded customization for the application, follow these steps:

1. Go to the **Project Properties** of the application's project.
2. In the **ADF Views** section, check the **Enable Seeded Customizations** option.

Figure 5–5 Enable Seeded Customizations



Project Properties - /home/rshanbha/Work/jdev_workspace/View/com.ofss.fc.ui.view/com

ADF View

Use Custom Settings

[✓ Use Project Settings]

When ADF Faces is present, JSF HTML widgets will not show when dropping datacontrols.

☐ Include JSF HTML Widgets

Configure customization options for ADF Faces. Note that write access to WEB-INF/web.xml is required.

☑ Enable User Customizations

For Duration of Session [✓ Across Sessions using MDS]

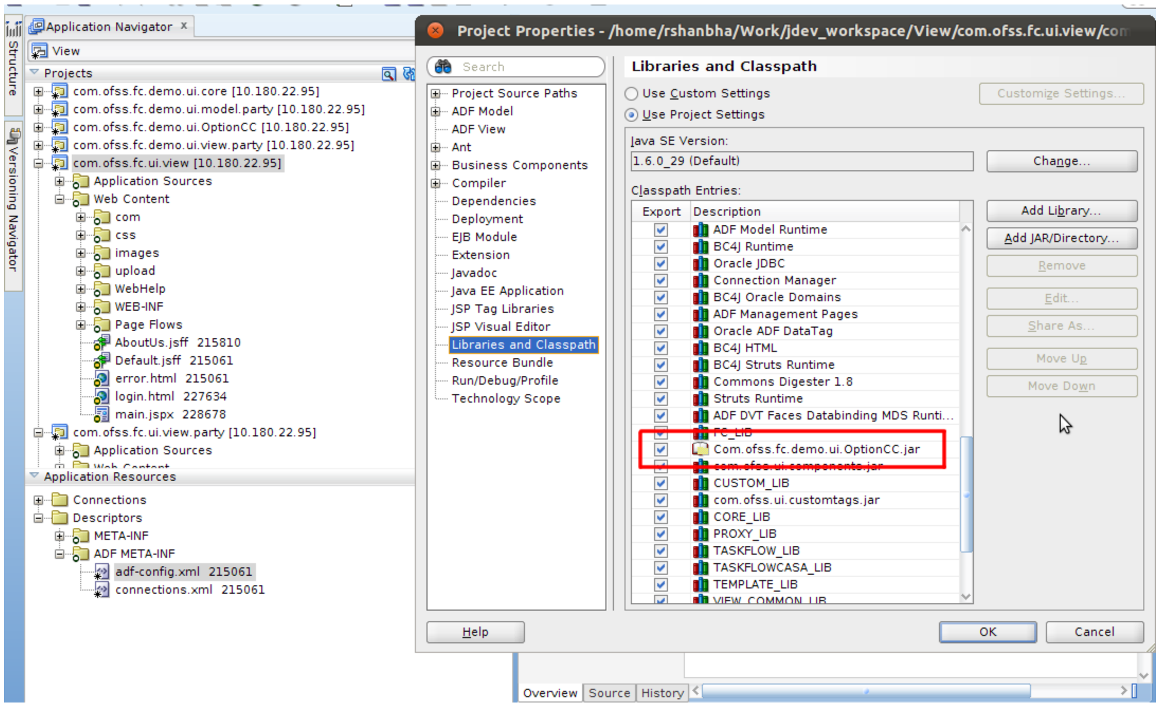
☑ Enable Seeded Customizations

Configure default skin family for this project.

Default Skin Family: #{sessionScope.skinFamily}

3. In the Libraries and Classpath section, add the previously deployed com.ofss.fc.demo.ui.OptionCC.jar which contains the customization class.

Figure 5–6 Adding com.ofss.fc.demo.ui.OptionCC.jar



Project Properties - /home/rshanbha/Work/jdev_workspace/View/com.ofss.fc.ui.view/co

Libraries and Classpath

Use Custom Settings
[✓ Use Project Settings]

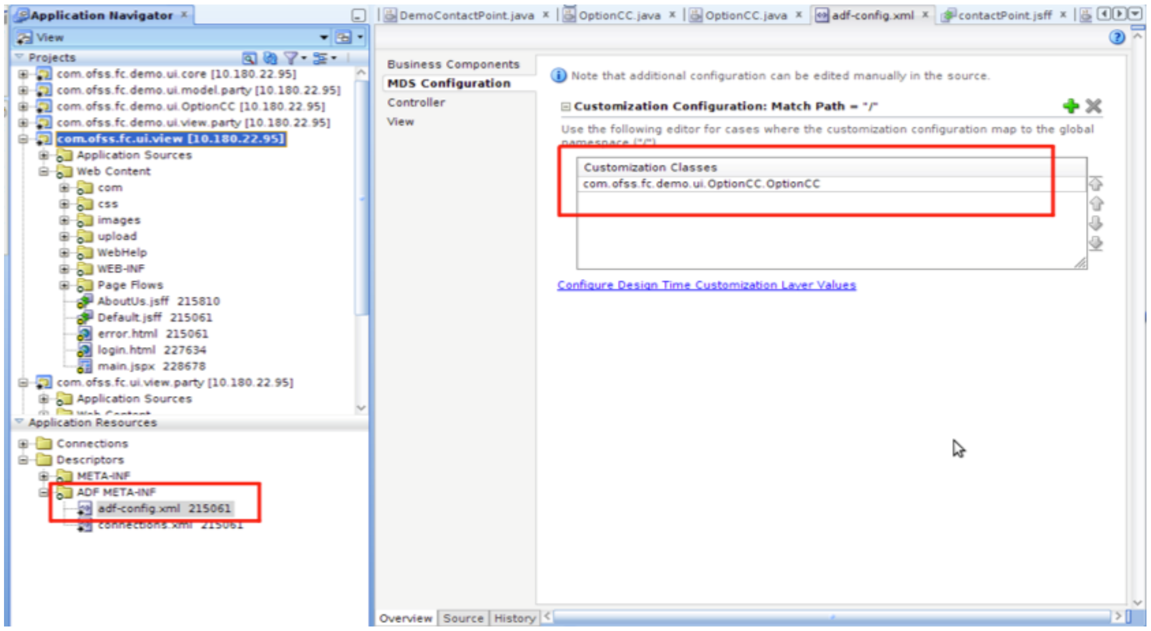
Java SE Version: 1.6.0_29 (Default)

Classpath Entries:

Export	Description
✓	ADF Model Runtime
✓	BC4J Runtime
✓	Oracle JDBC
✓	Connection Manager
✓	BC4J Oracle Domains
✓	ADF Management Pages
✓	Oracle ADF DataTag
✓	BC4J HTML
✓	BC4J Struts Runtime
✓	Commons Digester 1.8
✓	Struts Runtime
✓	ADF DVT Faces Databinding MDS Runti...
✓	FC_LIB
✓	Com.ofss.fc.demo.ui.OptionCC.jar
✓	com.ofss.ui.components.jar
✓	CUSTOM_LIB
✓	com.ofss.ui.customtags.jar
✓	CORE_LIB
✓	PROXY_LIB
✓	TASKFLOW_LIB
✓	TASKFLOWCASA_LIB
✓	TEMPLATE_LIB
✓	VIEW_COMMON_LIB

4. In the Application Resources tab, open the adf-config.xml present in the Descriptors/ADF META-INF folder. In the list of Customization Classes, remove all the entries and add the com.ofss.fc.demo.ui.OptionCC.OptionCC class to this list.

Figure 5–7 Adding com.ofss.fc.demo.ui.OptionCC.OptionCC



MDS Configuration

Note that additional configuration can be edited manually in the source.

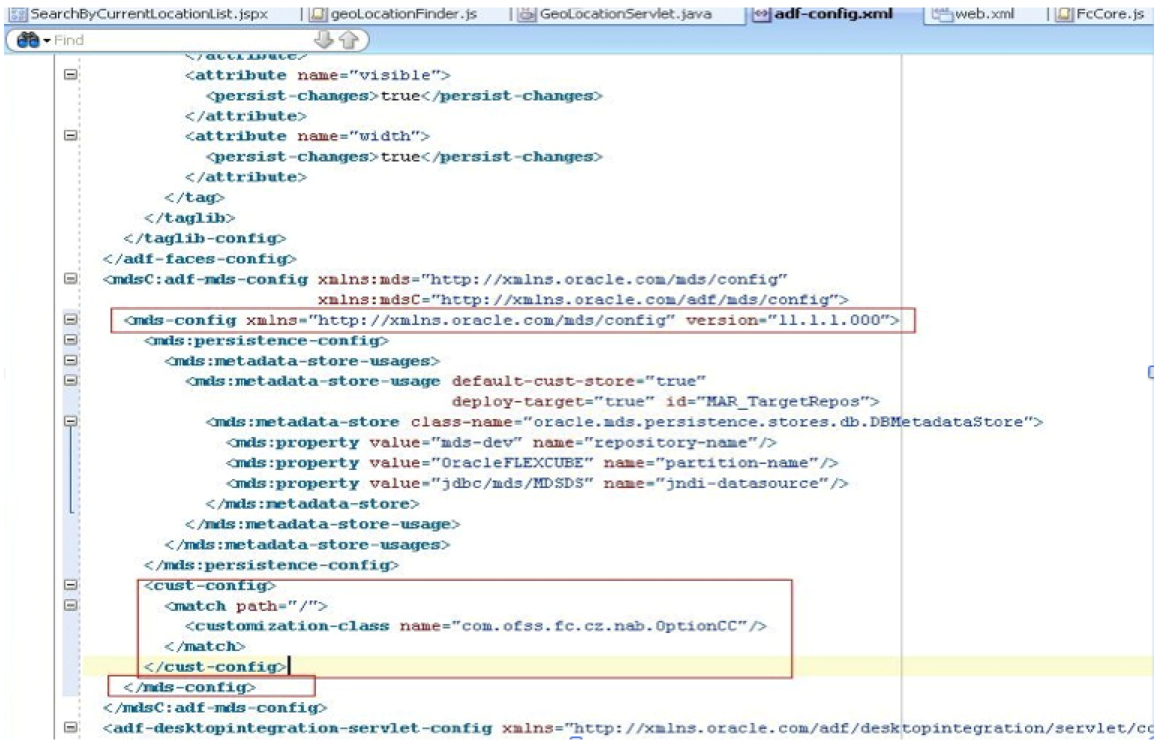
Customization Configuration: Match Path = "/"

Use the following editor for cases where the customization configuration map to the global namespace ("/").

Customization Classes
com.ofss.fc.demo.ui.OptionCC.OptionCC

Configure Design Time Customization Layer Values

Figure 5–8 Adf-config.xml



```
            </attribute>
            <attribute name="visible">
              <persist-changes>true</persist-changes>
            </attribute>
            <attribute name="width">
              <persist-changes>true</persist-changes>
            </attribute>
          </tag>
        </taglib>
      </taglib-config>
    </adf-faces-config>
    <mdsC:adf-mds-config xmlns:mds="http://xmlns.oracle.com/mds/config"
                         xmlns:mdsC="http://xmlns.oracle.com/adf/mds/config">
      <mds-config xmlns="http://xmlns.oracle.com/mds/config" version="11.1.1.000">
        <mds:persistence-config>
          <mds:metadata-store-usages>
            <mds:metadata-store-usage default-cust-store="true"
                                      deploy-target="true" id="MAR_TargetRepos">
              <mds:metadata-store class-name="oracle.mds.persistence.stores.db.DBMetadataStore">
                <mds:property value="mds-dev" name="repository-name"/>
                <mds:property value="OracleFLEXCUBE" name="partition-name"/>
                <mds:property value="jdbc/mds/MDSDS" name="jndi-datasource"/>
              </mds:metadata-store>
            </mds:metadata-store-usage>
          </mds:metadata-store-usages>
        </mds:persistence-config>
        <cust-config>
          <match path="/">
            <customization-class name="com.ofss.fc.cz.nab.OptionCC"/>
          </match>
        </cust-config>
      </mds-config>
    </mdsC:adf-mds-config>
    <adf-desktopintegration-servlet-config xmlns="http://xmlns.oracle.com/adf/desktopintegration/servlet/co
```

5.5 Customization Project

After creating the Customization Layer and the Customization Class and enabling the application for Seeded Customizations, the next step is to create a project which will hold the customizations for the application.

To create the customization project, follow these steps:

1. From the main menu, choose **File -> New**. Create a new Web Project with the following technologies:
   - ADF Business Components
   - Java
   - JSF
   - JSP and Servlets
2. Go to the **Project Properties** of the project and in the classpath of the project, add the following jars:
   - Customization class JAR (com.ofss.fc.demo.ui.OptionCC.jar)
   - The project JAR which contains the screen / component to be customized. For example, if you want to customize the *Party -> Contact Information -> Contact Point* screen, the related project JAR is com.ofss.fc.ui.view.party.jar.
   - All the dependent JARS / libraries for the project JAR.
   - Enable this project for **Seeded Customizations.**

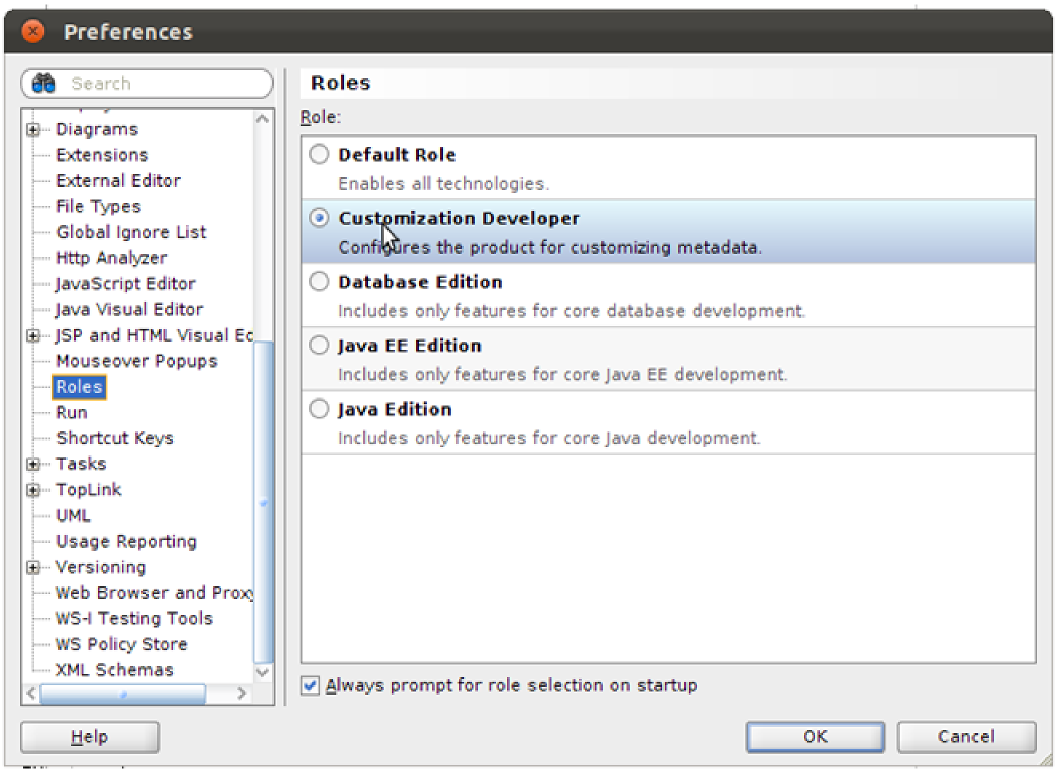
5.6 Customization Role and Context

Oracle JDeveloper 11g includes a specific role called Customization Developer Role that is used for editing seeded customizations.

To edit customizations to an application, you will need to switch JDeveloper to that role, follow these steps:

1. In **Tools > Preferences > Roles**, select the Customization Developer Role.

Figure 5–9 Customization Developer



Roles

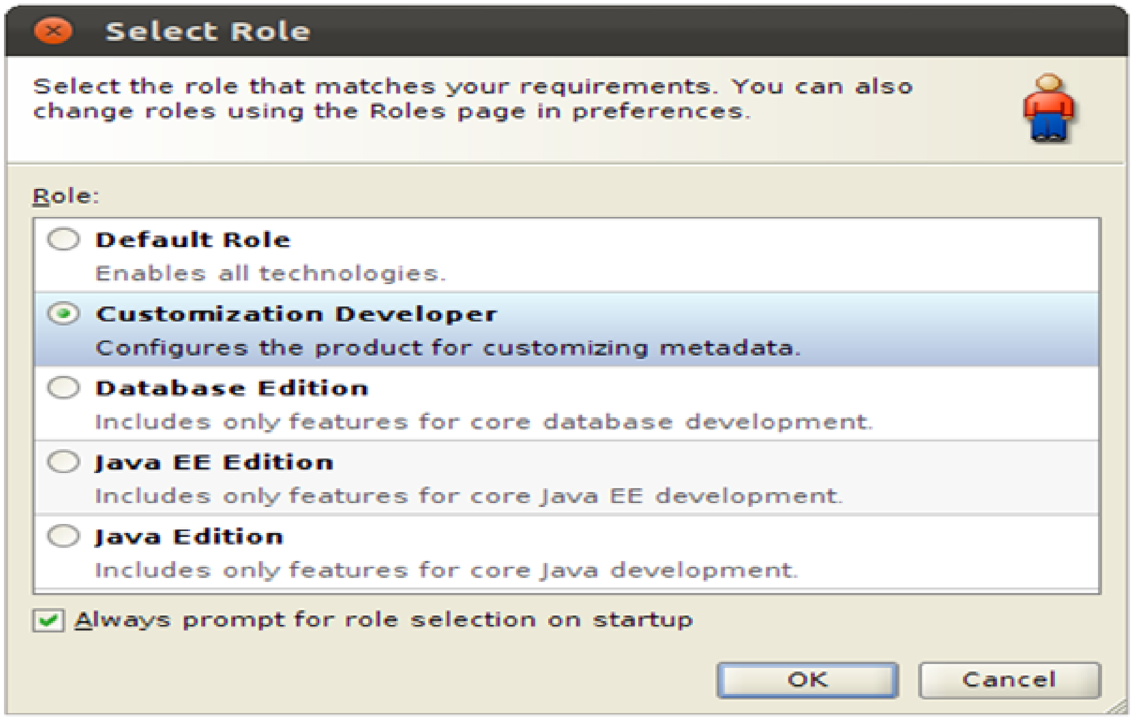
Role:

- Default Role — Enables all technologies.
- [✓ Customization Developer] — Configures the product for customizing metadata.
- Database Edition — Includes only features for core database development.
- Java EE Edition — Includes only features for core Java EE development.
- Java Edition — Includes only features for core Java development.

[✓] Always prompt for role selection on startup

2. Select the "Always prompt for role selection on start up" option.

Figure 5–10 Selecting Always Prompt for Role Selection on Start Up


Select Role

Select the role that matches your requirements. You can also change roles using the Roles page in preferences.

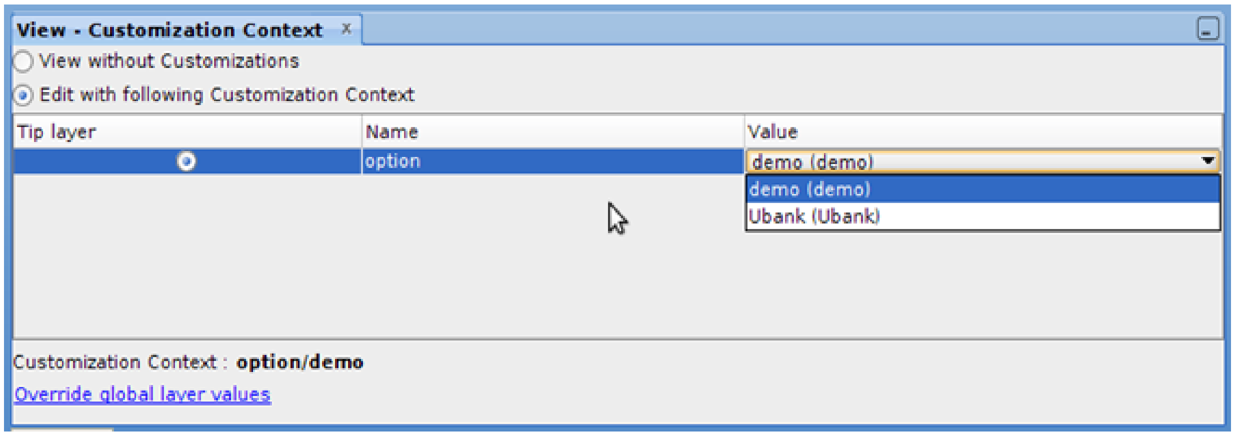
Role:

- ( ) **Default Role** Enables all technologies.
- (●) **Customization Developer** Configures the product for customizing metadata.
- ( ) **Database Edition** Includes only features for core database development.
- ( ) **Java EE Edition** Includes only features for core Java EE development.
- ( ) **Java Edition** Includes only features for core Java development.

[✓] Always prompt for role selection on startup

3. On restarting JDeveloper, you will be prompted for role selection. Select *Customization Developer Role.*
4. Once Oracle JDeveloper 11g has restarted, ensure that the application to be customized is selected in the Application Navigator and have a look around the integrated development environment. You will notice a few changes from the Default Role. The first change you might notice is that files (such as Java classes), that are not customizable, are now read only. The Customization Developer Role can only be used for editing seeded customizations. Anything that is not related to seeded customizations will be disabled. The second major difference you might notice is the *MDS - Customization Context* window that is displayed.
5. Check the *Edit with following Customization Context* option. You will see a list of customization layer name and customization layer values which were defined in the *CustomizationLayerValues.xml* file.
6. Select the Customization Context for which, the customizations you edit should be applicable.

Figure 5–11 View Customization Context


View - Customization Context

View without Customizations

[✓ Edit with following Customization Context]

Tip layer	Name	Value
●	option	demo (demo)

demo (demo)
Ubank (Ubank)

Customization Context : **option/demo**

Override global layer values

All the customizations which are done to the application are now stored for the selected Customization Context.

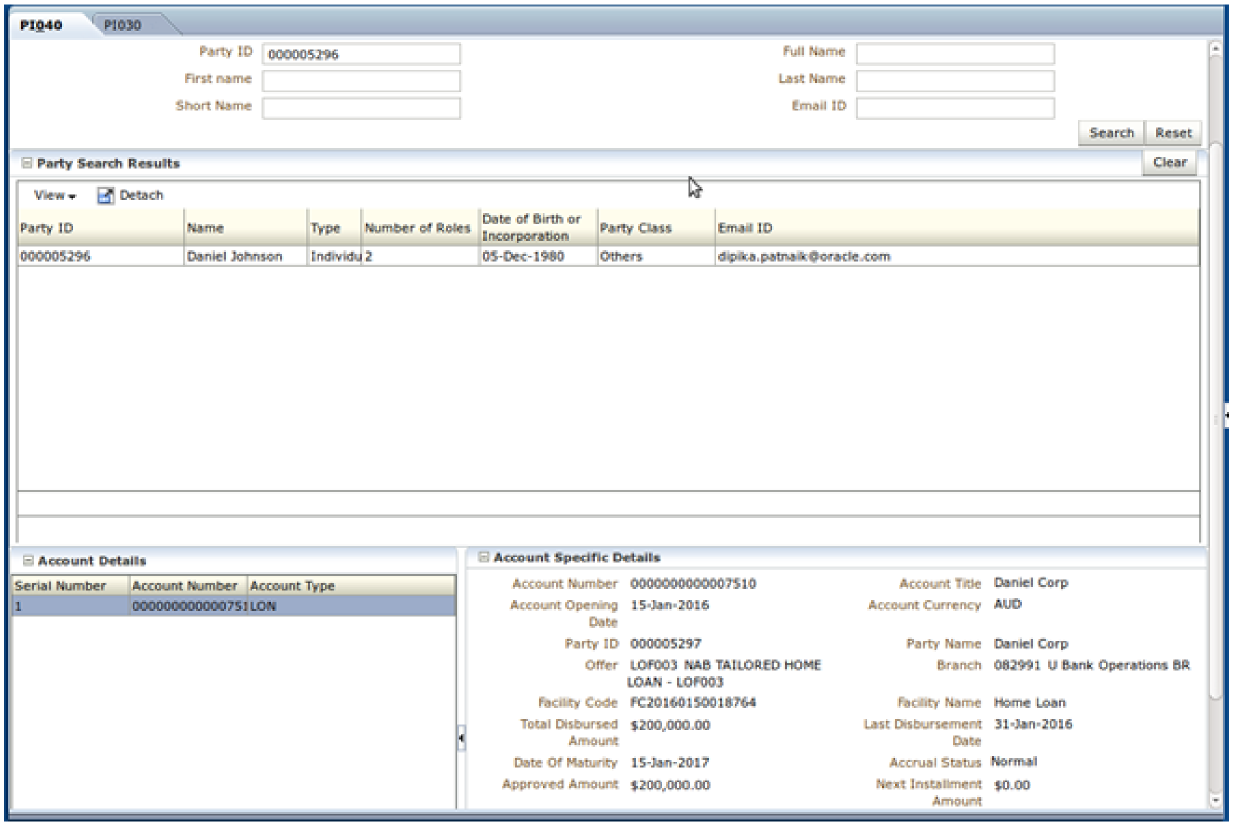
5.7 Customization Layer Use Cases

5.7.1 Adding a UI Table Component to the Screen

This second example of customization, explains adding a table *UI Component*, which displays data to a screen.

Use Case Description: The Advanced Search screen is used to display the related accounts and their details for a party. The *Party -> On-Boarding -> Related Party* screen displays the related parties for a party. This section explains adding the table UI component used for displaying the related parties on the *Related Party* screen to the *Advanced Search* screen and populate data in this table on search and selection of a party.

Figure 5–12 Adding a UI Table Component - Party Search screen



Party ID	000005296	Full Name	
First name		Last Name	
Short Name		Email ID	

⊟ Party Search Results

Party ID	Name	Type	Number of Roles	Date of Birth or Incorporation	Party Class	Email ID
000005296	Daniel Johnson	Individu	2	05-Dec-1980	Others	dipika.patnaik@oracle.com

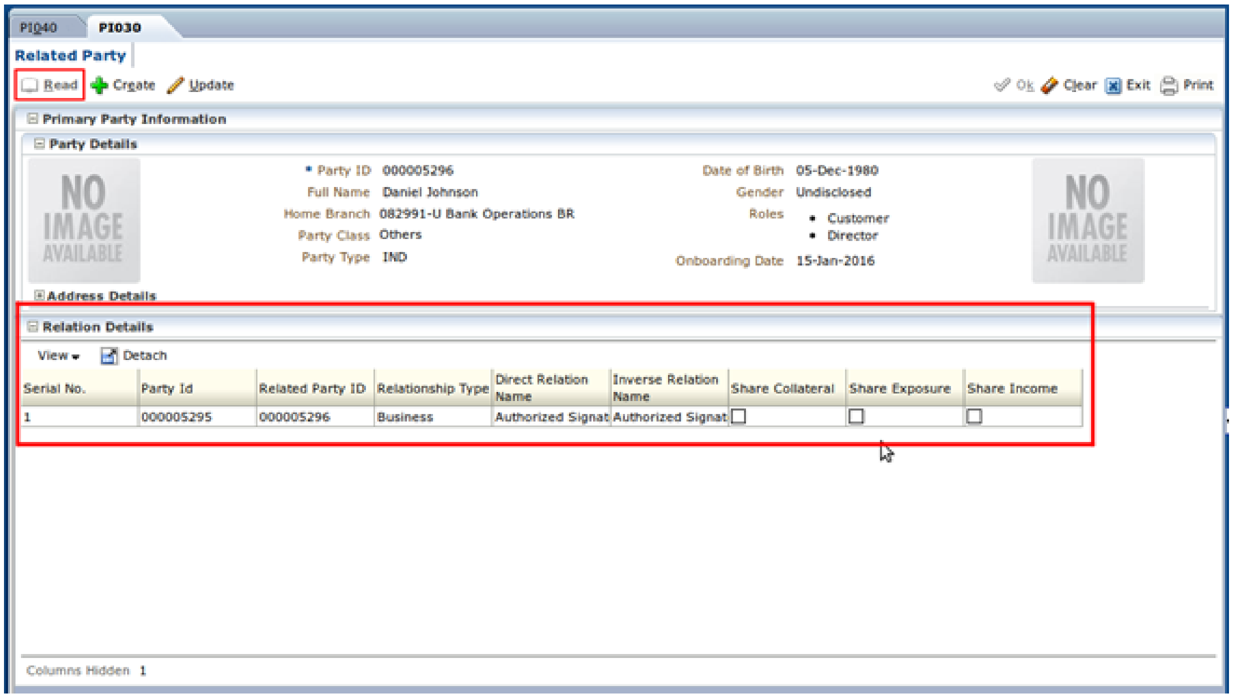
⊟ Account Details

Serial Number	Account Number	Account Type
1	000000000000751	LON

⊟ Account Specific Details

Account Number	0000000000007510	Account Title	Daniel Corp
Account Opening Date	15-Jan-2016	Account Currency	AUD
Party ID	000005297	Party Name	Daniel Corp
Offer	LOF003 NAB TAILORED HOME LOAN - LOF003	Branch	082991 U Bank Operations BR
Facility Code	FC20160150018764	Facility Name	Home Loan
Total Disbursed Amount	$200,000.00	Last Disbursement Date	31-Jan-2016
Date Of Maturity	15-Jan-2017	Accrual Status	Normal
Approved Amount	$200,000.00	Next Installment Amount	$0.00

Figure 5–13 Adding a UI Table Component - Related Party screen



⊟ Primary Party Information

⊟ Party Details

* Party ID	000005296	Date of Birth	05-Dec-1980
Full Name	Daniel Johnson	Gender	Undisclosed
Home Branch	082991-U Bank Operations BR	Roles	• Customer • Director
Party Class	Others		
Party Type	IND	Onboarding Date	15-Jan-2016

⊞ Address Details

⊟ Relation Details

Serial No.	Party Id	Related Party ID	Relationship Type	Direct Relation Name	Inverse Relation Name	Share Collateral	Share Exposure	Share Income
1	000005295	000005296	Business	Authorized Signat	Authorized Signat	☐	☐	☐

Columns Hidden 1

To create the customization as mentioned in this use case, start JDeveloper in the *Default Role* and follow these steps:

Step 1 Create Customization Project

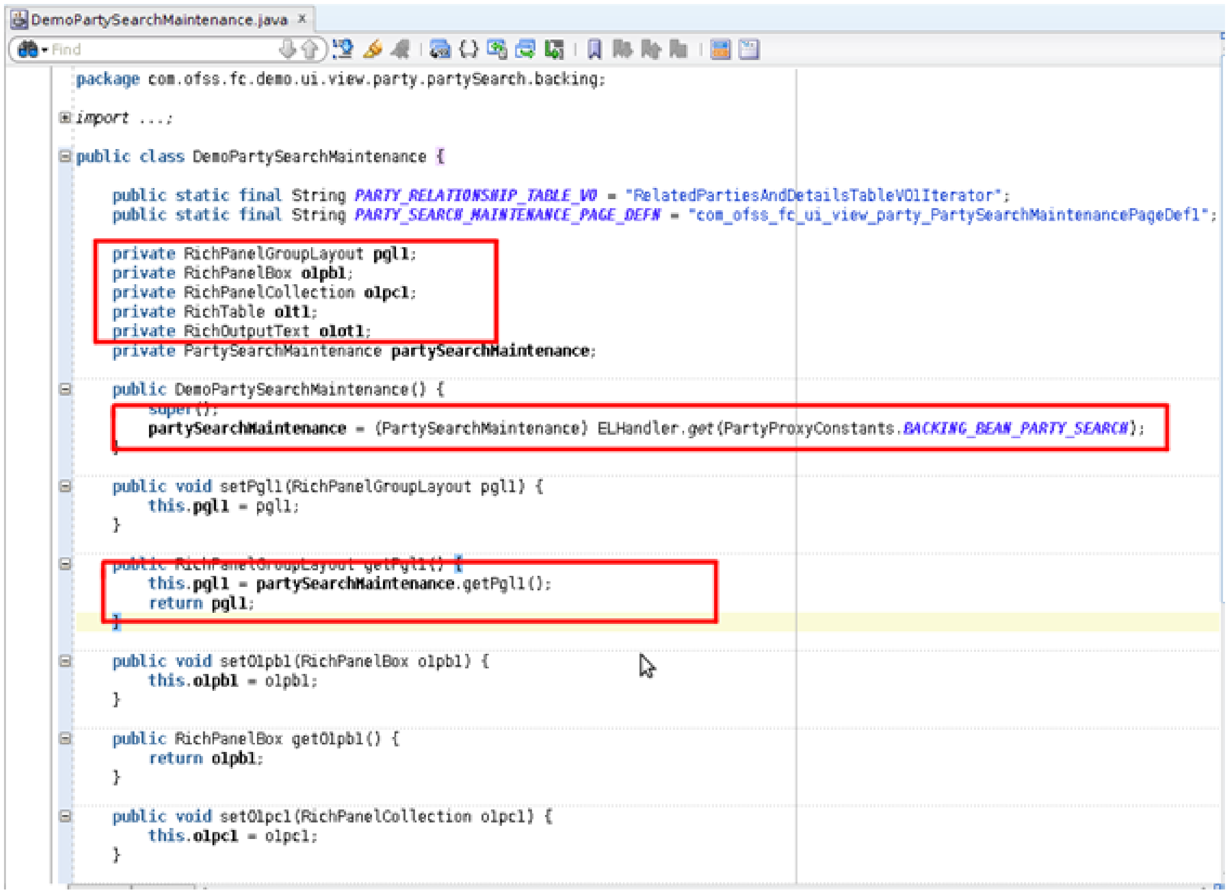
1. As mentioned in the section **Customization Project**, create a project *(com.ofss.fc.demo.ui.view.party)* to hold the customization.
2. Add the required libraries and JARS along with JAR which contains the above screen *(com.ofss.fc.ui.view.party.jar)*.
3. Enable the project for seeded customizations.

Step 2 Create Binding Bean Class

You will need to create a class which will contain the binding for the *UI Components* which will be added to the screen during customization. Create the class with the following features:

- Private members for the UI Components and public accessors for the same.
- Private member for the backing bean of the screen *(PartySearchMaintenance)* which is initialized in the constructor of this class.
- Private member for the parent UI Component of the newly added UI components and public accessors which returns the corresponding component of the backing bean.

Figure 5–14 Creating Binding Bean Class



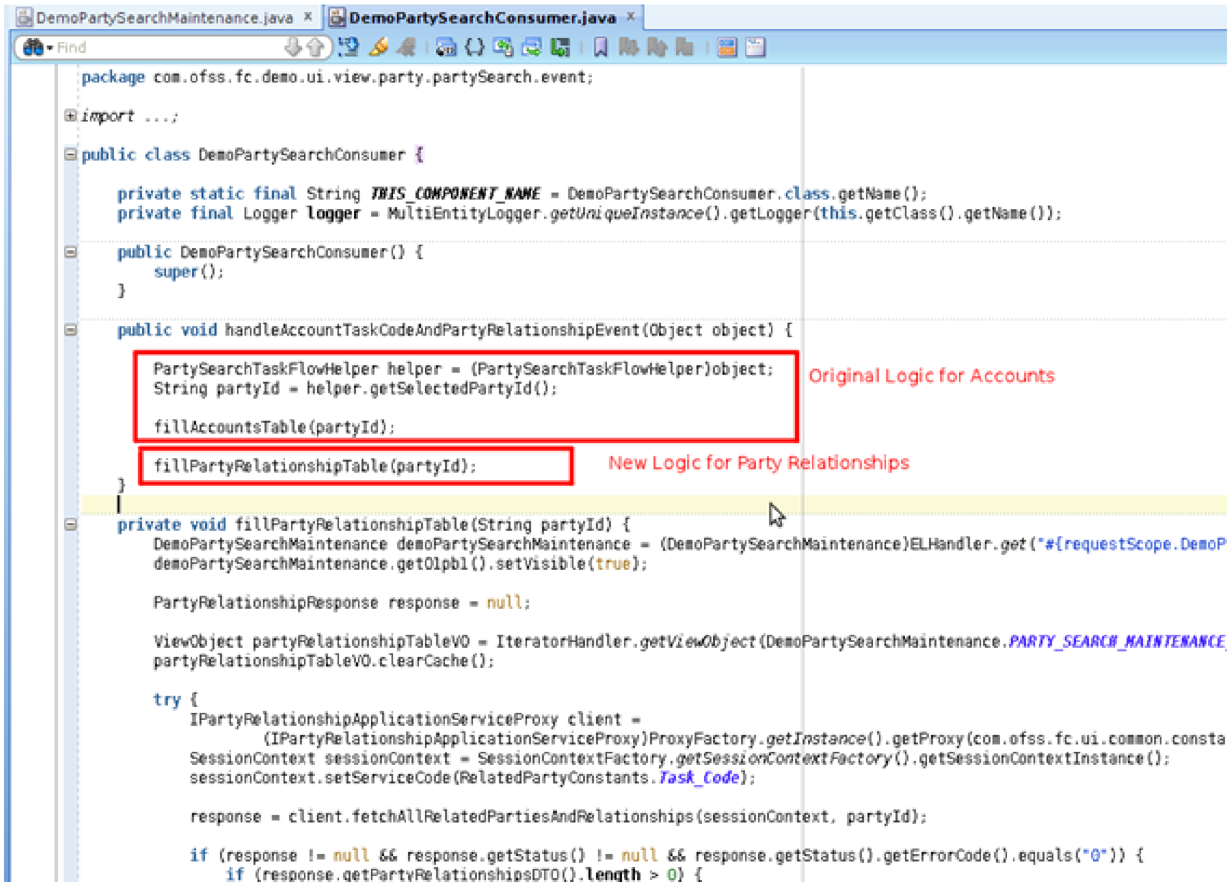
```
package com.ofss.fc.demo.ui.view.party.partySearch.backing;

import ...;

public class DemoPartySearchMaintenance {

    public static final String PARTY_RELATIONSHIP_TABLE_VO = "RelatedPartiesAndDetailsTableVO1Iterator";
    public static final String PARTY_SEARCH_MAINTENANCE_PAGE_DEFN = "com_ofss_fc_ui_view_party_PartySearchMaintenancePageDef1";

    private RichPanelGroupLayout pgl1;
    private RichPanelBox olpbl;
    private RichPanelCollection olpcl;
    private RichTable oltl;
    private RichOutputText olotl;
    private PartySearchMaintenance partySearchMaintenance;

    public DemoPartySearchMaintenance() {
        super();
        partySearchMaintenance = (PartySearchMaintenance) ELHandler.get(PartyProxyConstants.BACKING_BEAN_PARTY_SEARCH);
    }

    public void setPgl1(RichPanelGroupLayout pgl1) {
        this.pgl1 = pgl1;
    }

    public RichPanelGroupLayout getPgl1() {
        this.pgl1 = partySearchMaintenance.getPgl1();
        return pgl1;
    }

    public void setOlpbl(RichPanelBox olpbl) {
        this.olpbl = olpbl;
    }

    public RichPanelBox getOlpbl() {
        return olpbl;
    }

    public void setOlpcl(RichPanelCollection olpcl) {
        this.olpcl = olpcl;
    }
```

Step 3 Create Event Consumer Class

You will need to create a class which contains the business logic for populating the table UI component with the related parties' data. The search and selection of a party in the *Advanced Search* screen raises an event. By binding this event consumer class to the party's selection event, the business logic for populating the related party's data will be executed automatically on selection of a party by the user.

The original event consumer class bound to this event contains the business logic for populating the accounts data. Since your event consumer class would be over-riding the original binding, you will need to incorporate the original business logic for populating the accounts data in your event consumer class.

Figure 5–15 Create Event Consumer Class

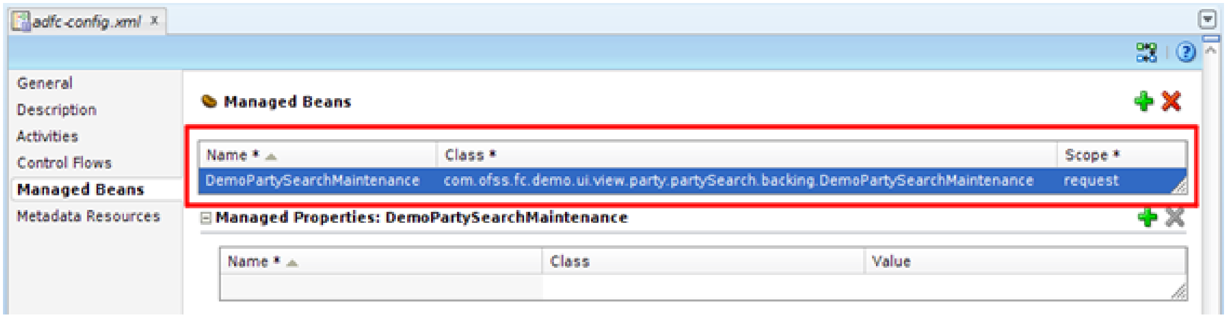

DemoPartySearchMaintenance.java
DemoPartySearchConsumer.java
Find
Original Logic for Accounts
New Logic for Party Relationships

```
package com.ofss.fc.demo.ui.view.party.partySearch.event;

import ...;

public class DemoPartySearchConsumer {

    private static final String THIS_COMPONENT_NAME = DemoPartySearchConsumer.class.getName();
    private final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(this.getClass().getName());

    public DemoPartySearchConsumer() {
        super();
    }

    public void handleAccountTaskCodeAndPartyRelationshipEvent(Object object) {

        PartySearchTaskFlowHelper helper = (PartySearchTaskFlowHelper)object;
        String partyId = helper.getSelectedPartyId();

        fillAccountsTable(partyId);

        fillPartyRelationshipTable(partyId);
    }

    private void fillPartyRelationshipTable(String partyId) {
        DemoPartySearchMaintenance demoPartySearchMaintenance = (DemoPartySearchMaintenance)ELHandler.get("#{requestScope.DemoP
        demoPartySearchMaintenance.getOlpbl().setVisible(true);

        PartyRelationshipResponse response = null;

        ViewObject partyRelationshipTableVO = IteratorHandler.getViewObject(DemoPartySearchMaintenance.PARTY_SEARCH_MAINTENANCE
        partyRelationshipTableVO.clearCache();

        try {
            IPartyRelationshipApplicationServiceProxy client =
                    (IPartyRelationshipApplicationServiceProxy)ProxyFactory.getInstance().getProxy(com.ofss.fc.ui.common.consta
            SessionContext sessionContext = SessionContextFactory.getSessionContextFactory().getSessionContextInstance();
            sessionContext.setServiceCode(RelatedPartyConstants.Task_Code);

            response = client.fetchAllRelatedPartiesAndRelationships(sessionContext, partyId);

            if (response != null && response.getStatus() != null && response.getStatus().getErrorCode().equals("0")) {
                if (response.getPartyRelationshipsDTO().length > 0) {
```

Step 4 Create Managed Bean

You will need to register the binding bean class as a managed bean. Open the project's adfc-config.xml which is present in the WEB-INF folder. In the Managed Beans tab, add the binding bean class as a managed bean with request scope as follows:

Figure 5–16 Creating Managed Bean



Name *	Class *	Scope *
DemoPartySearchMaintenance	com.ofss.fc.demo.ui.view.party.partySearch.backing.DemoPartySearchMaintenance	request

Managed Properties: DemoPartySearchMaintenance

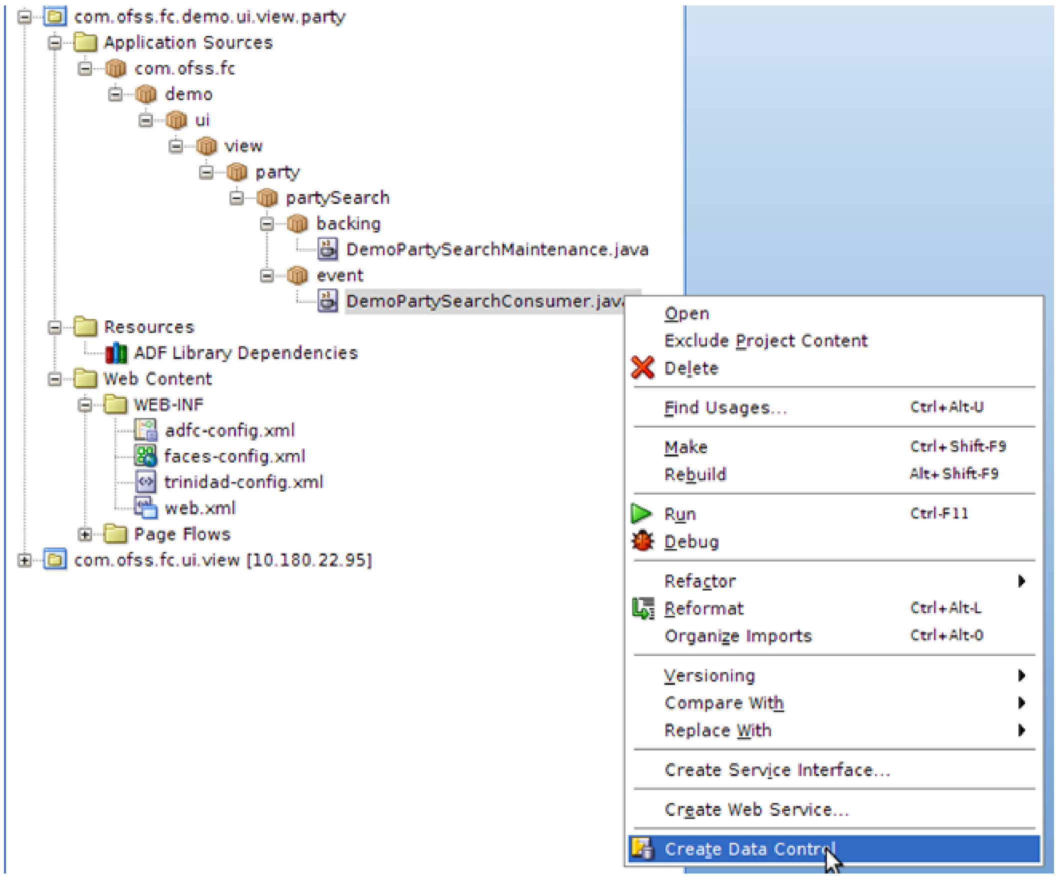
Name *	Class	Value

Step 5 Create Data Control

For the event consumer class's method to be exposed as an event handler, you will need to create a *data control* for this class.

1. In the *Application Navigator*, right-click the event consumer Java file and create data control.
2. On creation of data control, an XML file is generated for the class and a *DataControls.dcx* file is generated containing the information about the data controls present in the project. You will be able to see the event consumer data control in the *Data Controls* tab.

Figure 5–17 Create Data Control



3. Restart JDeveloper in the *Customization Developer Role* to edit the customizations.
4. Ensure that the appropriate *Customization Context* is selected.

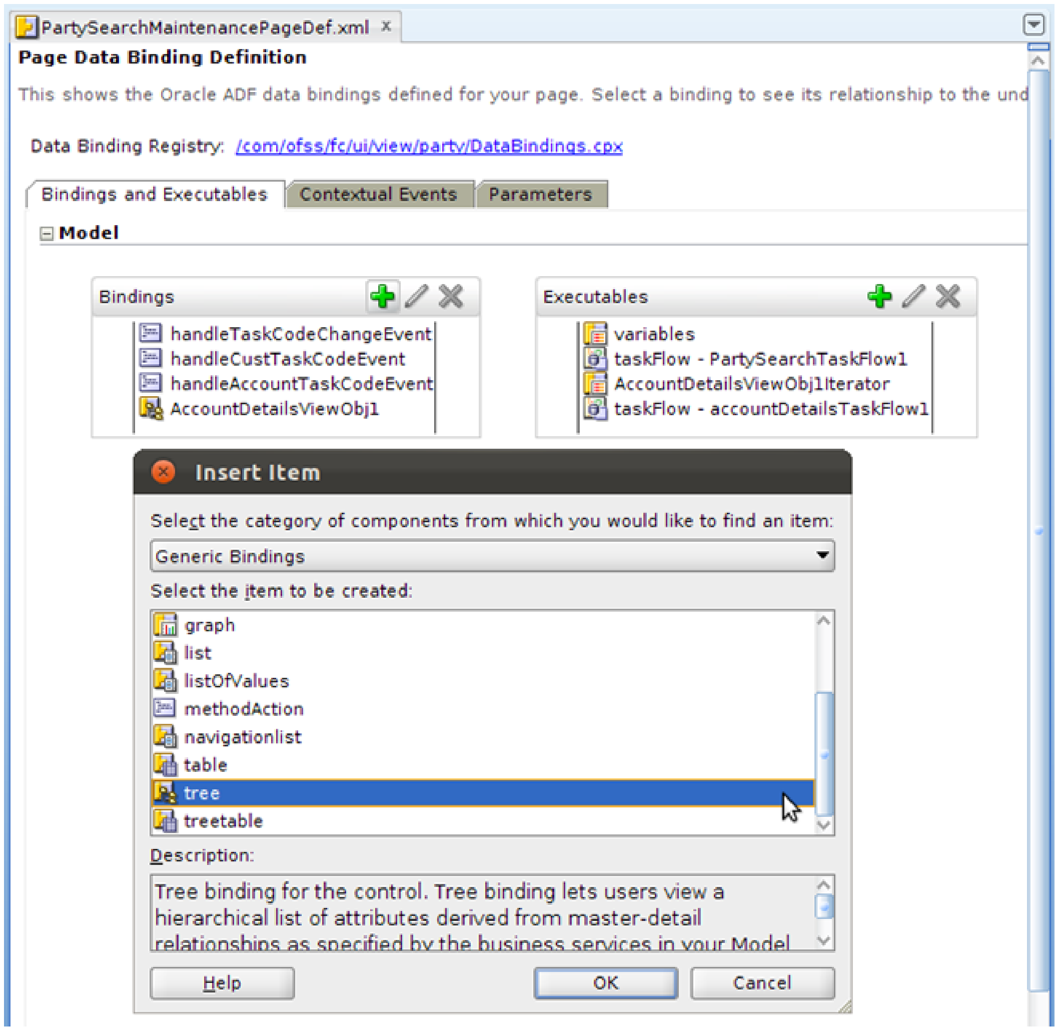
Step 6 Add View Object Binding to Page Definition

You will need to add the view object binding to the page definition of the screen. To open the page definition of the screen, follow these steps:

1. In the Application Navigator, open the Navigator Display Options for Projects tab and check the Show Libraries option.
2. In the navigator tree, locate the JAR that contains the screen (*com.ofss.fc.ui.view.party.jar*).
3. Inside this JAR, locate and open the page definition XML (*com.ofss.fc.ui.view.party.partySearch.pageDefn.PartySearchMaintenancePageDef.xml*)
4. After opening the page definition XML, add a tree binding for the view object

(RelatedPartiesAndDetailsTableVO1) as follows:

Figure 5–18 Adding View Object Binding to Page Definition - Add Tree Binding


PartySearchMaintenancePageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the und

Data Binding Registry: /com/ofss/fc/ui/view/party/DataBindings.cpx

Bindings and Executables | Contextual Events | Parameters

Model

Bindings	Executables
handleTaskCodeChangeEvent	variables
handleCustTaskCodeEvent	taskFlow - PartySearchTaskFlow1
handleAccountTaskCodeEvent	AccountDetailsViewObj1Iterator
AccountDetailsViewObj1	taskFlow - accountDetailsTaskFlow1

Insert Item

Select the category of components from which you would like to find an item:

Generic Bindings

Select the item to be created:

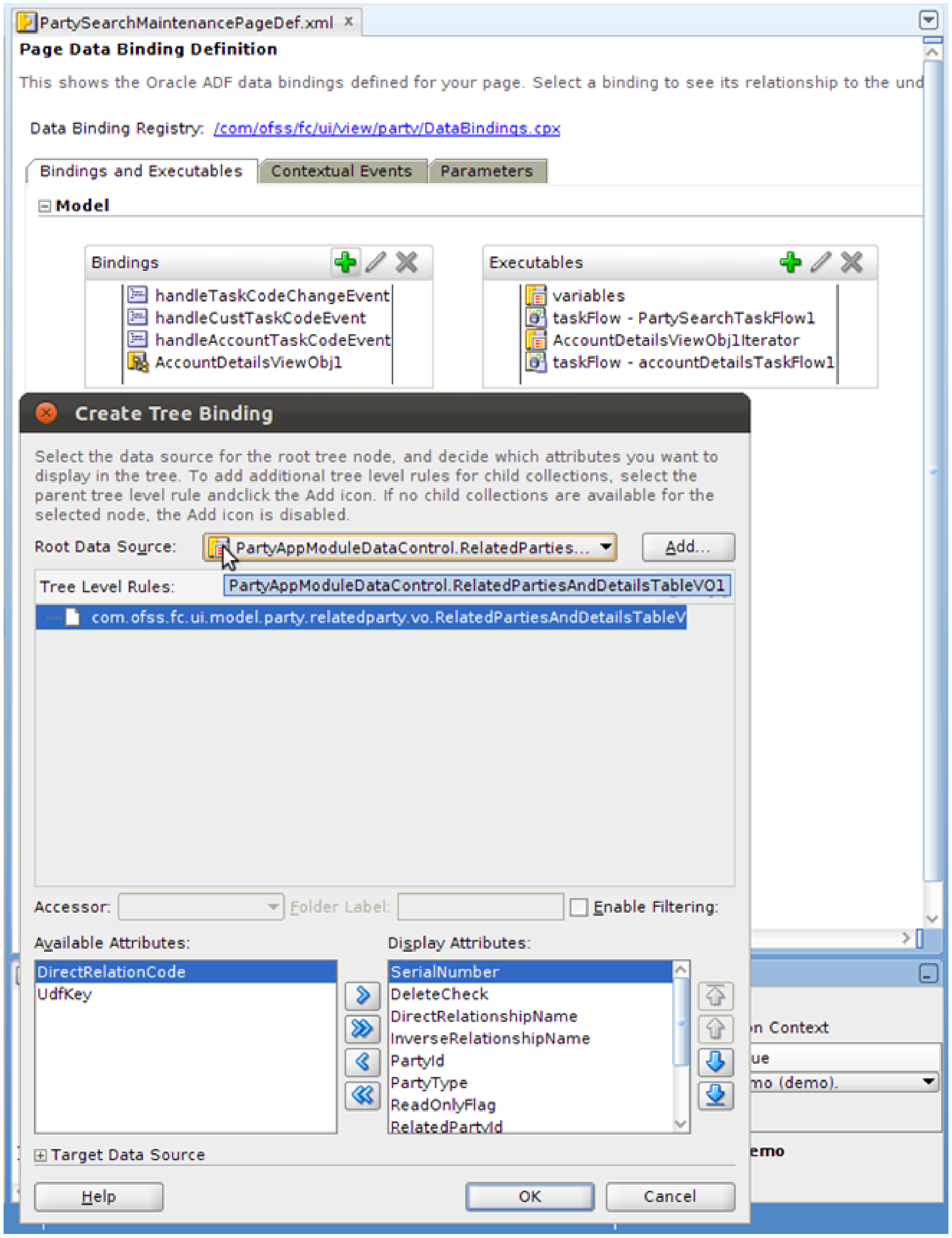
- graph
- list
- listOfValues
- methodAction
- navigationlist
- table
- tree
- treetable

Description:

Tree binding for the control. Tree binding lets users view a hierarchical list of attributes derived from master-detail relationships as specified by the business services in your Model

Help | OK | Cancel

Figure 5–19 Adding View Object Binding to Page Definition - Update Root Data Source


PartySearchMaintenancePageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the und

Data Binding Registry: /com/ofss/fc/ui/view/party/DataBindings.cpx

Bindings and Executables | Contextual Events | Parameters

Model

Bindings:
- handleTaskCodeChangeEvent
- handleCustTaskCodeEvent
- handleAccountTaskCodeEvent
- AccountDetailsViewObj1

Executables:
- variables
- taskFlow - PartySearchTaskFlow1
- AccountDetailsViewObj1Iterator
- taskFlow - accountDetailsTaskFlow1

Create Tree Binding

Select the data source for the root tree node, and decide which attributes you want to display in the tree. To add additional tree level rules for child collections, select the parent tree level rule andclick the Add icon. If no child collections are available for the selected node, the Add icon is disabled.

Root Data Source: PartyAppModuleDataControl.RelatedParties...

Tree Level Rules: PartyAppModuleDataControl.RelatedPartiesAndDetailsTableVO1

com.ofss.fc.ui.model.party.relatedparty.vo.RelatedPartiesAndDetailsTableV

Accessor: Folder Label: Enable Filtering:

Available Attributes:
- DirectRelationCode
- UdfKey

Display Attributes:
- SerialNumber
- DeleteCheck
- DirectRelationshipName
- InverseRelationshipName
- PartyId
- PartyType
- ReadOnlyFlag
- RelatedPartyId

Target Data Source

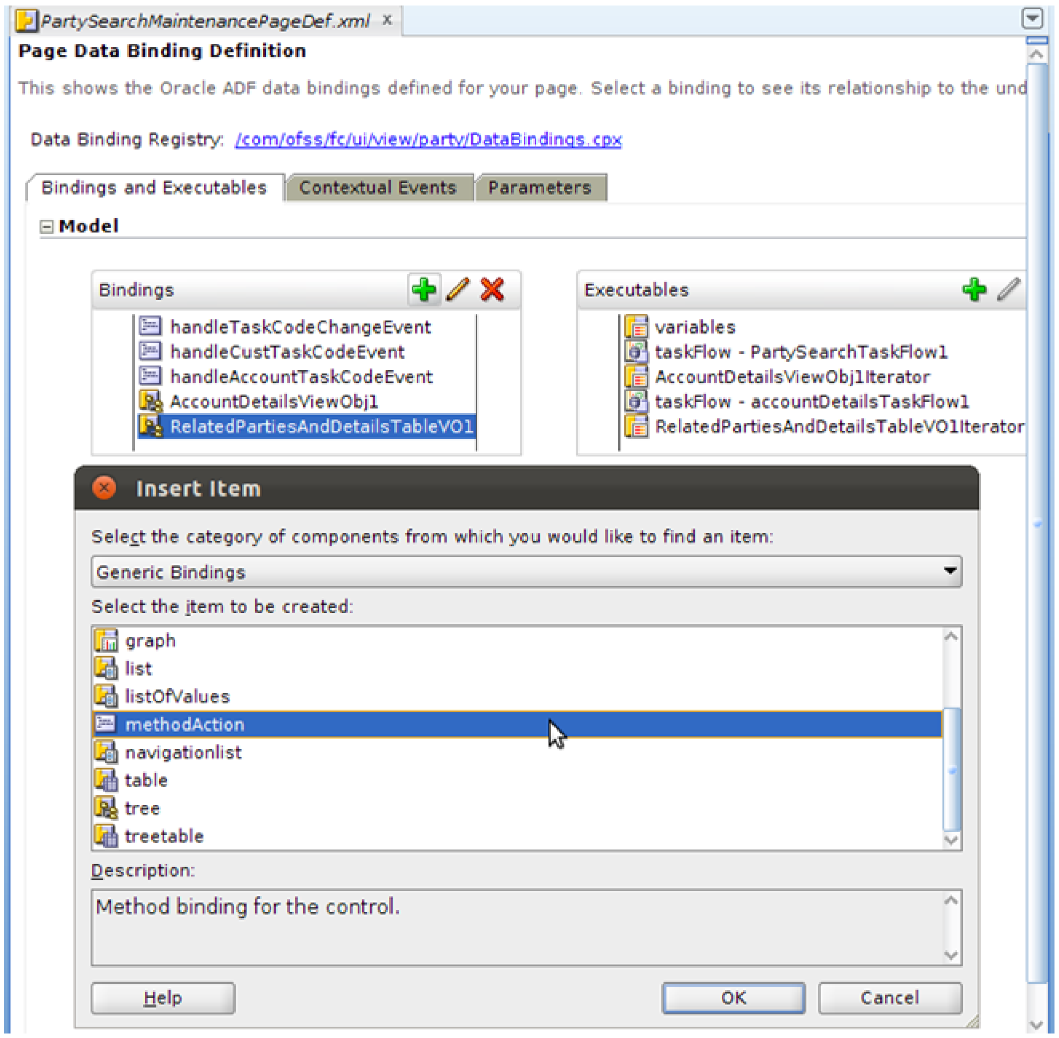
5. In Root Data Source, locate the view object which is present in the *PartyAppModuleDataControl*. Select the required display attributes and click **OK**.

Step 7 Add Method Action Binding to the Page Definition

You will need to add the method action binding for the event consumer data control to the page definition of the screen.

1. After opening the page definition XML, add the method action binding for the *DemoPartySearchConsumer* data control to the page definition as follows:

Figure 5–20 Page Data Binding Definition - Insert Item



PartySearchMaintenancePageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the und

Data Binding Registry: /com/ofss/fc/ui/view/party/DataBindings.cpx

Bindings:
- handleTaskCodeChangeEvent
- handleCustTaskCodeEvent
- handleAccountTaskCodeEvent
- AccountDetailsViewObj1
- RelatedPartiesAndDetailsTableVO1

Executables:
- variables
- taskFlow - PartySearchTaskFlow1
- AccountDetailsViewObj1Iterator
- taskFlow - accountDetailsTaskFlow1
- RelatedPartiesAndDetailsTableVO1Iterator

Insert Item

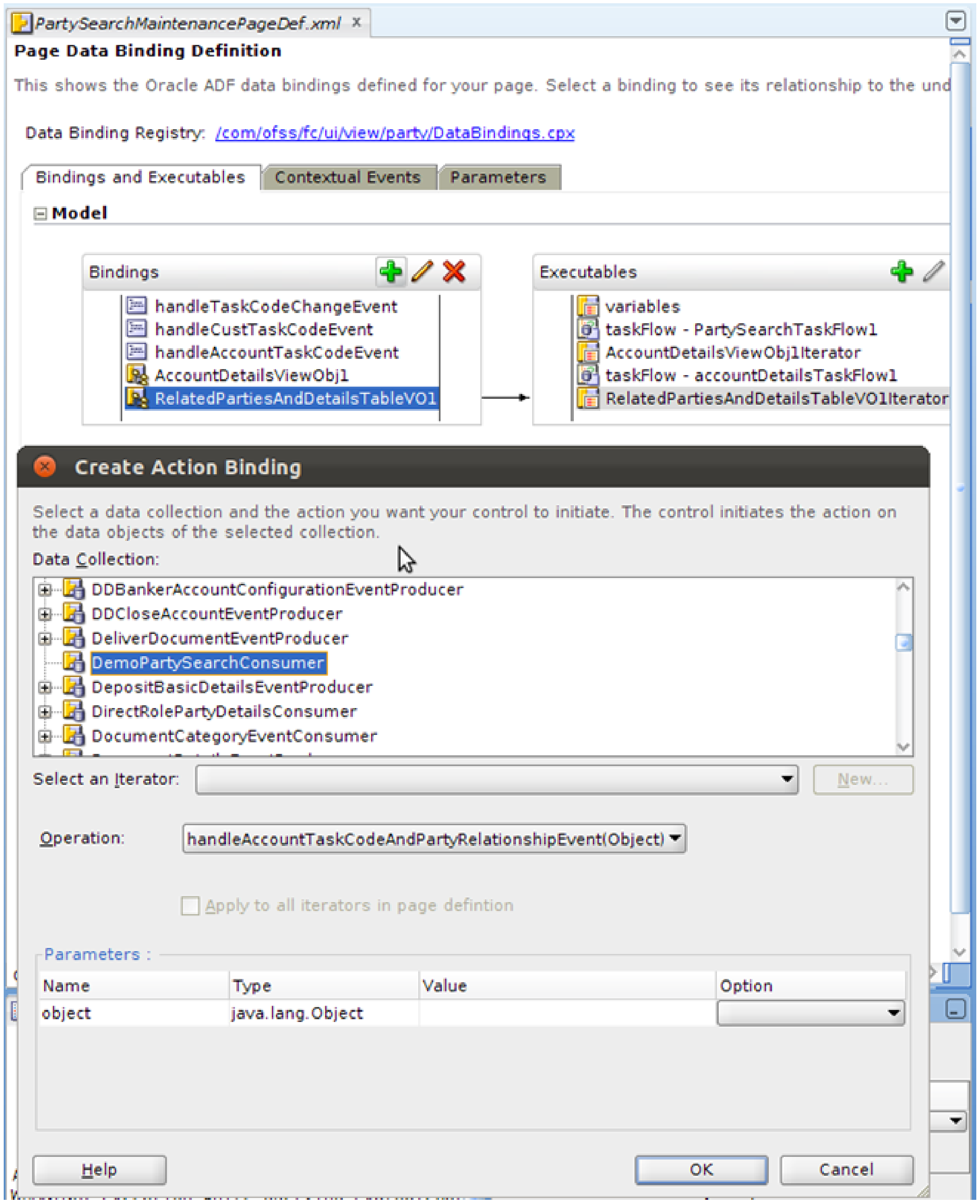
Select the category of components from which you would like to find an item: Generic Bindings

Select the item to be created:
- graph
- list
- listOfValues
- methodAction
- navigationlist
- table
- tree
- treetable

Description: Method binding for the control.

2. Browse and locate the data control and click **OK**.

Figure 5–21 Page Data Binding Definition - Create Action Binding



PartySearchMaintenancePageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the und

Data Binding Registry: /com/ofss/fc/ui/view/party/DataBindings.cpx

Bindings:
- handleTaskCodeChangeEvent
- handleCustTaskCodeEvent
- handleAccountTaskCodeEvent
- AccountDetailsViewObj1
- RelatedPartiesAndDetailsTableVO1

Executables:
- variables
- taskFlow - PartySearchTaskFlow1
- AccountDetailsViewObj1Iterator
- taskFlow - accountDetailsTaskFlow1
- RelatedPartiesAndDetailsTableVO1Iterator

Create Action Binding

Select a data collection and the action you want your control to initiate. The control initiates the action on the data objects of the selected collection.

Data Collection:
- DDBankerAccountConfigurationEventProducer
- DDCloseAccountEventProducer
- DeliverDocumentEventProducer
- DemoPartySearchConsumer
- DepositBasicDetailsEventProducer
- DirectRolePartyDetailsConsumer
- DocumentCategoryEventConsumer

Select an Iterator:

Operation: handleAccountTaskCodeAndPartyRelationshipEvent(Object)

Apply to all iterators in page defintion

Parameters :

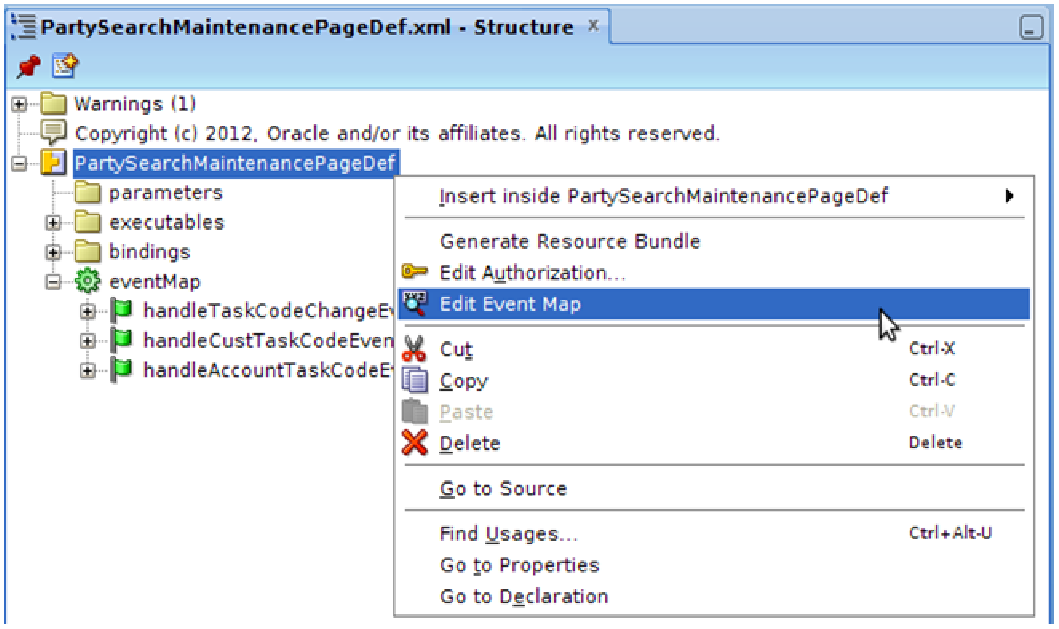
Name	Type	Value	Option
object	java.lang.Object		

Step 8 Edit Event Map

You will need to map the *Event Producer* for the party selection event to the **Event Consumer** defined by you in the page definition.

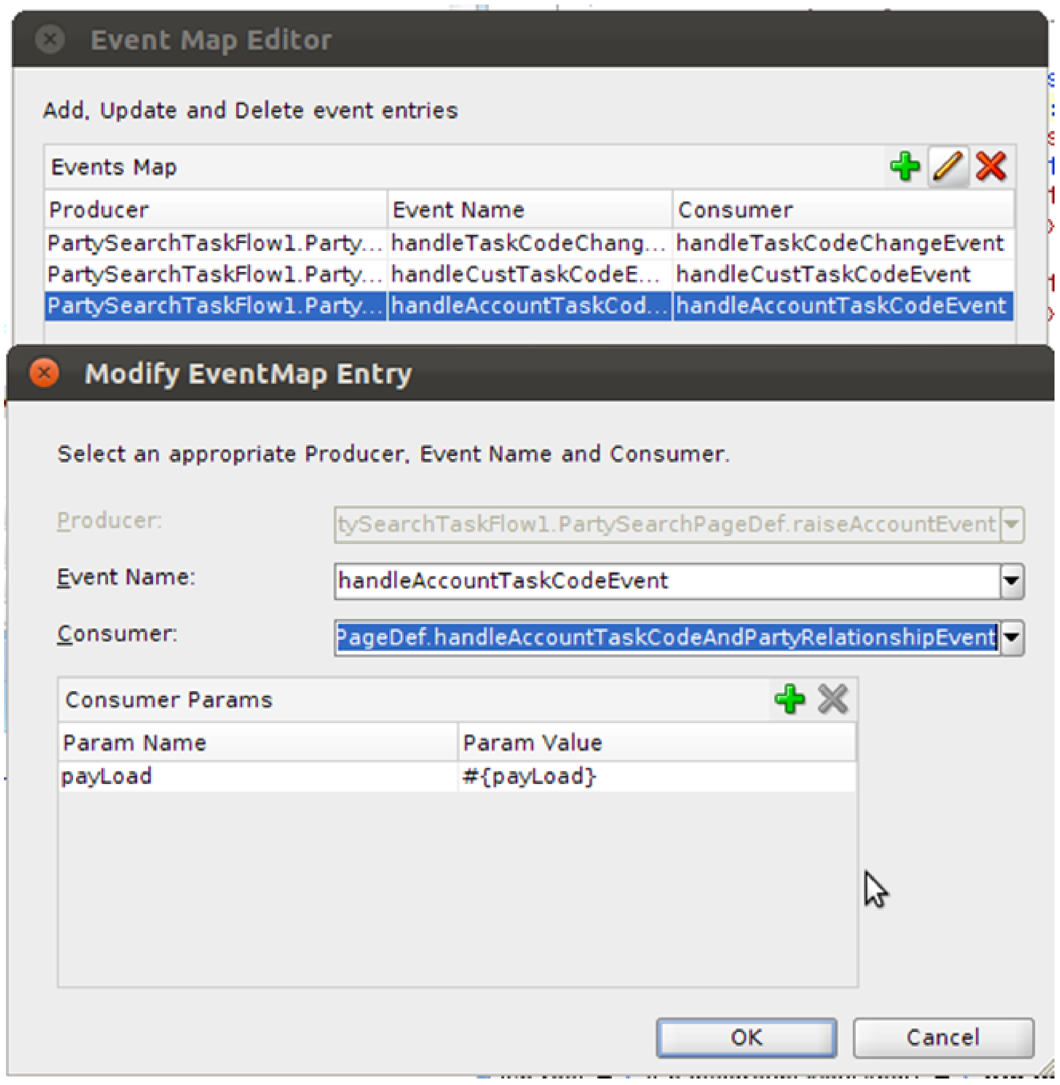
1. In the *Application Navigator*, select the page definition XML file.
2. In the *Structure panel* of JDeveloper, right-click the page definition XML and select *Edit Event Map*.

Figure 5–22 Edit Event Map



3. In the **Event Map Editor** panel, edit the mapping for the required event.
4. Select the newly added Event Consumer's method.

Figure 5–23 Event Map Editor


Event Map Editor
Add, Update and Delete event entries
Events Map
Modify EventMap Entry
Select an appropriate Producer, Event Name and Consumer.
Consumer Params
OK
Cancel

Producer	Event Name	Consumer
PartySearchTaskFlow1.Party...	handleTaskCodeChang...	handleTaskCodeChangeEvent
PartySearchTaskFlow1.Party...	handleCustTaskCodeE...	handleCustTaskCodeEvent
PartySearchTaskFlow1.Party...	handleAccountTaskCod...	handleAccountTaskCodeEvent

Producer: tySearchTaskFlow1.PartySearchPageDef.raiseAccountEvent

Event Name: handleAccountTaskCodeEvent

Consumer: PageDef.handleAccountTaskCodeAndPartyRelationshipEvent

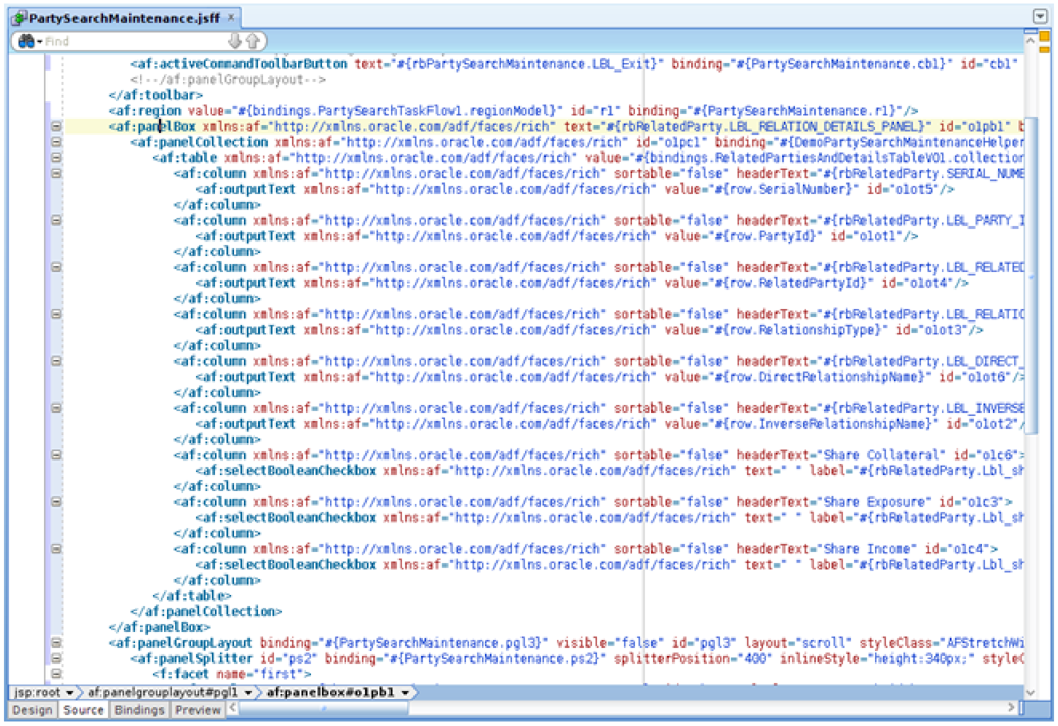
Param Name	Param Value
payLoad	#{payLoad}

Step 9 Add UI Components to Screen

After making the required changes to page definition of the screen, you will need to add the UI components to the screen JSFF. After opening the JSFF for the screen *(com.ofss.fc.ui.view.party.partySearch.PartySearchMaintenance.jsff)*, follow these steps:

1. Drag and drop the *Panel Box*, *Panel Collection* and *Table* components onto the screen.
2. Set the required columns for the *Table* component.
3. Drag and drop the *Output Text* or *Check Box* components as required inside the columns.
4. For each component, set the required attributes using the *Property Inspector* panel of JDeveloper.
5. Add the binding for required components to the binding bean members.
6. Add the view object binding to the *Table* component.
7. Save changes made to the JSFF.

Figure 5–24 Add UI Components to Screen

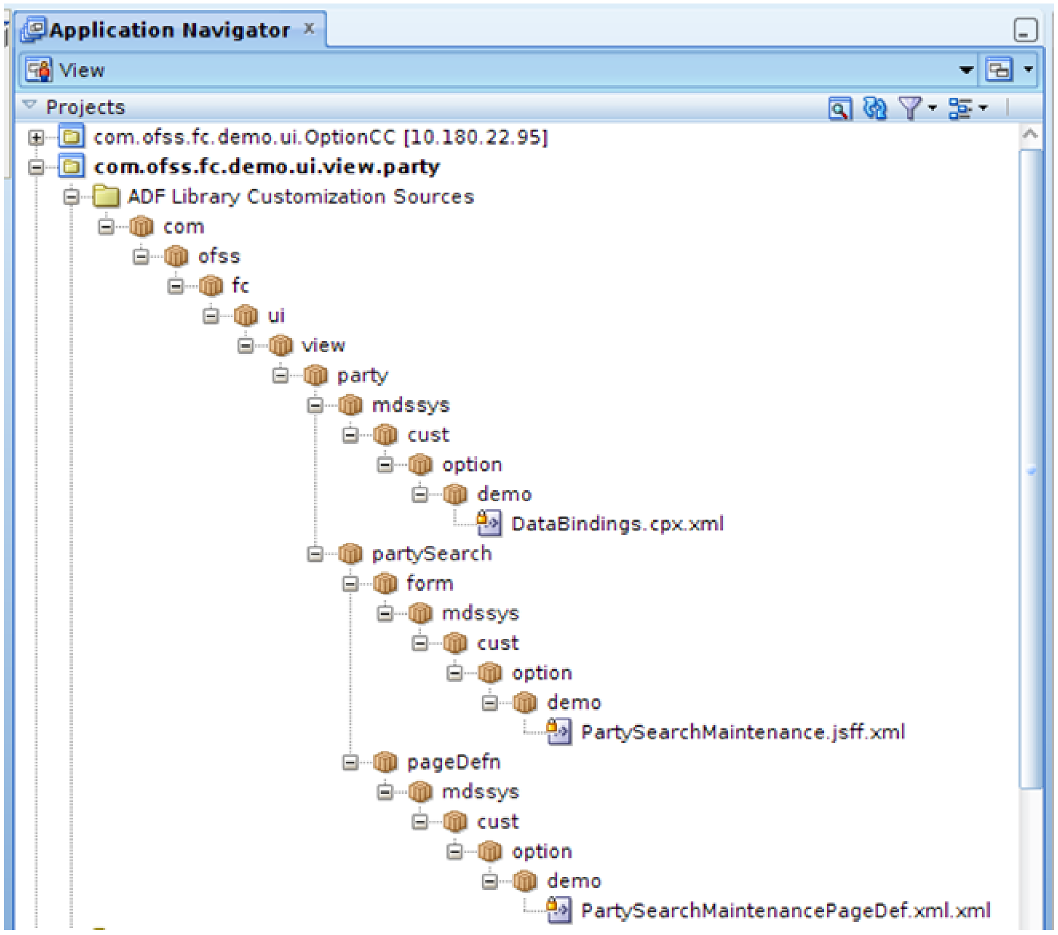


```
      <af:activeCommandToolbarButton text="#{rbPartySearchMaintenance.LBL_Exit}" binding="#{PartySearchMaintenance.cbl}" id="cbl"
      <!--/af:panelGroupLayout-->
    </af:toolbar>
    <af:region value="#{bindings.PartySearchTaskFlow1.regionModel}" id="r1" binding="#{PartySearchMaintenance.r1}"/>
    <af:panelBox xmlns:af="http://xmlns.oracle.com/adf/faces/rich" text="#{rbRelatedParty.LBL_RELATION_DETAILS_PANEL}" id="o1pb1" b
      <af:panelCollection xmlns:af="http://xmlns.oracle.com/adf/faces/rich" id="o1pc1" binding="#{DemoPartySearchMaintenanceHelper
        <af:table xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{bindings.RelatedPartiesAndDetailsTableVO1.collection
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.SERIAL_NUME
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.SerialNumber}" id="o1ot5"/>
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.LBL_PARTY_I
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.PartyId}" id="o1ot1"/>
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.LBL_RELATEI
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.RelatedPartyId}" id="o1ot4"/>
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.LBL_RELATIC
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.RelationshipType}" id="o1ot3"/>
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.LBL_DIRECT_
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.DirectRelationshipName}" id="o1ot6"/>
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="#{rbRelatedParty.LBL_INVERSE
            <af:outputText xmlns:af="http://xmlns.oracle.com/adf/faces/rich" value="#{row.InverseRelationshipName}" id="o1ot2"/
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="Share Collateral" id="o1c6">
            <af:selectBooleanCheckbox xmlns:af="http://xmlns.oracle.com/adf/faces/rich" text=" " label="#{rbRelatedParty.Lbl_sh
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="Share Exposure" id="o1c3">
            <af:selectBooleanCheckbox xmlns:af="http://xmlns.oracle.com/adf/faces/rich" text=" " label="#{rbRelatedParty.Lbl_sh
          </af:column>
          <af:column xmlns:af="http://xmlns.oracle.com/adf/faces/rich" sortable="false" headerText="Share Income" id="o1c4">
            <af:selectBooleanCheckbox xmlns:af="http://xmlns.oracle.com/adf/faces/rich" text=" " label="#{rbRelatedParty.Lbl_sh
          </af:column>
        </af:table>
      </af:panelCollection>
    </af:panelBox>
    <af:panelGroupLayout binding="#{PartySearchMaintenance.pgl3}" visible="false" id="pgl3" layout="scroll" styleClass="AFStretchWi
      <af:panelSplitter id="ps2" binding="#{PartySearchMaintenance.ps2}" splitterPosition="400" inlineStyle="height:340px;" style(
        <f:facet name="first">
```

After saving all these changes, you will notice that JDeveloper has created a customization XML for each of the customized entities in the *ADF Library Customizations Sources* folder packaged as per the corresponding base document's package and customization context *(Customization Layer Name & Customization Layer Value)*. These XML's store the difference between the base and customized entity. In our customization, you can see the following generated XML's:

- PartySearchMaintenancePageDef.xml for the page definition customizations.
- DataBindings.cpx.xml for the data binding (view object binding) customizations.
- PartySearchMaintenance.jsff.xml for the UI customization to the screen JSFF.

Figure 5–25 Application Navigator



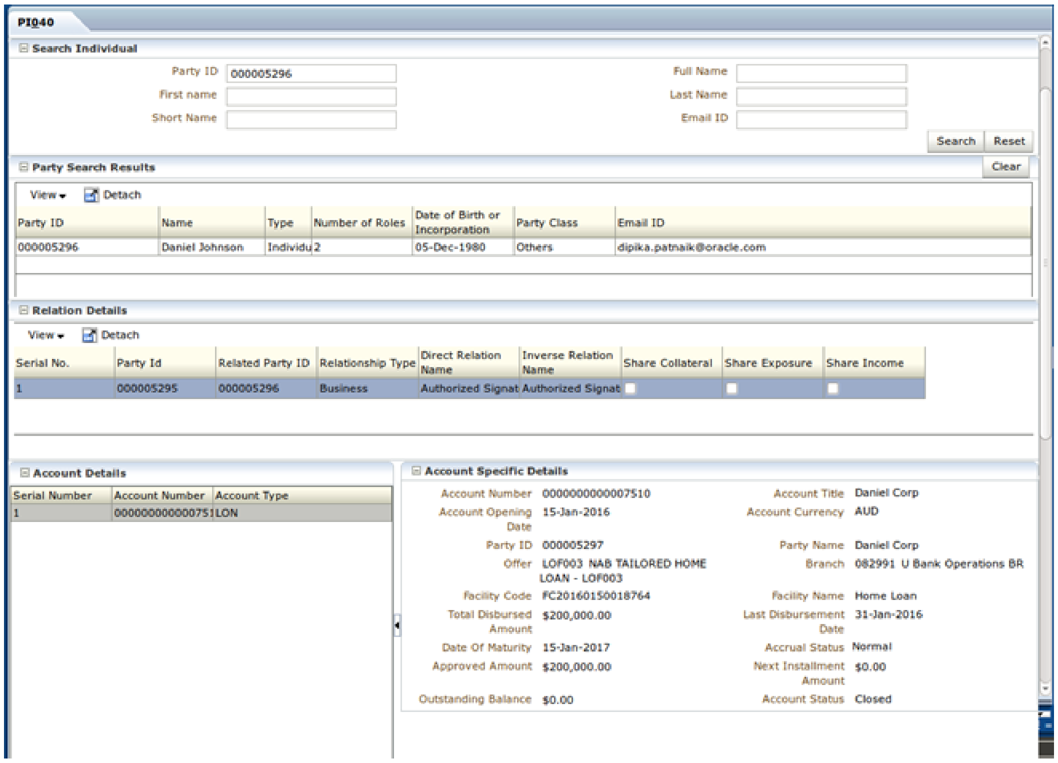
```
com.ofss.fc.demo.ui.OptionCC [10.180.22.95]
com.ofss.fc.demo.ui.view.party
  ADF Library Customization Sources
    com
      ofss
        fc
          ui
            view
              party
                mdssys
                  cust
                    option
                      demo
                        DataBindings.cpx.xml
                partySearch
                  form
                    mdssys
                      cust
                        option
                          demo
                            PartySearchMaintenance.jsff.xml
                  pageDefn
                    mdssys
                      cust
                        option
                          demo
                            PartySearchMaintenancePageDef.xml.xml
```

Step 10 Deploy Customization Project

After finishing the customization changes, exit the *Customization Developer Role* and start JDeveloper in *Default Role*. Deploy the customization project as an ADF Library JAR *(com.ofss.fc.demo.ui.view.party.jar)*.

1. Go to the **Project Properties** of the main application project and in the *Libraries* and *Classpath*, add the following JARS:
   - Customization Project JAR (com.ofss.fc.demo.ui.view.party.jar)
   - Customization Class JAR (com.ofss.fc.demo.ui.OptionCC.jar)
   - All dependency libraries and JARS for the project.
2. Start the application and navigate to the *Advanced Search* screen.
3. Search for a party ID and select a party from the *Party Search Results* table.
4. On selection of a party, the *Relation Details* panel containing the related party’s data is displayed.

Figure 5–26 Party Search



Search Individual

Party ID	000005296	Full Name	
First name		Last Name	
Short Name		Email ID	

Party Search Results

Party ID	Name	Type	Number of Roles	Date of Birth or Incorporation	Party Class	Email ID
000005296	Daniel Johnson	Individu	2	05-Dec-1980	Others	dipika.patnaik@oracle.com

Relation Details

Serial No.	Party Id	Related Party ID	Relationship Type	Direct Relation Name	Inverse Relation Name	Share Collateral	Share Exposure	Share Income
1	000005295	000005296	Business	Authorized Signat	Authorized Signat	☐	☐	☐

Account Details

Serial Number	Account Number	Account Type
1	00000000000751	LON

Account Specific Details

Account Number	0000000000007510	Account Title	Daniel Corp
Account Opening Date	15-Jan-2016	Account Currency	AUD
Party ID	000005297	Party Name	Daniel Corp
Offer	LOF003 NAB TAILORED HOME LOAN - LOF003	Branch	082991 U Bank Operations BR
Facility Code	FC20160150018764	Facility Name	Home Loan
Total Disbursed Amount	$200,000.00	Last Disbursement Date	31-Jan-2016
Date Of Maturity	15-Jan-2017	Accrual Status	Normal
Approved Amount	$200,000.00	Next Installment Amount	$0.00
Outstanding Balance	$0.00	Account Status	Closed

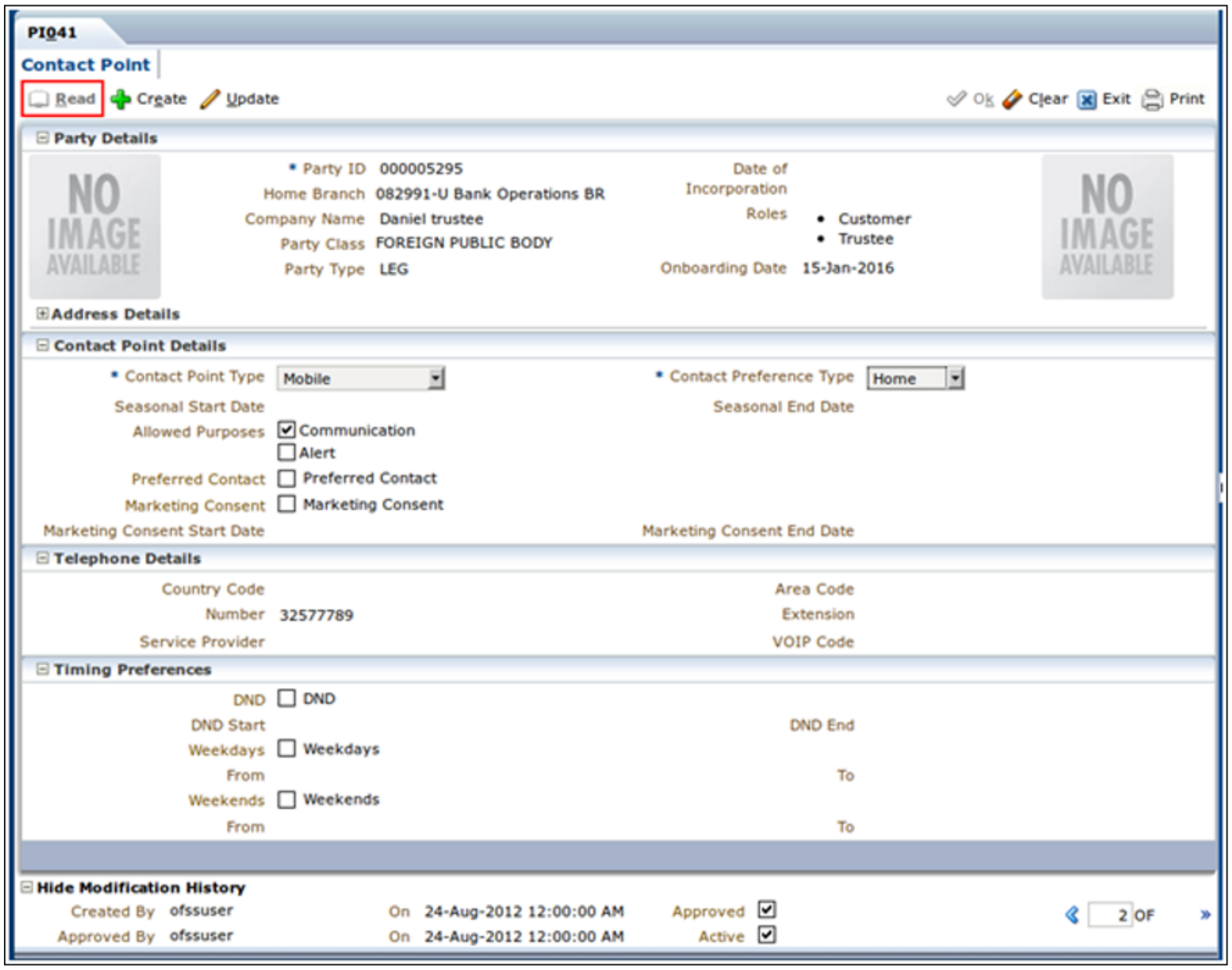
5.7.2 Approvals Framework

It is recommended to use ADF screen extensions for UI changes instead of mds in this scenario as it is easier to upgrade to new version of product however the mds approach is described below.

This third example of customization explains adding a Date Component to an existing screen to capture date input from the input. This input is saved in the database.

Use Case Description: The Party → Contact Information → Contact Point screen is used to store the various contact point details for a party. In the Contact Point Details tab, the user can select a Contact Point Type and a Contact Preference Type and provide details for the same. User will be adding a field Expiry Date as a date component to this tab. User will be adding a table to the database to save the user input for this field and services for this screen will be added or modified.

Figure 5–27 Contact Point Screen


PI041

Contact Point

Read | Create | Update | Ok | Clear | Exit | Print

Party Details

• Party ID	000005295	Date of Incorporation	
Home Branch	082991-U Bank Operations BR	Roles	• Customer • Trustee
Company Name	Daniel trustee		
Party Class	FOREIGN PUBLIC BODY		
Party Type	LEG	Onboarding Date	15-Jan-2016

Address Details

Contact Point Details

• Contact Point Type	Mobile	• Contact Preference Type	Home
Seasonal Start Date		Seasonal End Date	
Allowed Purposes	☑ Communication ☐ Alert		
Preferred Contact	☐ Preferred Contact		
Marketing Consent	☐ Marketing Consent		
Marketing Consent Start Date		Marketing Consent End Date	

Telephone Details

Country Code		Area Code	
Number	32577789	Extension	
Service Provider		VOIP Code	

Timing Preferences

DND	☐ DND		
DND Start		DND End	
Weekdays	☐ Weekdays		
From		To	
Weekends	☐ Weekends		
From		To	

Hide Modification History

Created By	ofssuser	On	24-Aug-2012 12:00:00 AM	Approved	☑
Approved By	ofssuser	On	24-Aug-2012 12:00:00 AM	Active	☑

2 OF

To create the customization as mentioned in this use case, follow these steps:

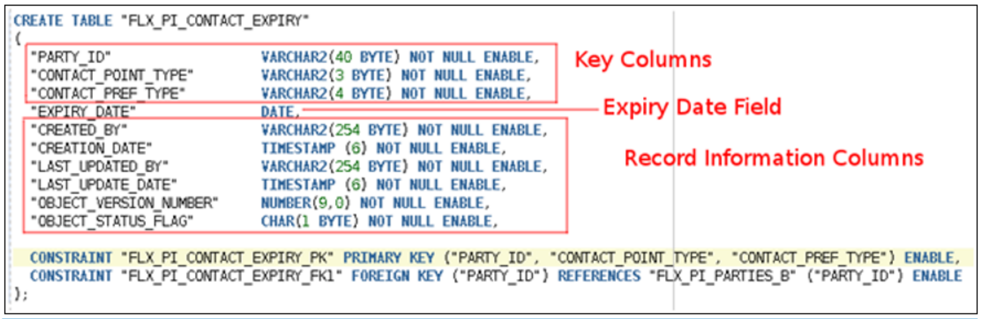
Step 1 Host Application Changes

Since in this use case you need to save the input data in the database of the application, you need to do certain modifications on the host application before creating the customizations on the client application. Following are the changes that need to be done to the host application.

Step 2 Create Table in Application Database

To save the input data for the Expiry Date field, create a table in the application database. The table will also need to have the Key columns for this field and the columns needed to store information about the record. Create appropriate primary and foreign keys for the table as well.

Figure 5–28 Create Table



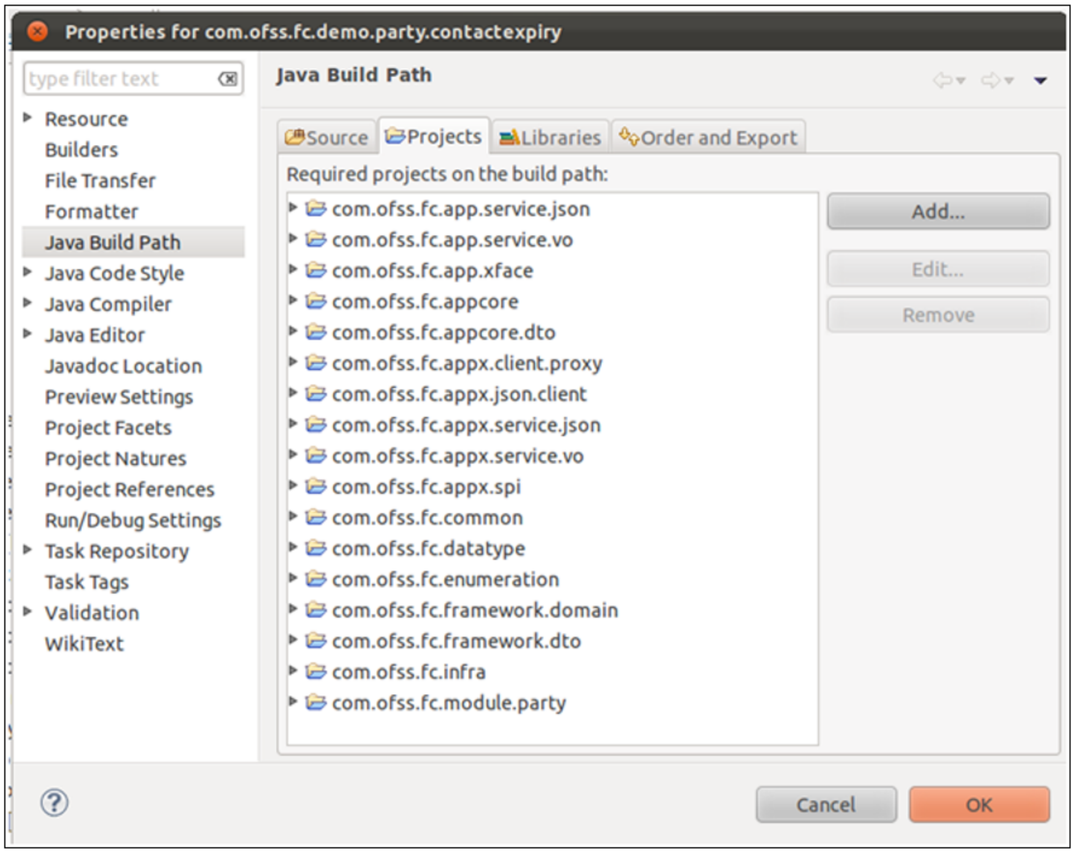
```
CREATE TABLE "FLX_PI_CONTACT_EXPIRY"
(
  "PARTY_ID"                 VARCHAR2(40 BYTE) NOT NULL ENABLE,
  "CONTACT_POINT_TYPE"       VARCHAR2(3 BYTE) NOT NULL ENABLE,
  "CONTACT_PREF_TYPE"        VARCHAR2(4 BYTE) NOT NULL ENABLE,
  "EXPIRY_DATE"              DATE,
  "CREATED_BY"               VARCHAR2(254 BYTE) NOT NULL ENABLE,
  "CREATION_DATE"            TIMESTAMP (6) NOT NULL ENABLE,
  "LAST_UPDATED_BY"          VARCHAR2(254 BYTE) NOT NULL ENABLE,
  "LAST_UPDATE_DATE"         TIMESTAMP (6) NOT NULL ENABLE,
  "OBJECT_VERSION_NUMBER"    NUMBER(9,0) NOT NULL ENABLE,
  "OBJECT_STATUS_FLAG"       CHAR(1 BYTE) NOT NULL ENABLE,

  CONSTRAINT "FLX_PI_CONTACT_EXPIRY_PK" PRIMARY KEY ("PARTY_ID", "CONTACT_POINT_TYPE", "CONTACT_PREF_TYPE") ENABLE,
  CONSTRAINT "FLX_PI_CONTACT_EXPIRY_FK1" FOREIGN KEY ("PARTY_ID") REFERENCES "FLX_PI_PARTIES_B" ("PARTY_ID") ENABLE
);
```

After creating the table, you need to create the domain object and service layers. To create these entities, follow these steps.

Step 3 Create Java Project

To contain the domain object and service layer classes, create a Java Project in eclipse. Give a title to the project (com.ofss.fc.demo.party.contactexpiry) and add the required projects to the classpath of the project.

Figure 5–29 Create Java Project



Required projects on the build path:

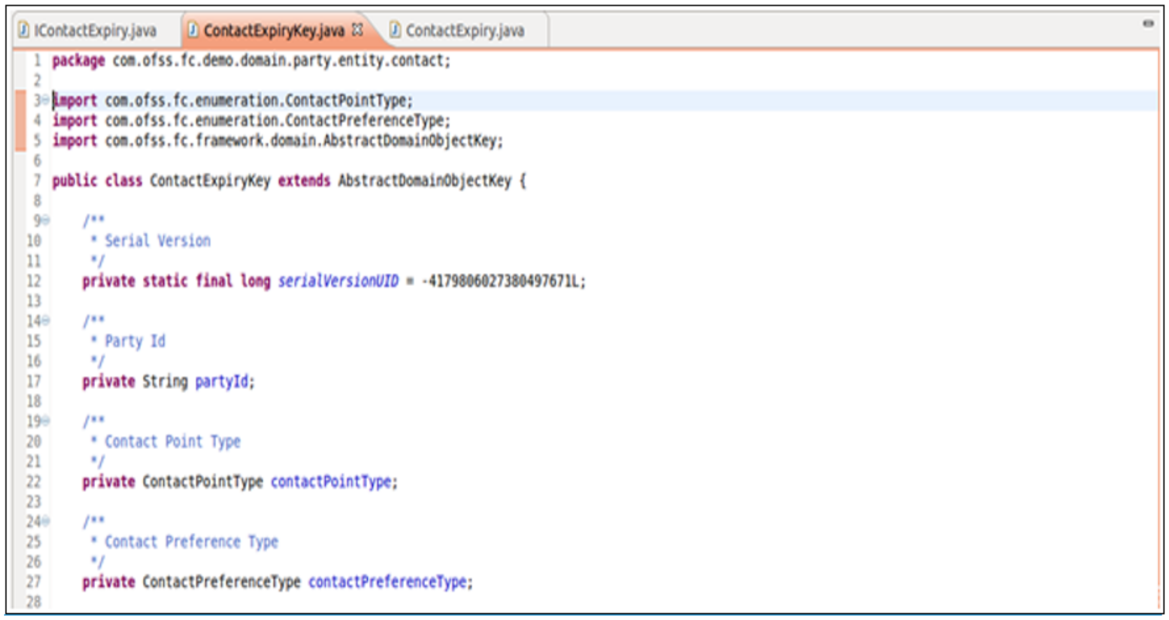
- com.ofss.fc.app.service.json
- com.ofss.fc.app.service.vo
- com.ofss.fc.app.xface
- com.ofss.fc.appcore
- com.ofss.fc.appcore.dto
- com.ofss.fc.appx.client.proxy
- com.ofss.fc.appx.json.client
- com.ofss.fc.appx.service.json
- com.ofss.fc.appx.service.vo
- com.ofss.fc.appx.spi
- com.ofss.fc.common
- com.ofss.fc.datatype
- com.ofss.fc.enumeration
- com.ofss.fc.framework.domain
- com.ofss.fc.framework.dto
- com.ofss.fc.infra
- com.ofss.fc.module.party

Step 4 Create Domain Objects

You need to create the domain objects for the newly added table. As per the structure and package conventions of OBP, create the domain objects as follows:

1. Create class (com.ofss.fc.demo.domain.party.entity.contact.ContactExpiryKey) for the key columns of the table. This class must extend the com.ofss.fc.framework.domain.AbstractDomainObject abstract class. Add the properties, getters and setters for the key columns of the table in this class. Implement the abstract methods of the superclass.

Figure 5–30 Create Domain Objects

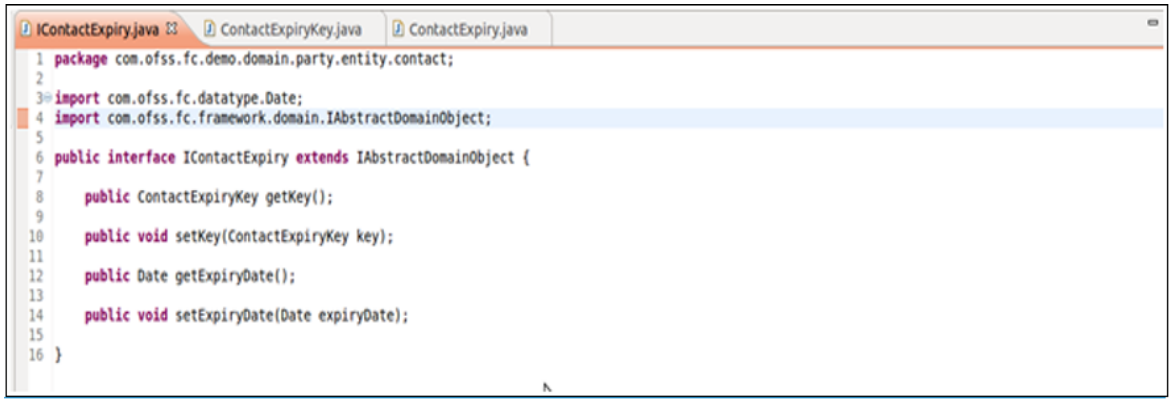


IContactExpiry.java | ContactExpiryKey.java | ContactExpiry.java

```java
package com.ofss.fc.demo.domain.party.entity.contact;

import com.ofss.fc.enumeration.ContactPointType;
import com.ofss.fc.enumeration.ContactPreferenceType;
import com.ofss.fc.framework.domain.AbstractDomainObjectKey;

public class ContactExpiryKey extends AbstractDomainObjectKey {

    /**
     * Serial Version
     */
    private static final long serialVersionUID = -4179806027380497671L;

    /**
     * Party Id
     */
    private String partyId;

    /**
     * Contact Point Type
     */
    private ContactPointType contactPointType;

    /**
     * Contact Preference Type
     */
    private ContactPreferenceType contactPreferenceType;

```

2. Create interface (com.ofss.fc.demo.domain.party.entity.contact.IContactExpiry) for the domain object class with getters and setters abstract methods for the Key domain object and the field Expiry Date. This interface must extend the interface com.ofss.fc.framework.domain.AbstractDomainObject.

Figure 5–31 Create Interface

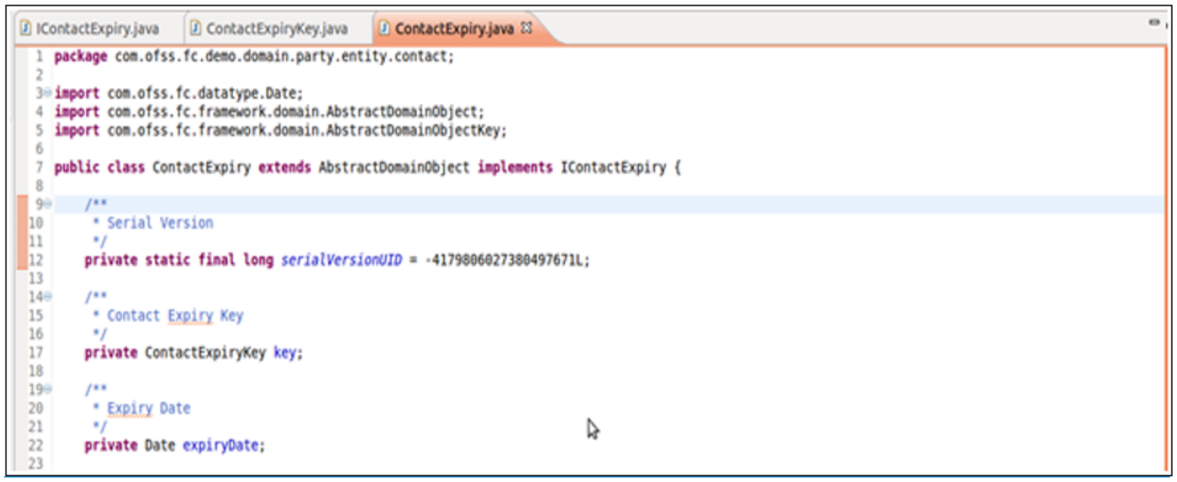


IContactExpiry.java | ContactExpiryKey.java | ContactExpiry.java

```java
package com.ofss.fc.demo.domain.party.entity.contact;

import com.ofss.fc.datatype.Date;
import com.ofss.fc.framework.domain.IAbstractDomainObject;

public interface IContactExpiry extends IAbstractDomainObject {

    public ContactExpiryKey getKey();

    public void setKey(ContactExpiryKey key);

    public Date getExpiryDate();

    public void setExpiryDate(Date expiryDate);

}
```

3. Create class (com.ofss.fc.demo.domain.party.entity.contact.ContactExpiry) for the domain object. This class must implement the previously created interface and extend com.ofss.fc.framework.domain.AbstractDomainObject abstract class. Add the properties, getters and setters for Key object and Expiry Date field. Implement the abstract methods of the superclass.

Figure 5–32 Create Class



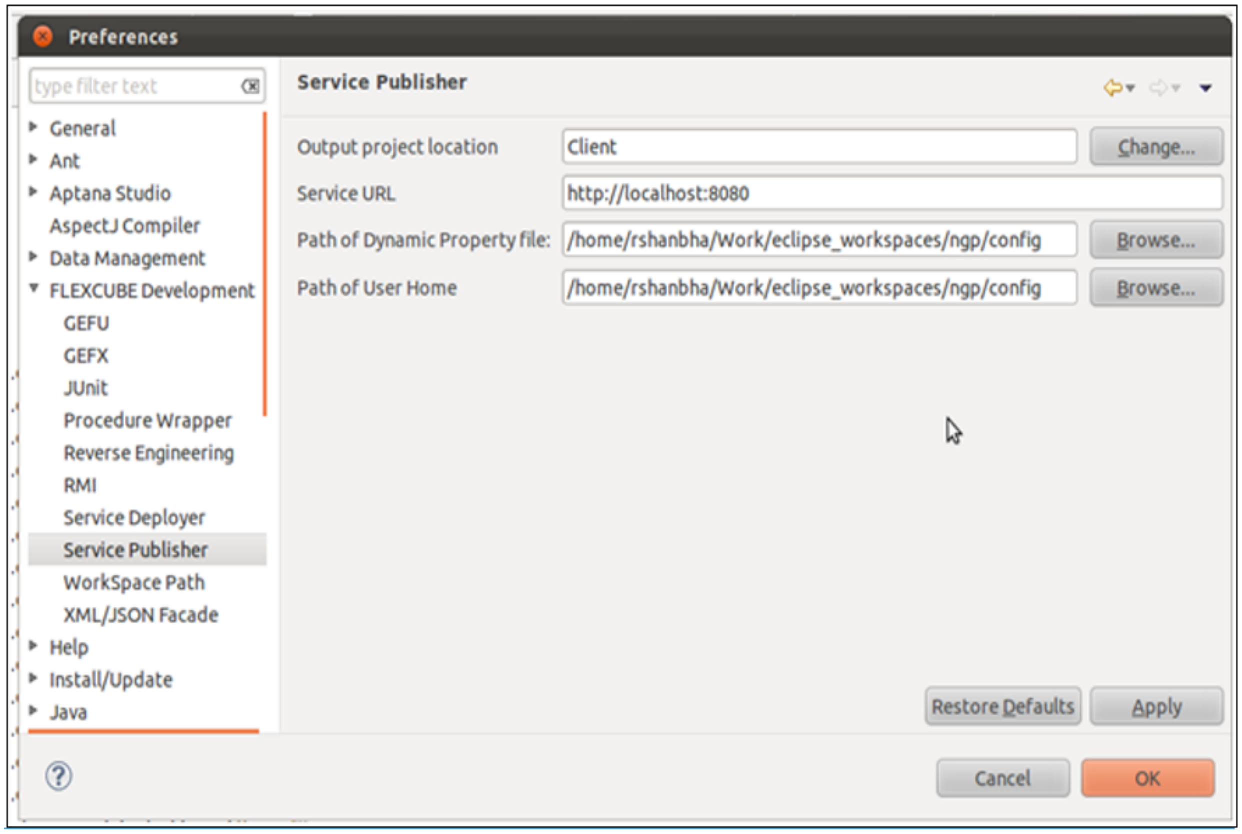
```
package com.ofss.fc.demo.domain.party.entity.contact;

import com.ofss.fc.datatype.Date;
import com.ofss.fc.framework.domain.AbstractDomainObject;
import com.ofss.fc.framework.domain.AbstractDomainObjectKey;

public class ContactExpiry extends AbstractDomainObject implements IContactExpiry {

    /**
     * Serial Version
     */
    private static final long serialVersionUID = -4179806027380497671L;

    /**
     * Contact Expiry Key
     */
    private ContactExpiryKey key;

    /**
     * Expiry Date
     */
    private Date expiryDate;

```

After creating the domain objects, build the project. You need to use the Flex cube development eclipse plug-in to generate the service layers.

Step 5 Set OBP Plugin Preferences

Before using the plug-in for generating service layer classes, you will need to set the required preferences for the plug-in. In eclipse, go to Windows → Preferences → OBP Development and the set the preferences as follows.

Figure 5–33 Set OBP Plugin Preferences



Service Publisher

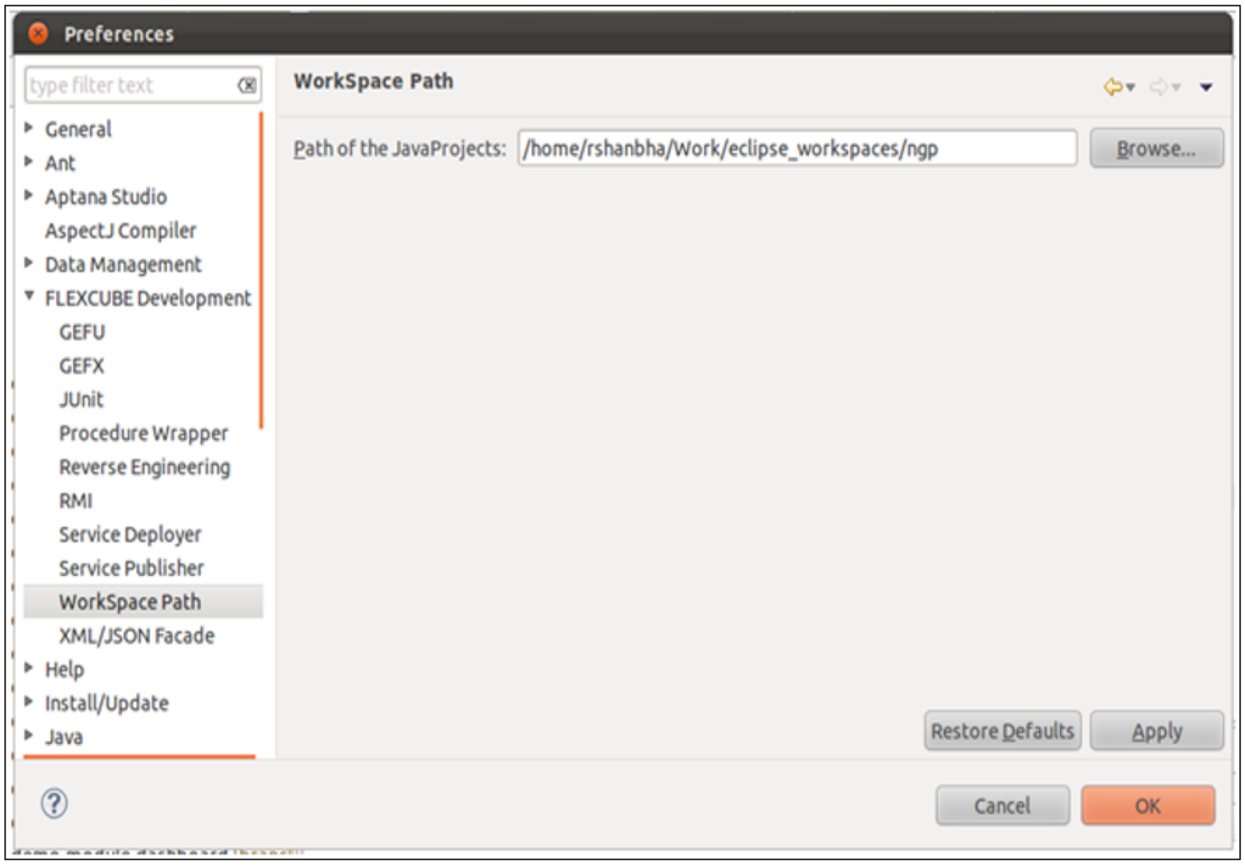
Output project location: Client

Service URL: http://localhost:8080

Path of Dynamic Property file: /home/rshanbha/Work/eclipse_workspaces/ngp/config

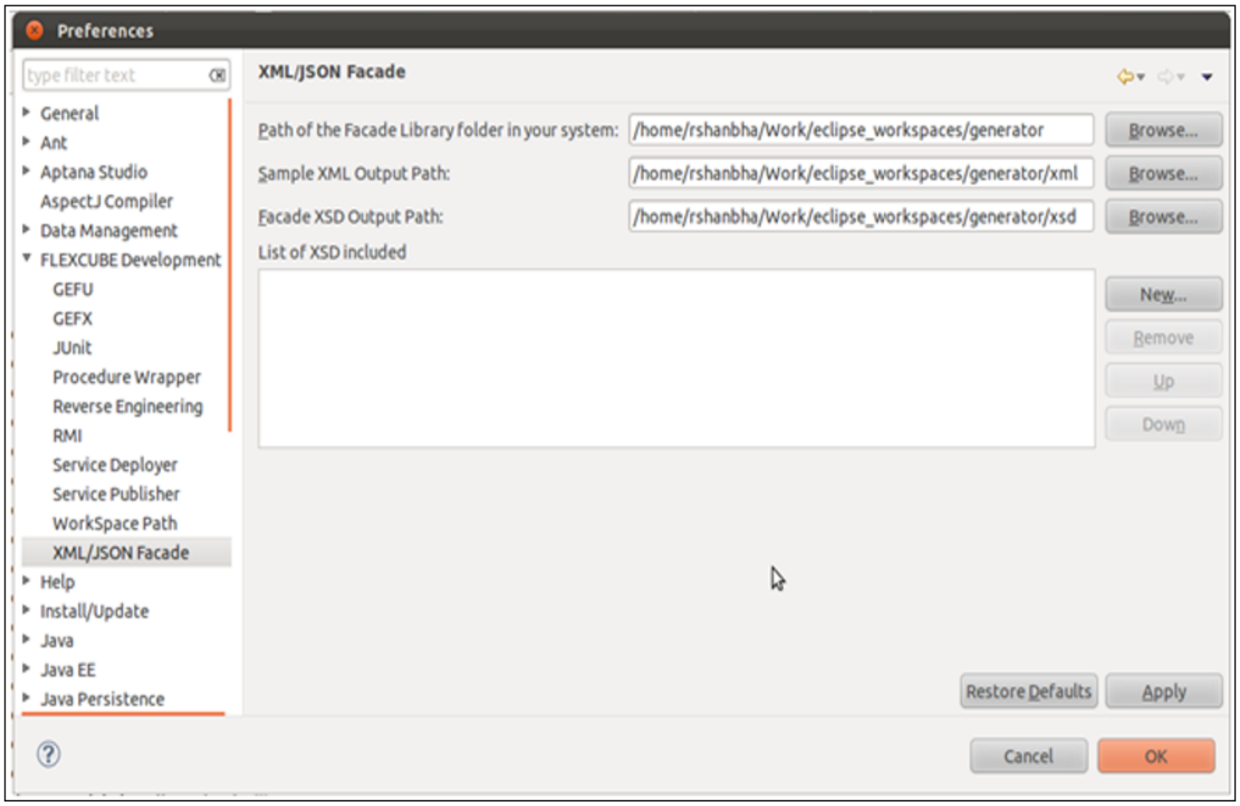
Path of User Home: /home/rshanbha/Work/eclipse_workspaces/ngp/config

Figure 5–34 Set OBP Plugin Preferences



Path of the JavaProjects: /home/rshanbha/Work/eclipse_workspaces/ngp

Figure 5–35 Set OBP Pugin Prefernces



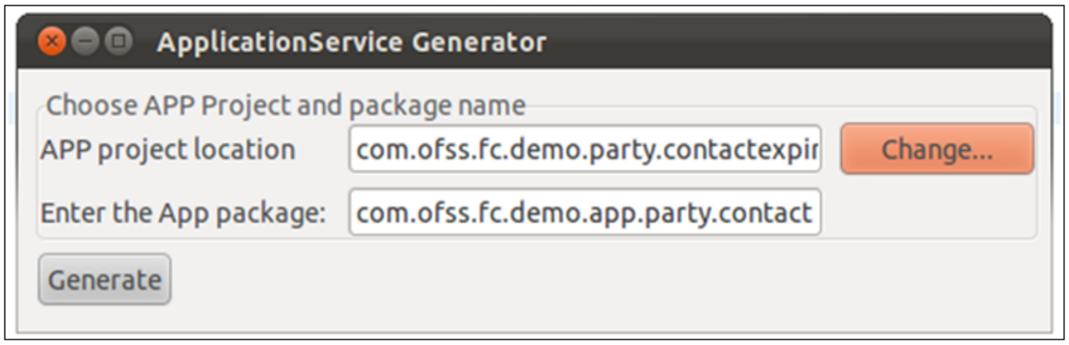
Path of the Facade Library folder in your system: /home/rshanbha/Work/eclipse_workspaces/generator Browse...
Sample XML Output Path: /home/rshanbha/Work/eclipse_workspaces/generator/xml Browse...
Facade XSD Output Path: /home/rshanbha/Work/eclipse_workspaces/generator/xsd Browse...
List of XSD included

Step 6 Create Application Service

You need to generate the application service layer classes using the OBP development plugin. Follow these steps:

1. Open the domain object class (ContactExpiry).
2. On the getter method of the Key object, add a javadoc comment @PK.
3. Right click on the editor window and from context menu that opens, choose OBP Development → Generate Application Service.
4. In the dialog that opens, select the Java project for generated classes. You can use the project previously created by you.

Figure 5–36 Create Application Service



Choose APP Project and package name

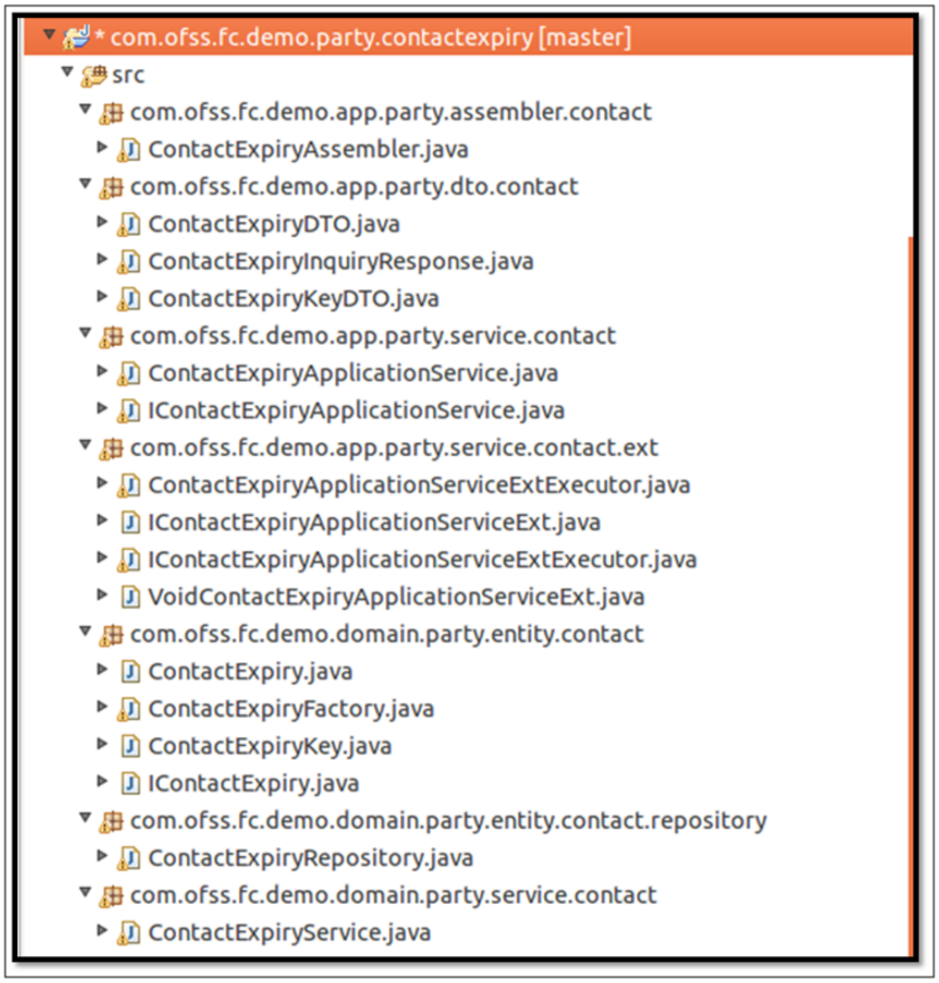
APP project location: com.ofss.fc.demo.party.contactexpir

Enter the App package: com.ofss.fc.demo.app.party.contact

5. Click on Generate. Application Service classes is generated in the project.

   The Java source might contain some compilation errors due to syntax. Fix these errors and build the project. The following classes should have been generated in the project.

Figure 5–37 Application Service Classes Generated



```
* com.ofss.fc.demo.party.contactexpiry [master]
  src
    com.ofss.fc.demo.app.party.assembler.contact
      ContactExpiryAssembler.java
    com.ofss.fc.demo.app.party.dto.contact
      ContactExpiryDTO.java
      ContactExpiryInquiryResponse.java
      ContactExpiryKeyDTO.java
    com.ofss.fc.demo.app.party.service.contact
      ContactExpiryApplicationService.java
      IContactExpiryApplicationService.java
    com.ofss.fc.demo.app.party.service.contact.ext
      ContactExpiryApplicationServiceExtExecutor.java
      IContactExpiryApplicationServiceExt.java
      IContactExpiryApplicationServiceExtExecutor.java
      VoidContactExpiryApplicationServiceExt.java
    com.ofss.fc.demo.domain.party.entity.contact
      ContactExpiry.java
      ContactExpiryFactory.java
      ContactExpiryKey.java
      IContactExpiry.java
    com.ofss.fc.demo.domain.party.entity.contact.repository
      ContactExpiryRepository.java
    com.ofss.fc.demo.domain.party.service.contact
      ContactExpiryService.java
```

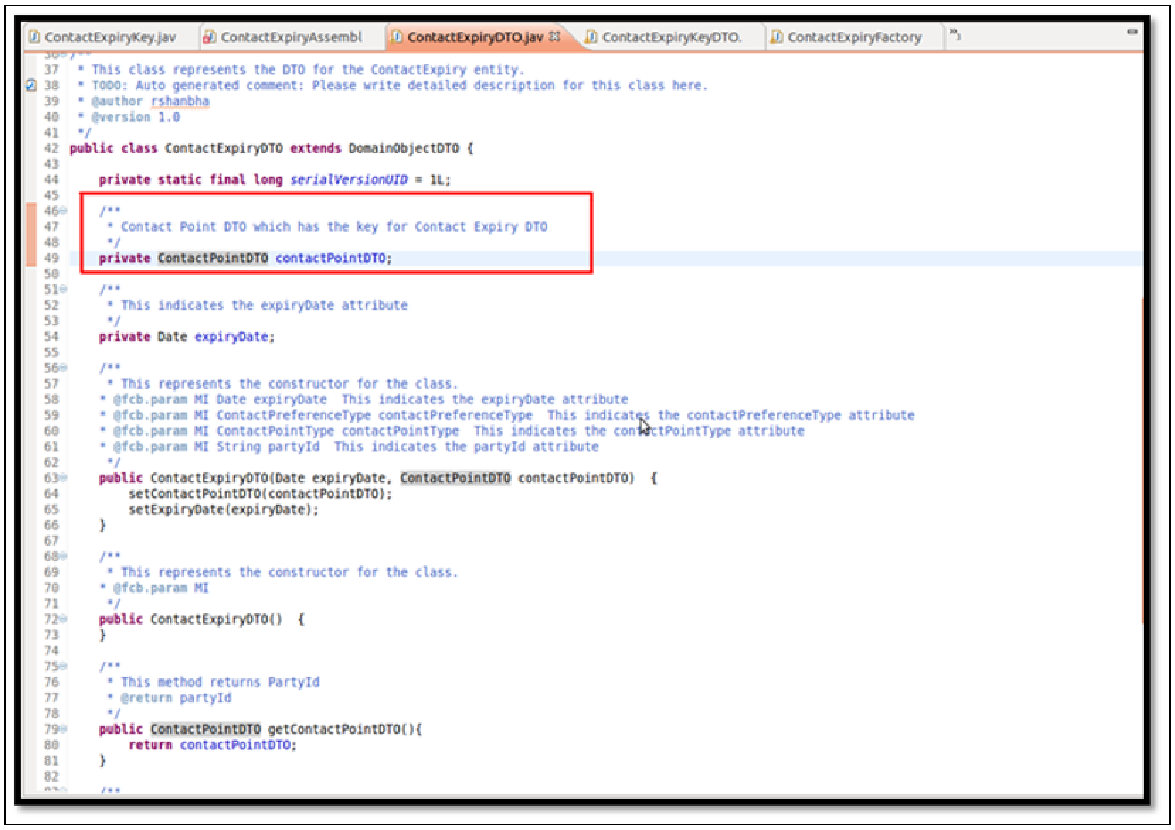
Step 7 Generate Service and Facade Layer Sources

Before generating the service and facade layer sources, you need to modify the Data Transfer Object (DTO). When a service call is made from the client application for a transaction related to Contact Point, the Contact Expiry transaction for the newly added Expiry Date field should be done in addition to the Contact Point transaction. Hence, the DTO for this transaction should also contain the DTO for the Contact Point transaction.

To modify the Data Transfer Object:

1. Open the ContactExpiryDTO class.
2. Delete the member ContactExpiryKey member and add ContactPoint member.
3. Re-factor references of the deleted member with the added member.

Figure 5–38 Modify Data Transfer Object (DTO)

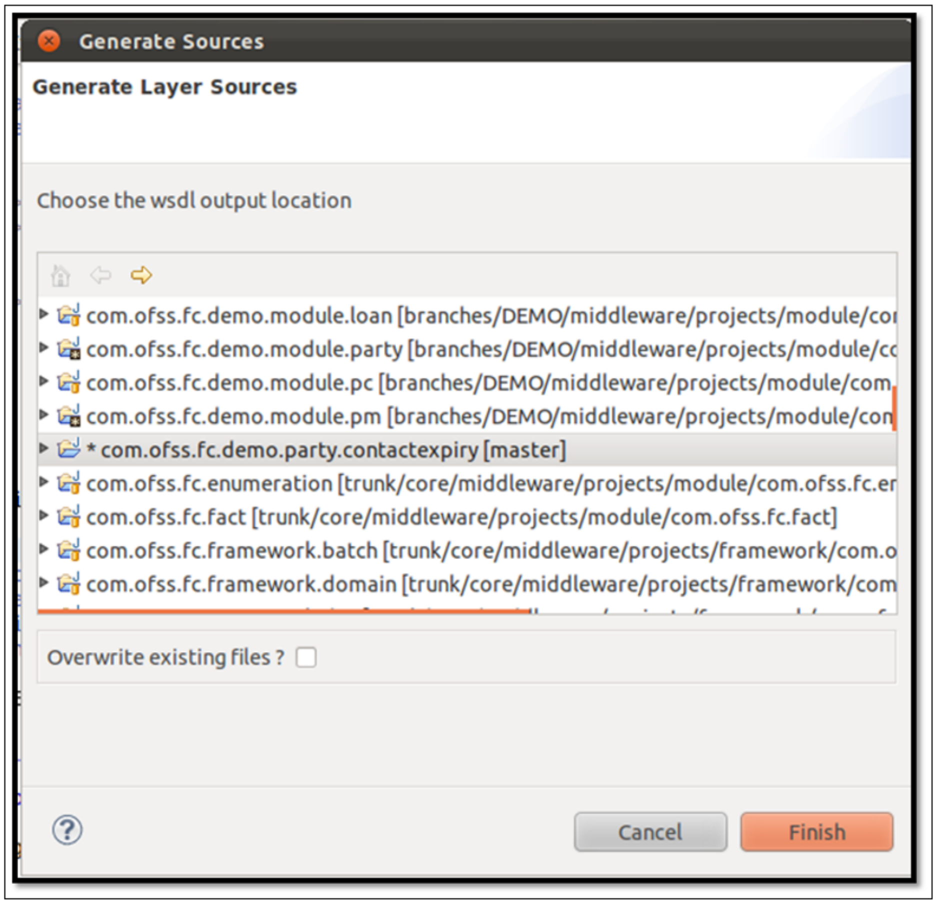


```java
 * This class represents the DTO for the ContactExpiry entity.
 * TODO: Auto generated comment: Please write detailed description for this class here.
 * @author rshanbha
 * @version 1.0
 */
public class ContactExpiryDTO extends DomainObjectDTO {

    private static final long serialVersionUID = 1L;

    /**
     * Contact Point DTO which has the key for Contact Expiry DTO
     */
    private ContactPointDTO contactPointDTO;

    /**
     * This indicates the expiryDate attribute
     */
    private Date expiryDate;

    /**
     * This represents the constructor for the class.
    * @fcb.param MI Date expiryDate  This indicates the expiryDate attribute
    * @fcb.param MI ContactPreferenceType contactPreferenceType  This indicates the contactPreferenceType attribute
    * @fcb.param MI ContactPointType contactPointType  This indicates the contactPointType attribute
    * @fcb.param MI String partyId  This indicates the partyId attribute
     */
    public ContactExpiryDTO(Date expiryDate, ContactPointDTO contactPointDTO)  {
        setContactPointDTO(contactPointDTO);
        setExpiryDate(expiryDate);
    }

    /**
     * This represents the constructor for the class.
    * @fcb.param MI
     */
    public ContactExpiryDTO()  {
    }

    /**
     * This method returns PartyId
     * @return partyId
     */
    public ContactPointDTO getContactPointDTO(){
        return contactPointDTO;
    }

    /**
```

To generate the service and facade layer sources:

1. Open the application service class (ContactExpiryApplicationService).
2. Right click on the editor window and from the context menu that opens, choose OBP Development → Generate Service and Facade Layer Sources.
3. In the dialog box that opens, select the Java project for the generated classes. You can use the project previously created by you. Un-check the Overwrite Existing Files option.

Figure 5–39 Generate Service and Facade Layer Sources



Choose the wsdl output location

- com.ofss.fc.demo.module.loan [branches/DEMO/middleware/projects/module/co
- com.ofss.fc.demo.module.party [branches/DEMO/middleware/projects/module/c
- com.ofss.fc.demo.module.pc [branches/DEMO/middleware/projects/module/com
- com.ofss.fc.demo.module.pm [branches/DEMO/middleware/projects/module/co
- * com.ofss.fc.demo.party.contactexpiry [master]
- com.ofss.fc.enumeration [trunk/core/middleware/projects/module/com.ofss.fc.e
- com.ofss.fc.fact [trunk/core/middleware/projects/module/com.ofss.fc.fact]
- com.ofss.fc.framework.batch [trunk/core/middleware/projects/framework/com.o
- com.ofss.fc.framework.domain [trunk/core/middleware/projects/framework/com

Overwrite existing files ? ☐

4. Click Finish.

   Service and facade layer sources is generated in the project.

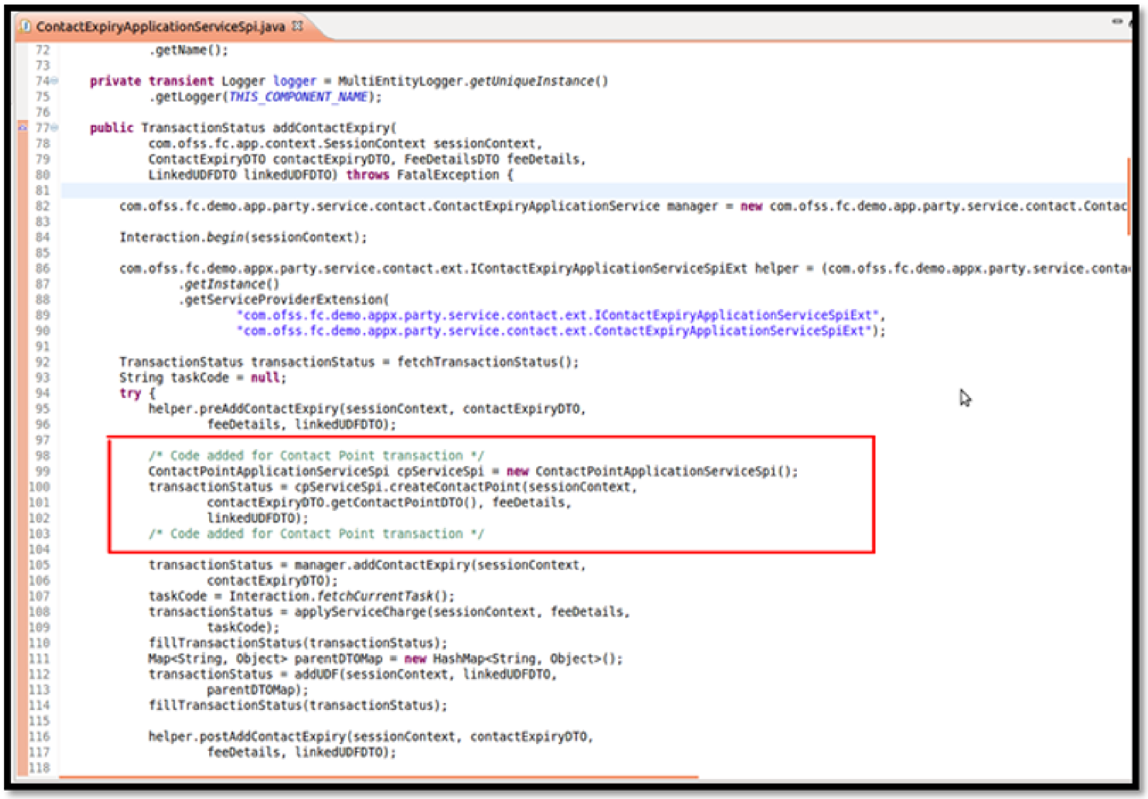
5. Certain classes might be generated twice. Delete the newly created copy of the classes and keep the original.

6. Certain compilation errors might be present in the generated classes due to erroneous syntax. Fix these compilation errors.

7. You will need to include a corresponding call to the Contact Point Application Service in the add, update and fetch transactions of the Contact Expiry Application Service.

8. Open ContactExpiryApplicationServiceSpi and modify the code as shown below.

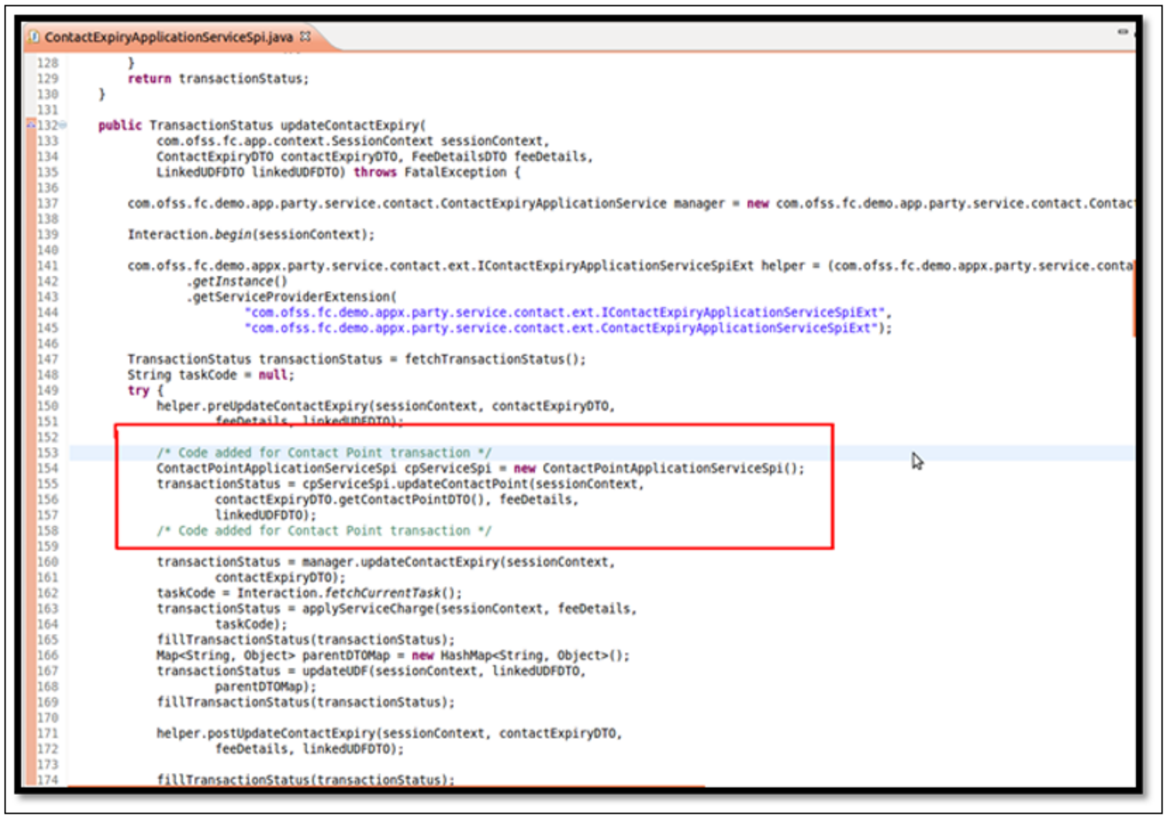
Figure 5–40 Modify ContactExpiryApplicationServiceSpi.java



ContactExpiryApplicationServiceSpi.java

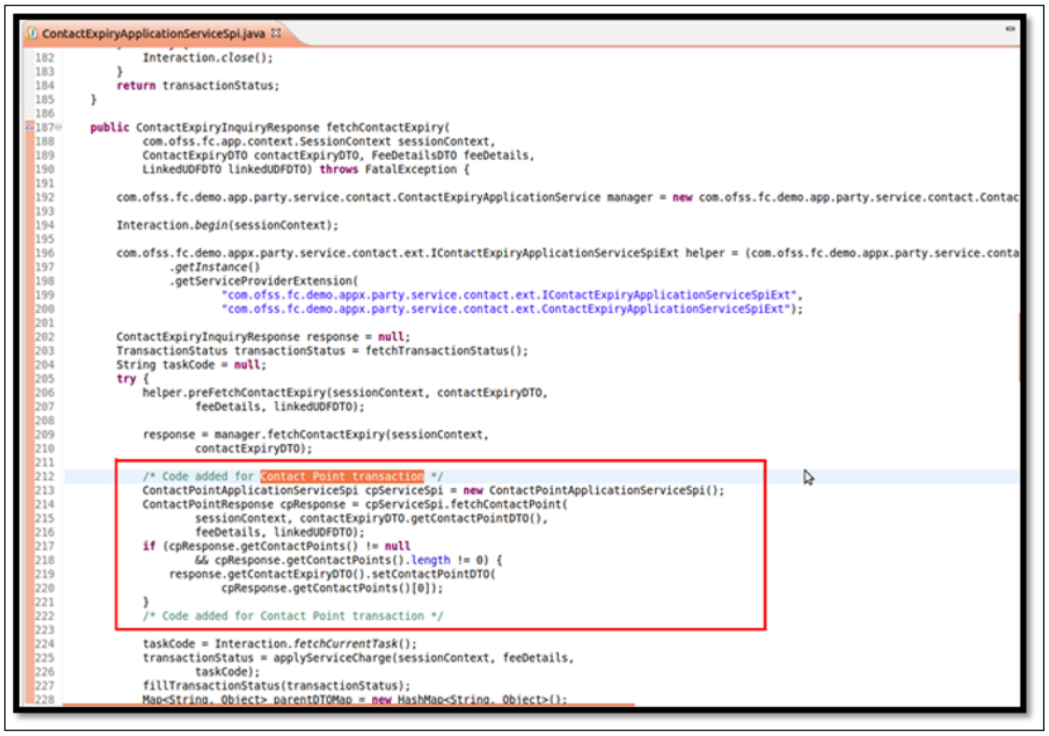
```java
                .getName();

    private transient Logger logger = MultiEntityLogger.getUniqueInstance()
                .getLogger(THIS_COMPONENT_NAME);

    public TransactionStatus addContactExpiry(
                com.ofss.fc.app.context.SessionContext sessionContext,
                ContactExpiryDTO contactExpiryDTO, FeeDetailsDTO feeDetails,
                LinkedUDFDTO linkedUDFDTO) throws FatalException {

        com.ofss.fc.demo.app.party.service.contact.ContactExpiryApplicationService manager = new com.ofss.fc.demo.app.party.service.contact.Contac

        Interaction.begin(sessionContext);

        com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt helper = (com.ofss.fc.demo.appx.party.service.conta
                .getInstance()
                .getServiceProviderExtension(
                        "com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt",
                        "com.ofss.fc.demo.appx.party.service.contact.ext.ContactExpiryApplicationServiceSpiExt");

        TransactionStatus transactionStatus = fetchTransactionStatus();
        String taskCode = null;
        try {
            helper.preAddContactExpiry(sessionContext, contactExpiryDTO,
                    feeDetails, linkedUDFDTO);

            /* Code added for Contact Point transaction */
            ContactPointApplicationServiceSpi cpServiceSpi = new ContactPointApplicationServiceSpi();
            transactionStatus = cpServiceSpi.createContactPoint(sessionContext,
                    contactExpiryDTO.getContactPointDTO(), feeDetails,
                    linkedUDFDTO);
            /* Code added for Contact Point transaction */

            transactionStatus = manager.addContactExpiry(sessionContext,
                    contactExpiryDTO);
            taskCode = Interaction.fetchCurrentTask();
            transactionStatus = applyServiceCharge(sessionContext, feeDetails,
                    taskCode);
            fillTransactionStatus(transactionStatus);
            Map<String, Object> parentDTOMap = new HashMap<String, Object>();
            transactionStatus = addUDF(sessionContext, linkedUDFDTO,
                    parentDTOMap);
            fillTransactionStatus(transactionStatus);

            helper.postAddContactExpiry(sessionContext, contactExpiryDTO,
                    feeDetails, linkedUDFDTO);

```

Figure 5–41 Modify ContactExpiryApplicationServiceSpi.java



```
        }
        return transactionStatus;
    }

    public TransactionStatus updateContactExpiry(
            com.ofss.fc.app.context.SessionContext sessionContext,
            ContactExpiryDTO contactExpiryDTO, FeeDetailsDTO feeDetails,
            LinkedUDFDTO linkedUDFDTO) throws FatalException {

        com.ofss.fc.demo.app.party.service.contact.ContactExpiryApplicationService manager = new com.ofss.fc.demo.app.party.service.contact.Contac

        Interaction.begin(sessionContext);

        com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt helper = (com.ofss.fc.demo.appx.party.service.conta
                .getInstance()
                .getServiceProviderExtension(
                        "com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt",
                        "com.ofss.fc.demo.appx.party.service.contact.ext.ContactExpiryApplicationServiceSpiExt");

        TransactionStatus transactionStatus = fetchTransactionStatus();
        String taskCode = null;
        try {
            helper.preUpdateContactExpiry(sessionContext, contactExpiryDTO,
                    feeDetails, linkedUDFDTO);

            /* Code added for Contact Point transaction */
            ContactPointApplicationServiceSpi cpServiceSpi = new ContactPointApplicationServiceSpi();
            transactionStatus = cpServiceSpi.updateContactPoint(sessionContext,
                    contactExpiryDTO.getContactPointDTO(), feeDetails,
                    linkedUDFDTO);
            /* Code added for Contact Point transaction */

            transactionStatus = manager.updateContactExpiry(sessionContext,
                    contactExpiryDTO);
            taskCode = Interaction.fetchCurrentTask();
            transactionStatus = applyServiceCharge(sessionContext, feeDetails,
                    taskCode);
            fillTransactionStatus(transactionStatus);
            Map<String, Object> parentDTOMap = new HashMap<String, Object>();
            transactionStatus = updateUDF(sessionContext, linkedUDFDTO,
                    parentDTOMap);
            fillTransactionStatus(transactionStatus);

            helper.postUpdateContactExpiry(sessionContext, contactExpiryDTO,
                    feeDetails, linkedUDFDTO);

            fillTransactionStatus(transactionStatus);
```

Figure 5–42 Modify ContactExpiryApplicationServiceSpi.java

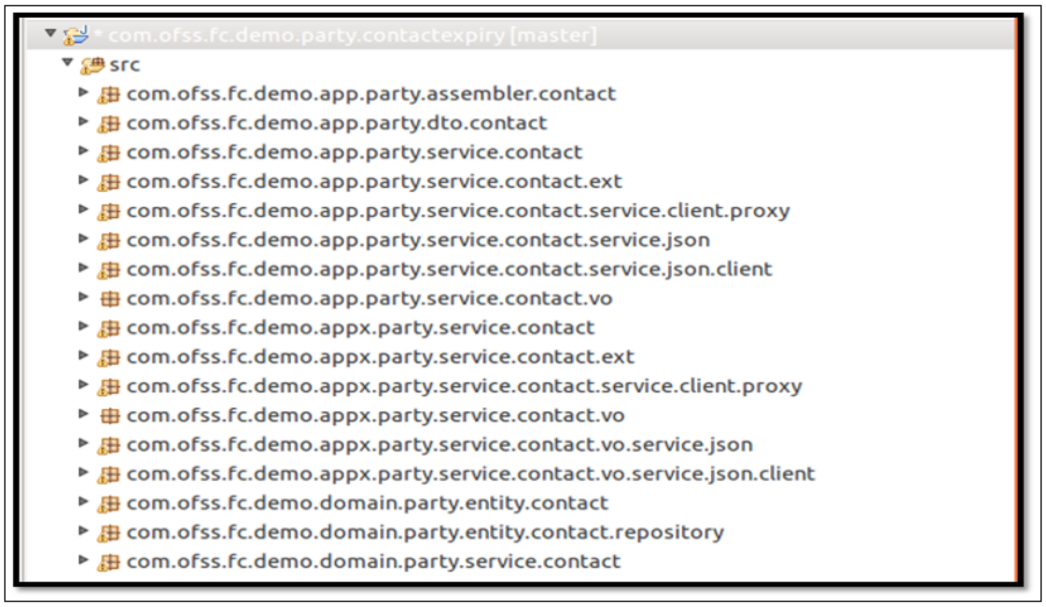


ContactExpiryApplicationServiceSpi.java

```
            Interaction.close();
        }
        return transactionStatus;
    }

    public ContactExpiryInquiryResponse fetchContactExpiry(
            com.ofss.fc.app.context.SessionContext sessionContext,
            ContactExpiryDTO contactExpiryDTO, FeeDetailsDTO feeDetails,
            LinkedUDFDTO linkedUDFDTO) throws FatalException {

        com.ofss.fc.demo.app.party.service.contact.ContactExpiryApplicationService manager = new com.ofss.fc.demo.app.party.service.contact.Contac

        Interaction.begin(sessionContext);

        com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt helper = (com.ofss.fc.demo.appx.party.service.conta
                .getInstance()
                .getServiceProviderExtension(
                        "com.ofss.fc.demo.appx.party.service.contact.ext.IContactExpiryApplicationServiceSpiExt",
                        "com.ofss.fc.demo.appx.party.service.contact.ext.ContactExpiryApplicationServiceSpiExt");

        ContactExpiryInquiryResponse response = null;
        TransactionStatus transactionStatus = fetchTransactionStatus();
        String taskCode = null;
        try {
            helper.preFetchContactExpiry(sessionContext, contactExpiryDTO,
                    feeDetails, linkedUDFDTO);

            response = manager.fetchContactExpiry(sessionContext,
                    contactExpiryDTO);

            /* Code added for Contact Point transaction */
            ContactPointApplicationServiceSpi cpServiceSpi = new ContactPointApplicationServiceSpi();
            ContactPointResponse cpResponse = cpServiceSpi.fetchContactPoint(
                    sessionContext, contactExpiryDTO.getContactPointDTO(),
                    feeDetails, linkedUDFDTO);
            if (cpResponse.getContactPoints() != null
                    && cpResponse.getContactPoints().length != 0) {
                response.getContactExpiryDTO().setContactPointDTO(
                        cpResponse.getContactPoints()[0]);
            }
            /* Code added for Contact Point transaction */

            taskCode = Interaction.fetchCurrentTask();
            transactionStatus = applyServiceCharge(sessionContext, feeDetails,
                    taskCode);
            fillTransactionStatus(transactionStatus);
            Map<String, Object> parentDTOMap = new HashMap<String, Object>();
```

9. The project should contain the Java packages as shown below:

Figure 5–43 Java Packages



```
▼ * com.ofss.fc.demo.party.contactexpiry [master]
  ▼ src
    ▶ com.ofss.fc.demo.app.party.assembler.contact
    ▶ com.ofss.fc.demo.app.party.dto.contact
    ▶ com.ofss.fc.demo.app.party.service.contact
    ▶ com.ofss.fc.demo.app.party.service.contact.ext
    ▶ com.ofss.fc.demo.app.party.service.contact.service.client.proxy
    ▶ com.ofss.fc.demo.app.party.service.contact.service.json
    ▶ com.ofss.fc.demo.app.party.service.contact.service.json.client
    ▶ com.ofss.fc.demo.app.party.service.contact.vo
    ▶ com.ofss.fc.demo.appx.party.service.contact
    ▶ com.ofss.fc.demo.appx.party.service.contact.ext
    ▶ com.ofss.fc.demo.appx.party.service.contact.service.client.proxy
    ▶ com.ofss.fc.demo.appx.party.service.contact.vo
    ▶ com.ofss.fc.demo.appx.party.service.contact.vo.service.json
    ▶ com.ofss.fc.demo.appx.party.service.contact.vo.service.json.client
    ▶ com.ofss.fc.demo.domain.party.entity.contact
    ▶ com.ofss.fc.demo.domain.party.entity.contact.repository
    ▶ com.ofss.fc.demo.domain.party.service.contact
```

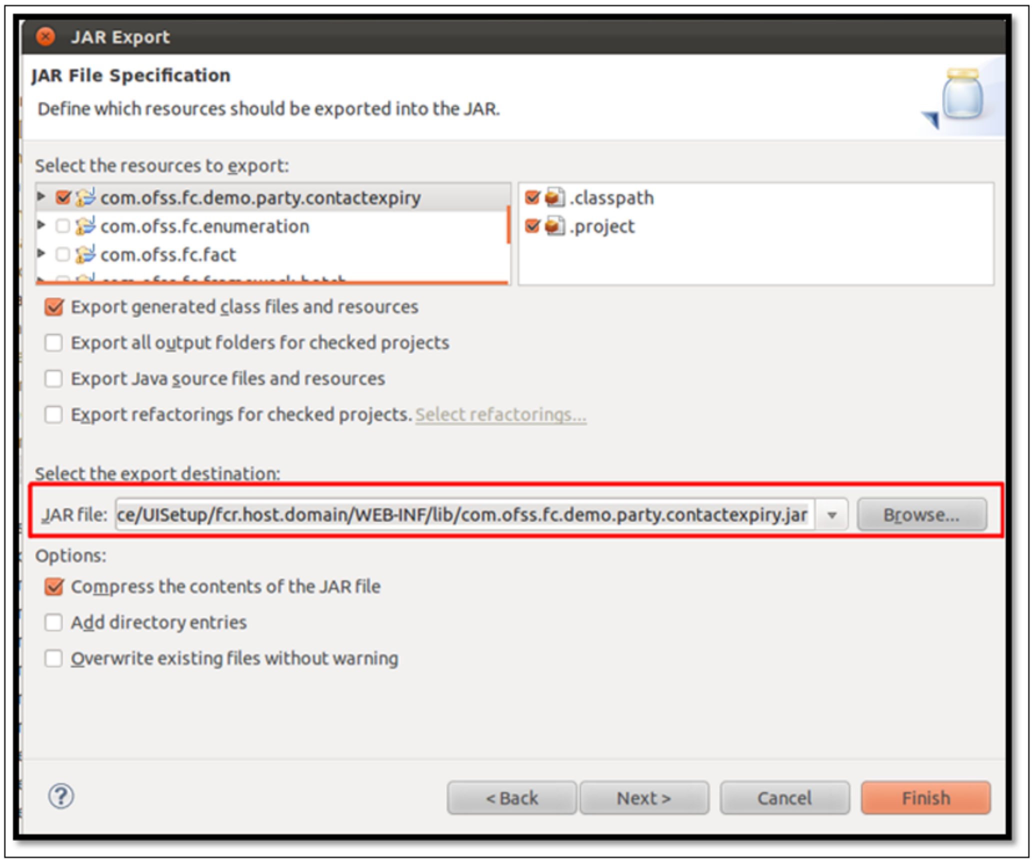
Step 8 Export Project as a JAR

You need to export the Java project containing the domain object, application service and facade layer source as a JAR.

To export java project as JAR:

1. Right click on the project and choose Export.
2. Choose JAR File in the export options.
3. Provide an export path and name (com.ofss.fc.demo.party.contactexpiry.jar) for the JAR file.
4. Click Finish.

Figure 5–44 Export Java Project as JAR


JAR Export

JAR File Specification

Define which resources should be exported into the JAR.

Select the resources to export:

- [✓] com.ofss.fc.demo.party.contactexpiry
- [ ] com.ofss.fc.enumeration
- [ ] com.ofss.fc.fact

- [✓] .classpath
- [✓] .project

- [✓] Export generated class files and resources
- [ ] Export all output folders for checked projects
- [ ] Export Java source files and resources
- [ ] Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: ce/UISetup/fcr.host.domain/WEB-INF/lib/com.ofss.fc.demo.party.contactexpiry.jar

Options:

- [✓] Compress the contents of the JAR file
- [ ] Add directory entries
- [ ] Overwrite existing files without warning

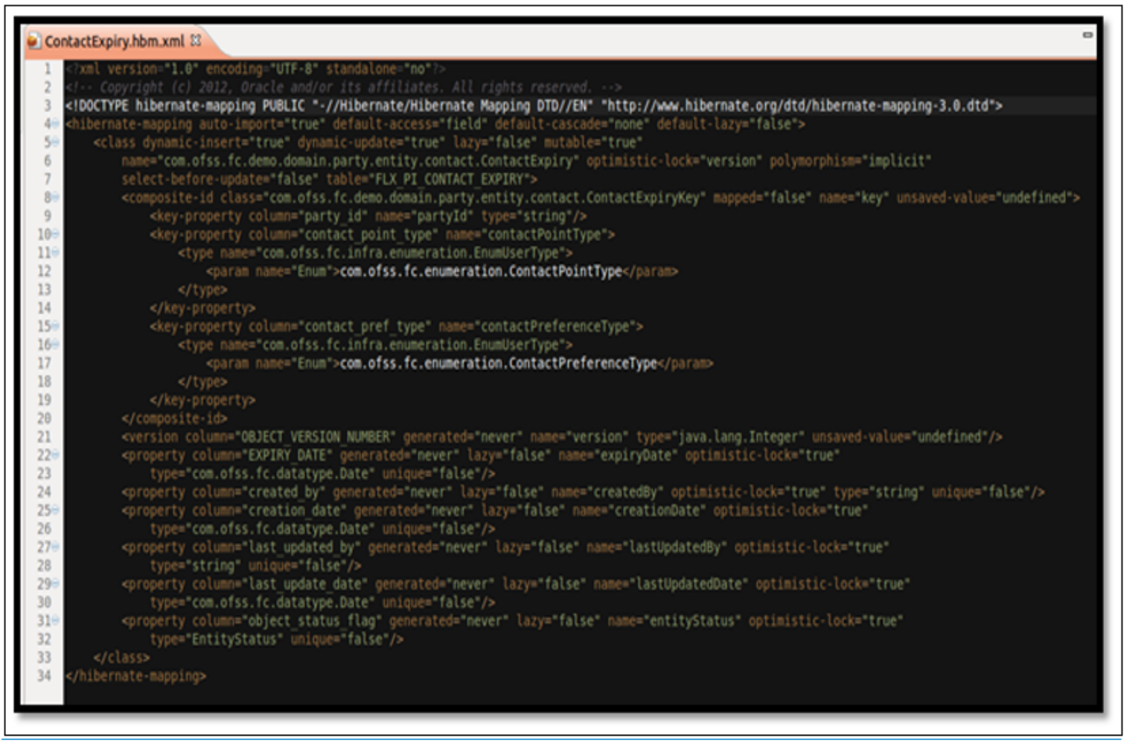
Step 9 Create Hibernate Mapping

You need to create a hibernate mapping to map the database table to the domain object.

Follow these steps:

1. Create ContactExpiry.hbm.xml file in the orm/hibernate/hbm folder of the config project of the host application.
2. Add the entry for this XML in the orm/hibernate/cfg/party-mapping.cfg.xml hibernate configuration XML.
3. Add the mapping in ContactExpiry.hbm.xml as shown below.

Figure 5–45 Create ContactExpiry.hbm.xml



ContactExpiry.hbm.xml

```xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!-- Copyright (c) 2012, Oracle and/or its affiliates. All rights reserved. -->
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping DTD//EN" "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping auto-import="true" default-access="field" default-cascade="none" default-lazy="false">
    <class dynamic-insert="true" dynamic-update="true" lazy="false" mutable="true"
        name="com.ofss.fc.demo.domain.party.entity.contact.ContactExpiry" optimistic-lock="version" polymorphism="implicit"
        select-before-update="false" table="FLX_PI_CONTACT_EXPIRY">
        <composite-id class="com.ofss.fc.demo.domain.party.entity.contact.ContactExpiryKey" mapped="false" name="key" unsaved-value="undefined">
            <key-property column="party_id" name="partyId" type="string"/>
            <key-property column="contact_point_type" name="contactPointType">
                <type name="com.ofss.fc.infra.enumeration.EnumUserType">
                    <param name="Enum">com.ofss.fc.enumeration.ContactPointType</param>
                </type>
            </key-property>
            <key-property column="contact_pref_type" name="contactPreferenceType">
                <type name="com.ofss.fc.infra.enumeration.EnumUserType">
                    <param name="Enum">com.ofss.fc.enumeration.ContactPreferenceType</param>
                </type>
            </key-property>
        </composite-id>
        <version column="OBJECT_VERSION_NUMBER" generated="never" name="version" type="java.lang.Integer" unsaved-value="undefined"/>
        <property column="EXPIRY_DATE" generated="never" lazy="false" name="expiryDate" optimistic-lock="true"
            type="com.ofss.fc.datatype.Date" unique="false"/>
        <property column="created_by" generated="never" lazy="false" name="createdBy" optimistic-lock="true" type="string" unique="false"/>
        <property column="creation_date" generated="never" lazy="false" name="creationDate" optimistic-lock="true"
            type="com.ofss.fc.datatype.Date" unique="false"/>
        <property column="last_updated_by" generated="never" lazy="false" name="lastUpdatedBy" optimistic-lock="true"
            type="string" unique="false"/>
        <property column="last_update_date" generated="never" lazy="false" name="lastUpdatedDate" optimistic-lock="true"
            type="com.ofss.fc.datatype.Date" unique="false"/>
        <property column="object_status_flag" generated="never" lazy="false" name="entityStatus" optimistic-lock="true"
            type="EntityStatus" unique="false"/>
    </class>
</hibernate-mapping>
```

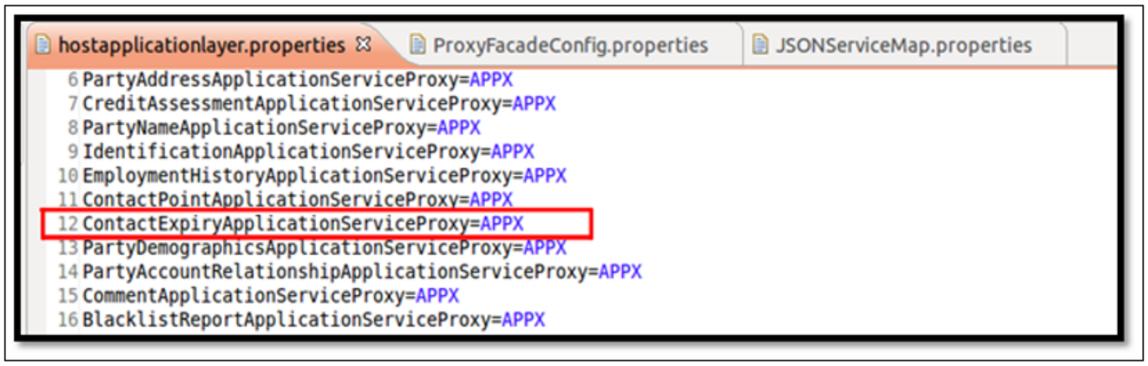
Step 10 Configure Host Application Project

You need to configure the Contact Expiry Application Service and Facade Layer in the host application.

To configure, follow these steps:

1. Configure APPX layer as the service layer for Contact Expiry service.
2. Open properties/hostapplicationlayer.properties present in the configuration project and add an entry as shown below.

Figure 5–46 Configure hostapplicationlayer.properties



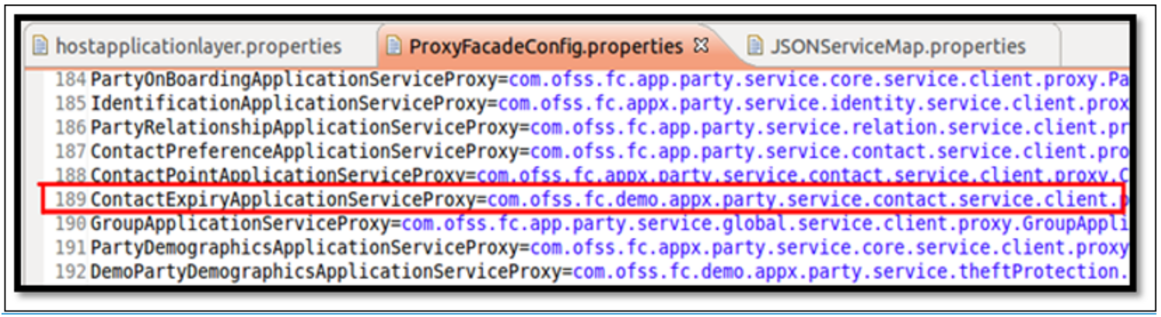
hostapplicationlayer.properties
ProxyFacadeConfig.properties
JSONServiceMap.properties

```
PartyAddressApplicationServiceProxy=APPX
CreditAssessmentApplicationServiceProxy=APPX
PartyNameApplicationServiceProxy=APPX
IdentificationApplicationServiceProxy=APPX
EmploymentHistoryApplicationServiceProxy=APPX
ContactPointApplicationServiceProxy=APPX
ContactExpiryApplicationServiceProxy=APPX
PartyDemographicsApplicationServiceProxy=APPX
PartyAccountRelationshipApplicationServiceProxy=APPX
CommentApplicationServiceProxy=APPX
BlacklistReportApplicationServiceProxy=APPX
```

3. Configure APPX layer proxy as the proxy for Contact Expiry service.
4. Open properties/ProxyFacadeConfig.properties present in the configuration project and add an entry as

shown below.

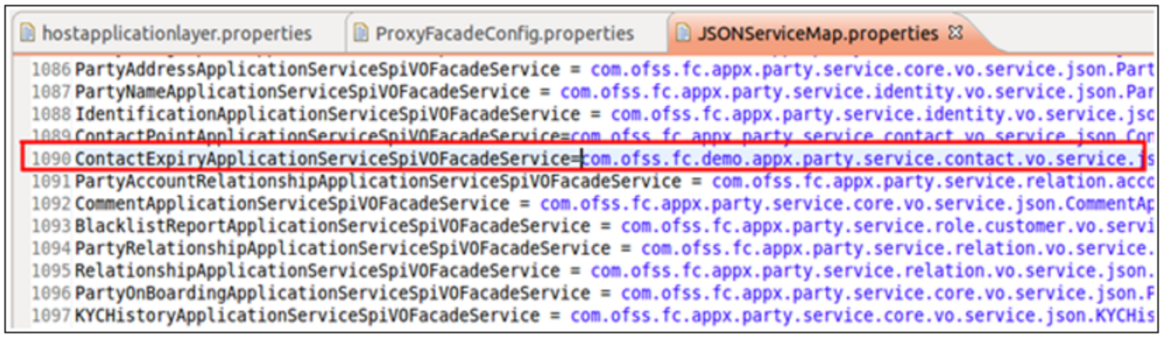
Figure 5–47 Configure ProxyFacadeConfig.properties


hostapplicationlayer.properties
ProxyFacadeConfig.properties
JSONServiceMap.properties

```
PartyOnBoardingApplicationServiceProxy=com.ofss.fc.app.party.service.core.service.client.proxy.Pa
IdentificationApplicationServiceProxy=com.ofss.fc.appx.party.service.identity.service.client.prox
PartyRelationshipApplicationServiceProxy=com.ofss.fc.app.party.service.relation.service.client.pr
ContactPreferenceApplicationServiceProxy=com.ofss.fc.app.party.service.contact.service.client.pro
ContactPointApplicationServiceProxy=com.ofss.fc.appx.party.service.contact.service.client.proxy.C
ContactExpiryApplicationServiceProxy=com.ofss.fc.demo.appx.party.service.contact.service.client.
GroupApplicationServiceProxy=com.ofss.fc.app.party.service.global.service.client.proxy.GroupAppli
PartyDemographicsApplicationServiceProxy=com.ofss.fc.appx.party.service.core.service.client.proxy
DemoPartyDemographicsApplicationServiceProxy=com.ofss.fc.demo.appx.party.service.theftProtection.
```

5. Configure the JSON and Facade layer mapping for Contact Expiry service.
6. Open properties/JSONServiceMap.properties present in the configuration project and add the two entries as shown below.

Figure 5–48 Configure JSONServiceMap.properties


hostapplicationlayer.properties
ProxyFacadeConfig.properties
JSONServiceMap.properties

```
PartyAddressApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.core.vo.service.json.Part
PartyNameApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.identity.vo.service.json.Par
IdentificationApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.identity.vo.service.jsc
ContactPointApplicationServiceSpiVOFacadeService=com.ofss.fc.appx.party.service.contact.vo.service.json.Cor
ContactExpiryApplicationServiceSpiVOFacadeService=com.ofss.fc.demo.appx.party.service.contact.vo.service.js
PartyAccountRelationshipApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.relation.accc
CommentApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.core.vo.service.json.CommentAp
BlacklistReportApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.role.customer.vo.servi
PartyRelationshipApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.relation.vo.service.
RelationshipApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.relation.vo.service.json.
PartyOnBoardingApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.core.vo.service.json.F
KYCHistoryApplicationServiceSpiVOFacadeService = com.ofss.fc.appx.party.service.core.vo.service.json.KYCHis
```

Step 11 Deploy Project

After performing all the above mentioned changes, deploy the project as follows:

1. Add this project (com.ofss.fc.demo.party.contactexpiry) to the classpath of the branch application project.
2. Open the launch configuration of the Tomcat Server. Add this project to the classpath of the server as well.
3. Deploy the branch application project on the server and start it.
4. Client Application Changes.

After creating database table to hold the input data and after creating the related domain objects and service and facade layers, you can customize the user interface. The customizations to the application have to be done on the client application. To customize the UI, follow these steps.

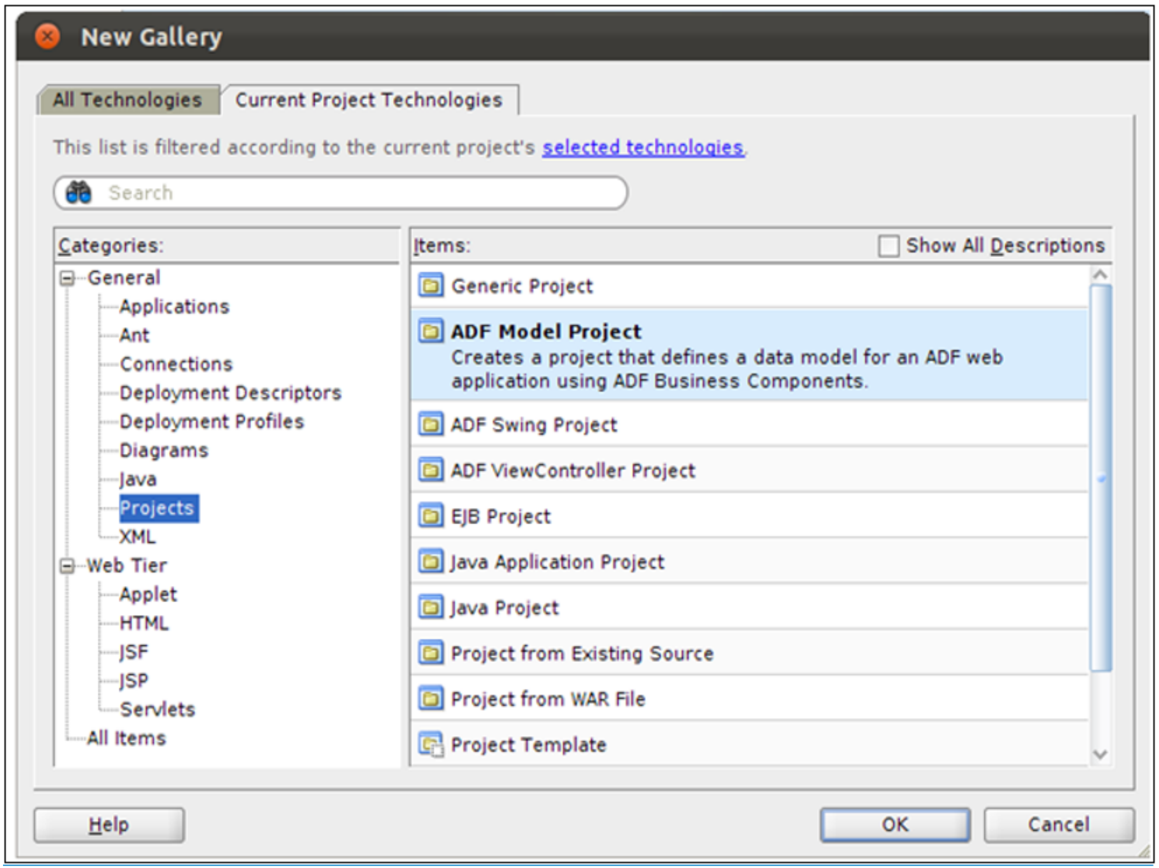
Step 12 Create Model Project

You need to create a model project to hold the required view objects and application module.

To create the model project, follow these steps:

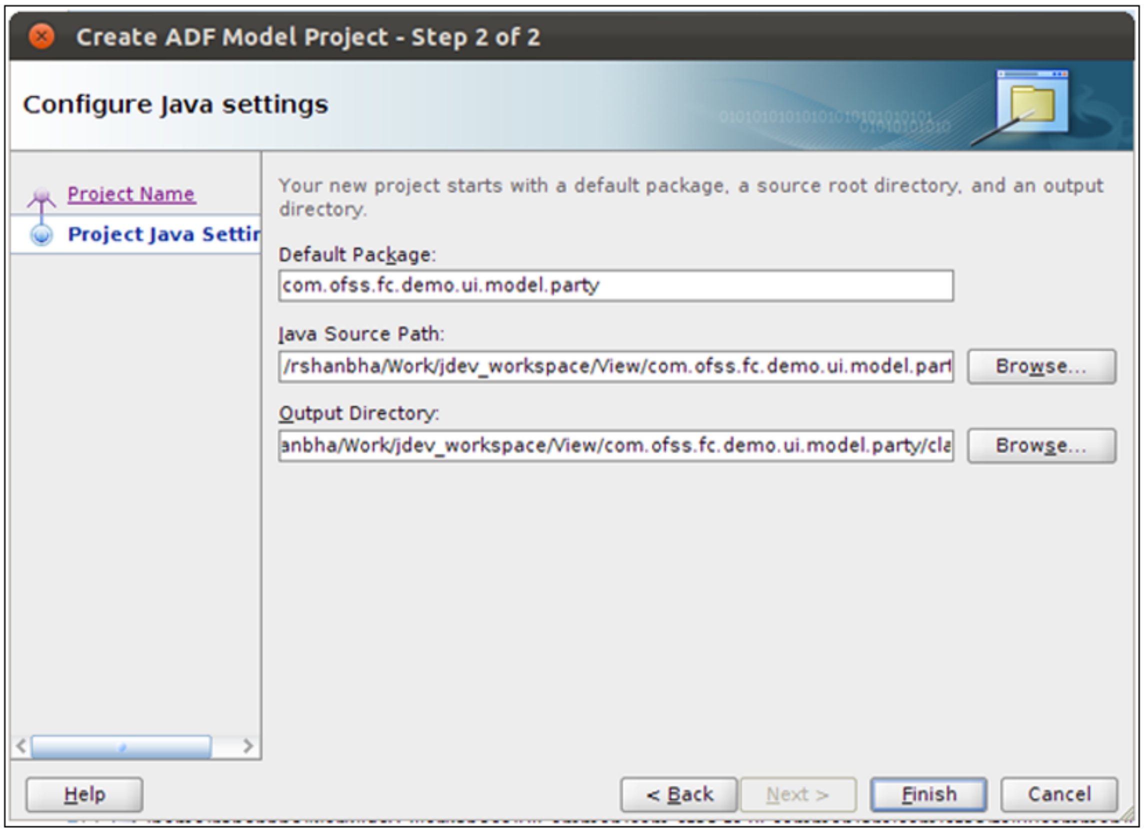
1. In the client application, create a new project of the type ADF Model Project.

Figure 5–49 Create Model Project



2. Give the project a title (com.ofss.fc.demo.ui.model.party) and set the default package as the same.

Figure 5–50 Create Model Project - Configure Java Settings



Create ADF Model Project - Step 2 of 2

Configure Java settings

Your new project starts with a default package, a source root directory, and an output directory.

Default Package:
com.ofss.fc.demo.ui.model.party

Java Source Path:
/rshanbha/Work/jdev_workspace/View/com.ofss.fc.demo.ui.model.part

Output Directory:
anbha/Work/jdev_workspace/View/com.ofss.fc.demo.ui.model.party/cla

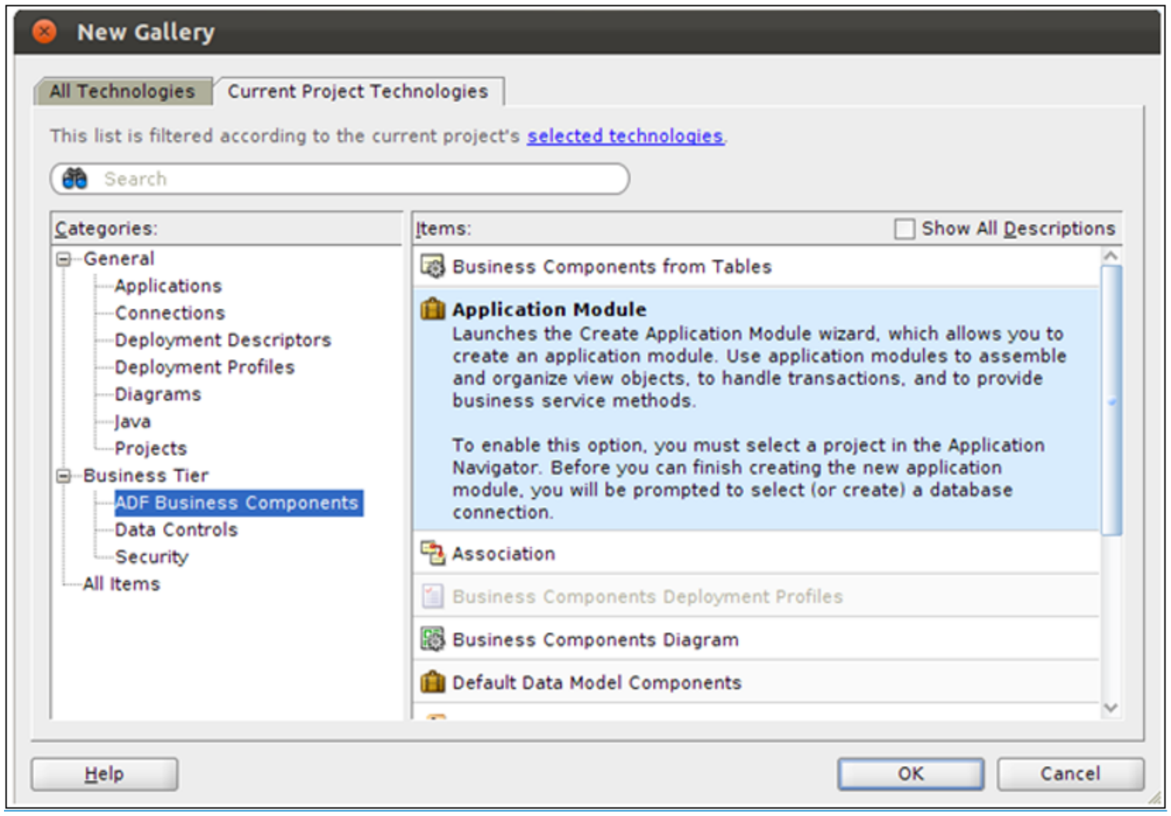
3. Click on Finish to create the project.

Step 13 Create Application Module

You need to create an application module to contain the information of all the view objects that you need to create. To create an application module, follow these steps:

1. Right click on the model project and select New.
2. Choose Application Module from the dialog box that opens.

Figure 5–51 Create Application Module



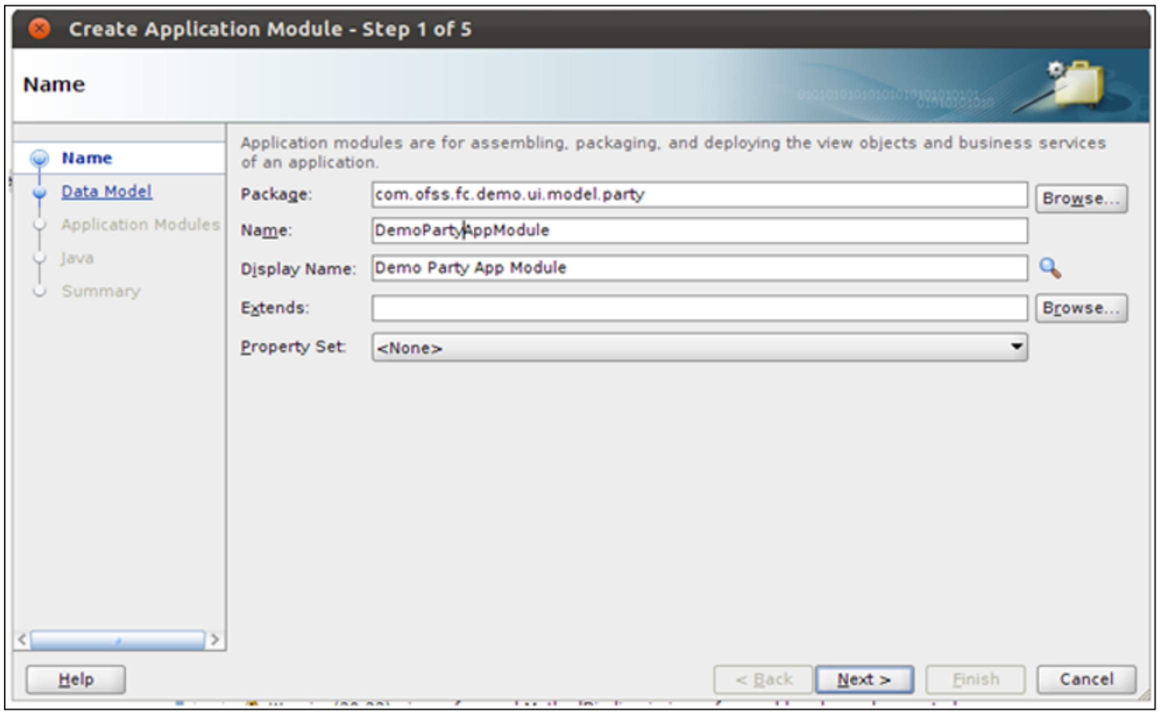
This list is filtered according to the current project's selected technologies.

Launches the Create Application Module wizard, which allows you to create an application module. Use application modules to assemble and organize view objects, to handle transactions, and to provide business service methods.

To enable this option, you must select a project in the Application Navigator. Before you can finish creating the new application module, you will be prompted to select (or create) a database connection.

3. Set the package of the application module to the default package (com.ofss.fc.demo.ui.model.party).
4. Provide a name to the application module (DemoPartyAppModule).

Figure 5–52 Set Package and Name of Application Module



Application modules are for assembling, packaging, and deploying the view objects and business services of an application.

Package: com.ofss.fc.demo.ui.model.party

Name: DemoPartyAppModule

Display Name: Demo Party App Module

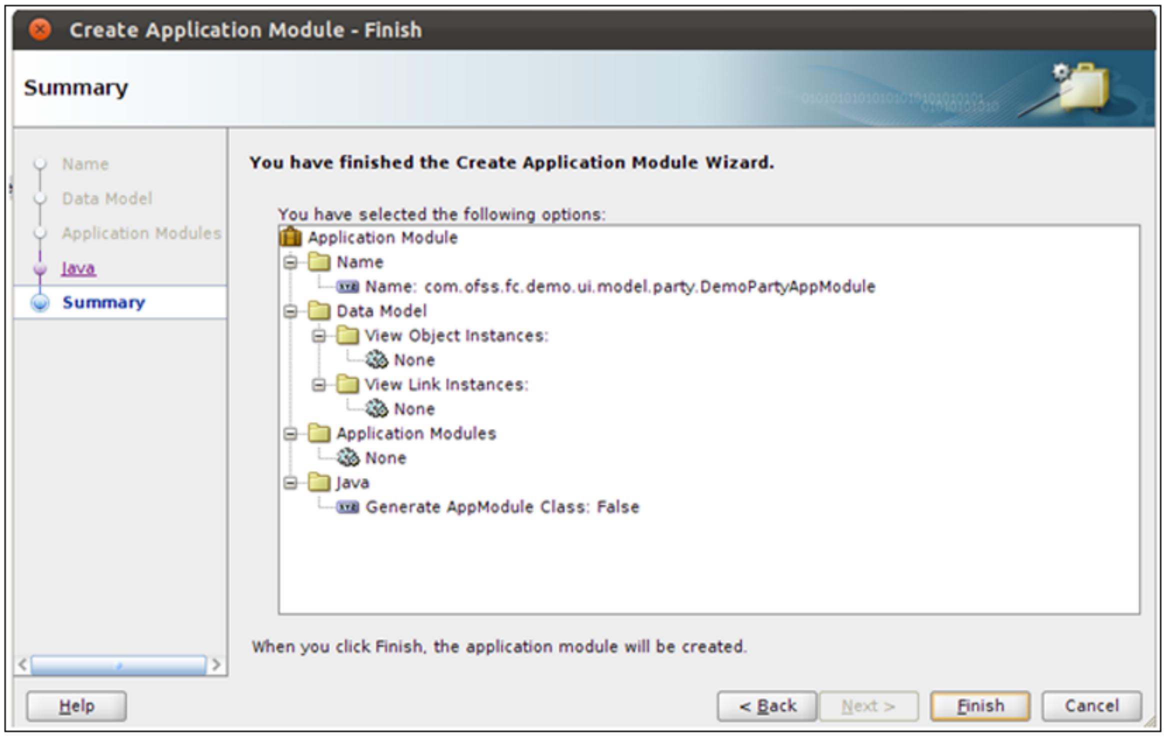
Extends:

Property Set: <None>

5. Click on Next and let the rest of the options be set to the default options.

6. You will see a summary screen for the application module. Click on Finish to create the application module.

Figure 5–53 Summary of Application Module Created



You have finished the Create Application Module Wizard.

You have selected the following options:

```
Application Module
  Name
    Name: com.ofss.fc.demo.ui.model.party.DemoPartyAppModule
  Data Model
    View Object Instances:
      None
    View Link Instances:
      None
  Application Modules
    None
  Java
    Generate AppModule Class: False
```

When you click Finish, the application module will be created.

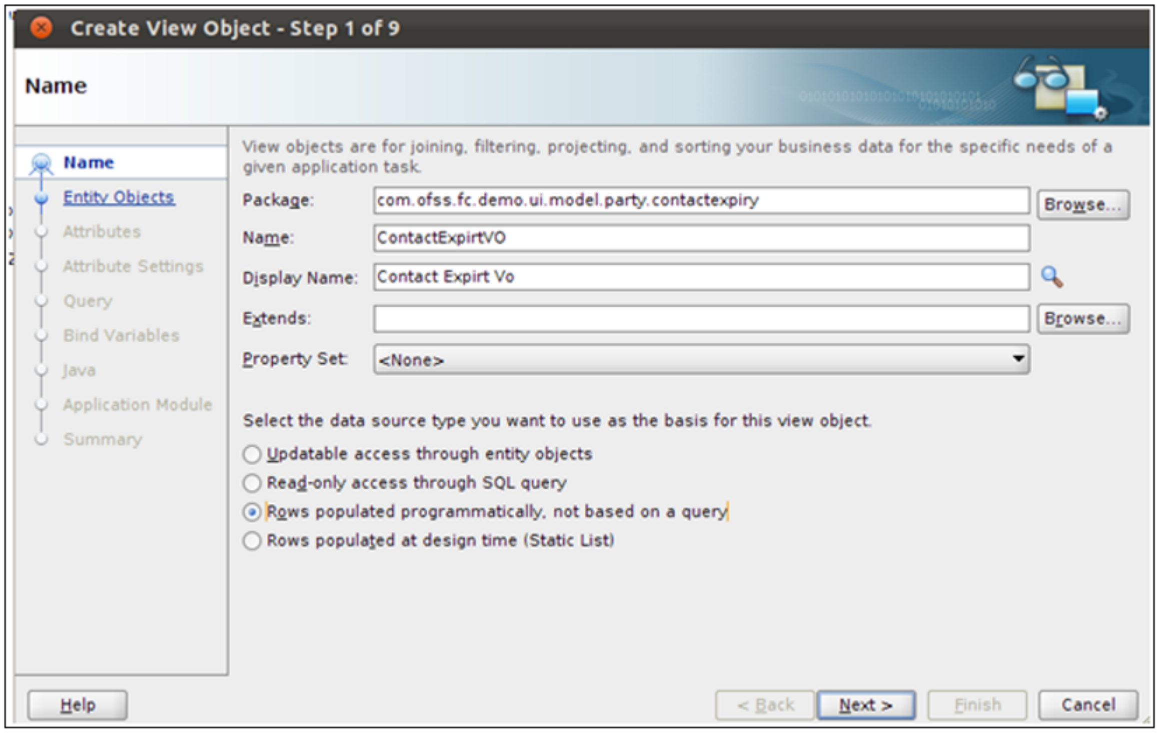
Step 14 Create View Object

You need to create a view object for the newly added Expiry Date field. This view object is used on the screen to display the value of the field as well as to take the input for the field.

To create the view object, follow these steps:

1. Right click on the Java package com.ofss.fc.demo.ui.model.party and select New View Object.
2. In the dialog box that opens, provide a name (ContactExpiryVO) for the view object.
3. Provide a package (com.ofss.fc.demo.ui.model.party.contactexpiry) for the view object.
4. For the Data Source Type option, select Rows populated programmatically, not based on a query.
5. Click on Next.
6. In the Attributes dialog, create a new attribute for Expiry Date field.
7. Provide a name (ExpiryDate) and type (Date) for the attribute.
8. For the Updatable option, select Always.

Figure 5–54 Create View Object



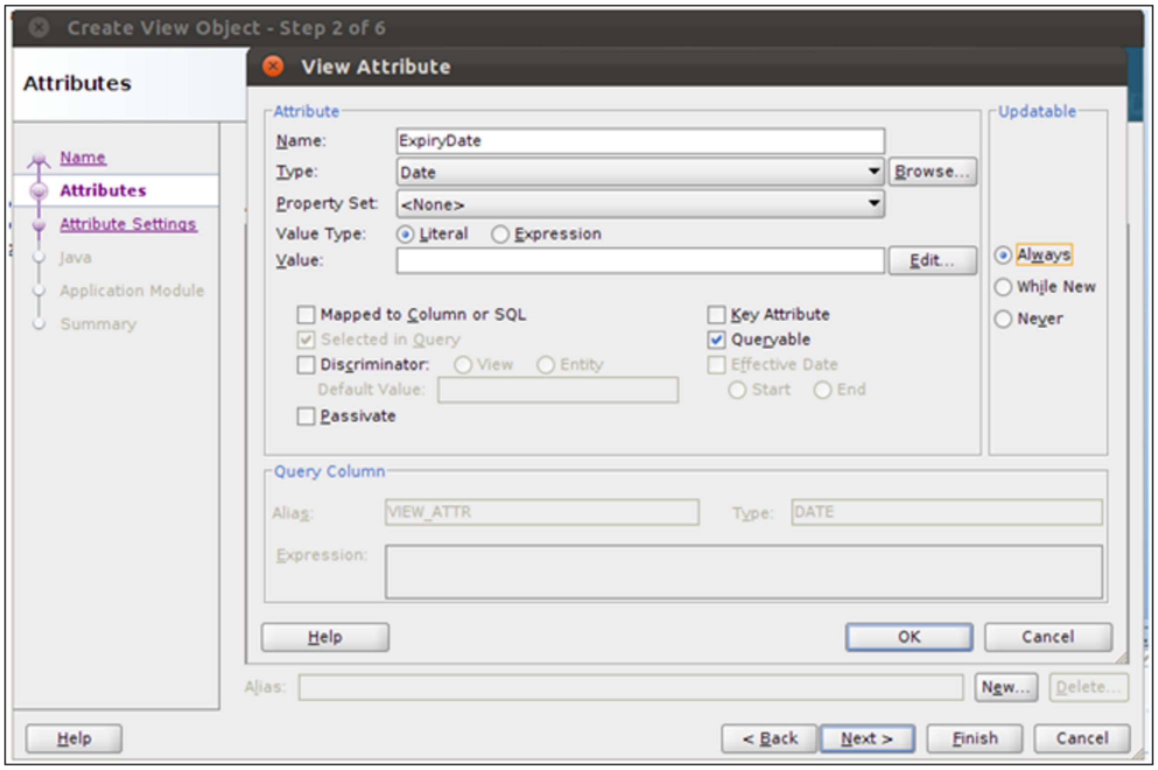
View objects are for joining, filtering, projecting, and sorting your business data for the specific needs of a given application task.

Package:	com.ofss.fc.demo.ui.model.party.contactexpiry
Name:	ContactExpirtVO
Display Name:	Contact Expirt Vo
Extends:	
Property Set:	<None>

Select the data source type you want to use as the basis for this view object.

- Updatable access through entity objects
- Read-only access through SQL query
- [✓ Rows populated programmatically, not based on a query]
- Rows populated at design time (Static List)

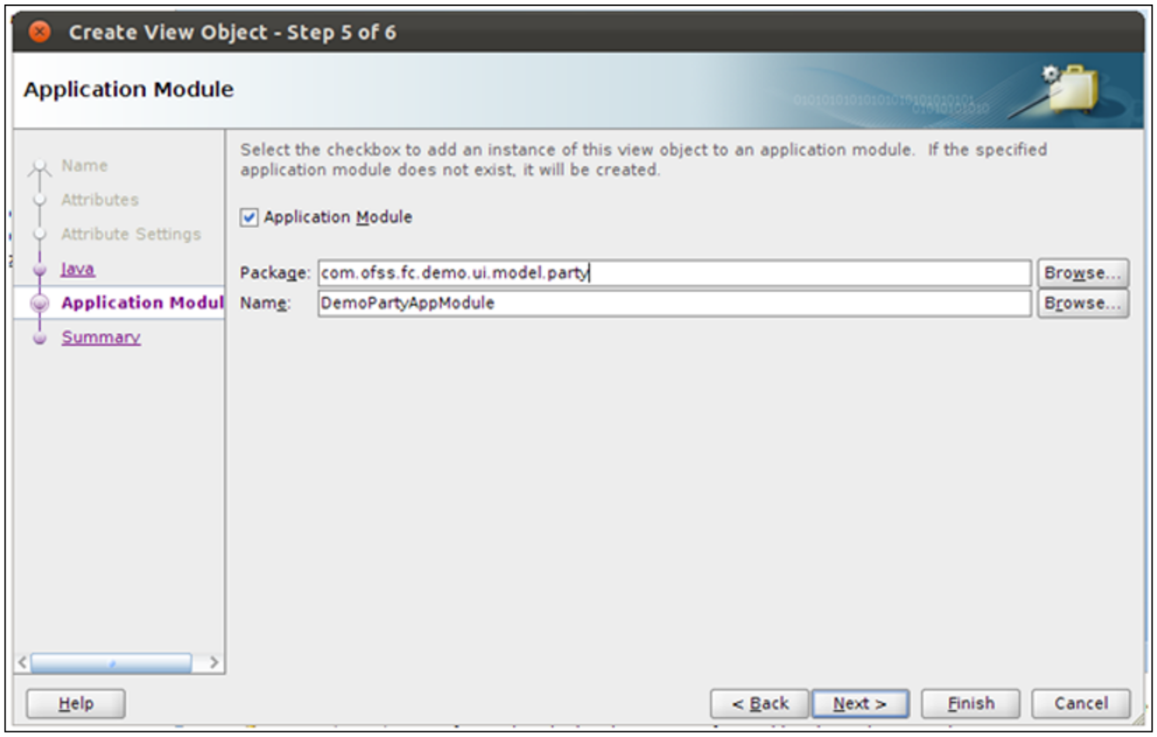
Figure 5–55 View Attribute



Name: ExpiryDate
Type: Date
Property Set: <None>
Value Type: [✓ Literal] Expression
Value:
Updatable: [✓ Always] While New Never
Mapped to Column or SQL ☐
Selected in Query ✓
Discriminator: ☐ View Entity
Default Value:
Passivate ☐
Key Attribute ☐
Queryable ✓
Effective Date ☐ Start End
Alias: VIEW_ATTR
Type: DATE
Expression:

9. Click Next. On the Application Module dialog, browse for the previously created DemoPartyAppModule.

Figure 5–56 Application Module



Select the checkbox to add an instance of this view object to an application module. If the specified application module does not exist, it will be created.

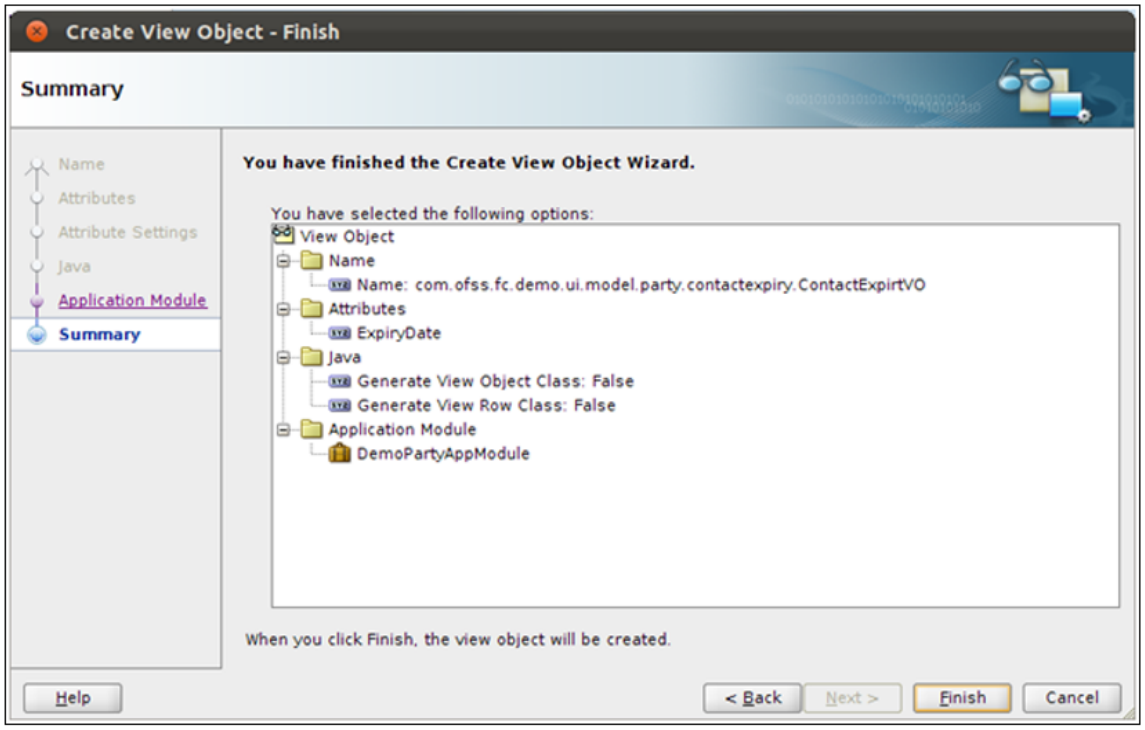
[✓] Application Module

Package: com.ofss.fc.demo.ui.model.party

Name: DemoPartyAppModule

10. For all other dialogs, keep the default options.
11. Click Next till you reach the summary screen as shown below.
12. Click on Finish to create the view object.

Figure 5–57 Create View Object - Summary



You have finished the Create View Object Wizard.

You have selected the following options:

```
View Object
  Name
    Name: com.ofss.fc.demo.ui.model.party.contactexpiry.ContactExpirtVO
  Attributes
    ExpiryDate
  Java
    Generate View Object Class: False
    Generate View Row Class: False
  Application Module
    DemoPartyAppModule
```

When you click Finish, the view object will be created.

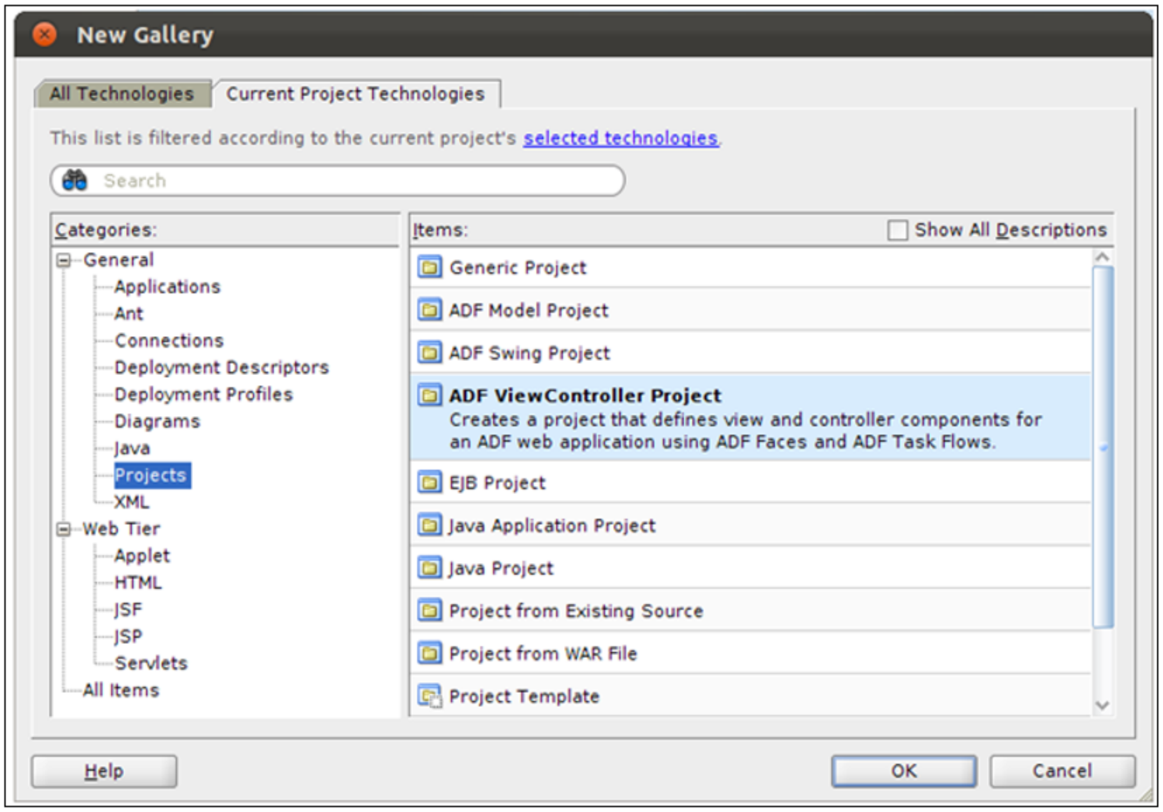
Step 15 Create View Controller Project

You need to create a view controller project to contain the UI elements. This project will also hold the customizations to the application.

To create the view controller project, follow these steps:

1. In the client application, create new project of the type ADF View Controller Project.
2. Give the project a title (com.ofss.fc.demo.ui.view.party) and set the defaults package to the same.

Figure 5–58 Create View Controller Project



This list is filtered according to the current project's selected technologies.

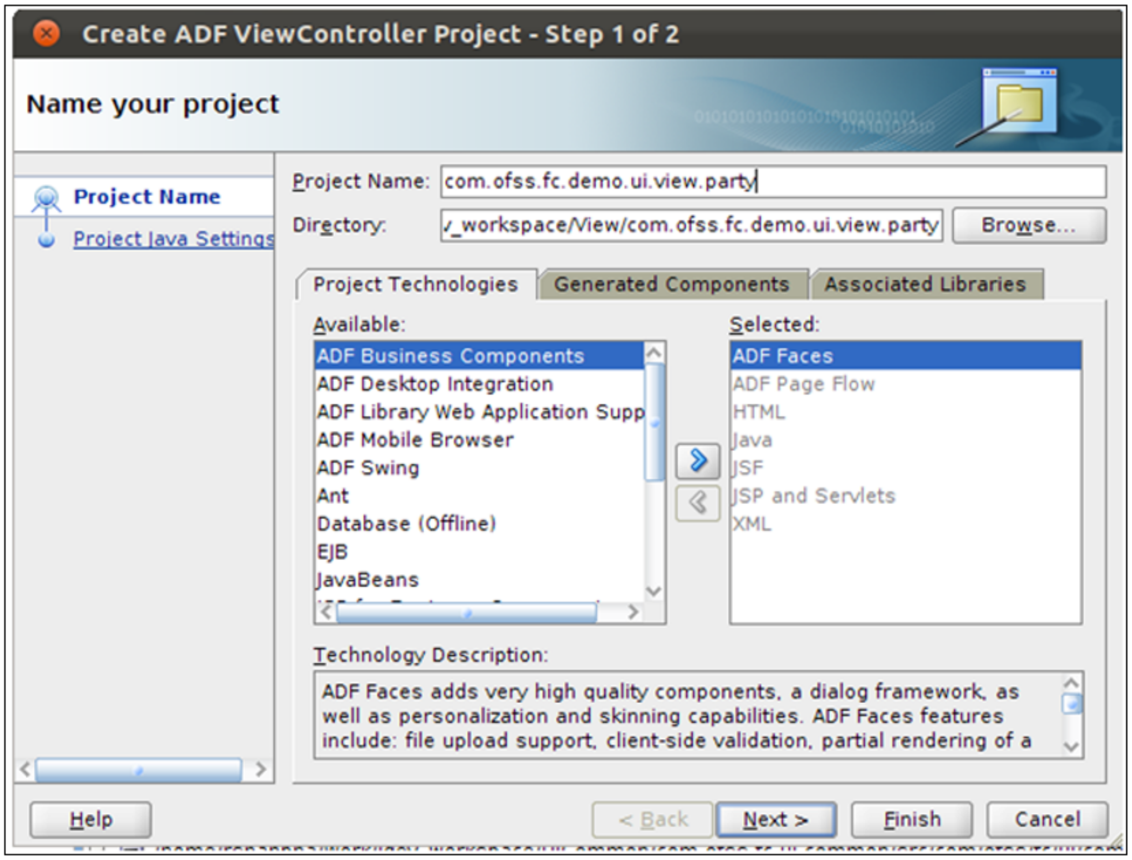
Categories:
- General
  - Applications
  - Ant
  - Connections
  - Deployment Descriptors
  - Deployment Profiles
  - Diagrams
  - Java
  - Projects
  - XML
- Web Tier
  - Applet
  - HTML
  - JSF
  - JSP
  - Servlets
- All Items

Items: ☐ Show All Descriptions

- Generic Project
- ADF Model Project
- ADF Swing Project
- **ADF ViewController Project**
  Creates a project that defines view and controller components for an ADF web application using ADF Faces and ADF Task Flows.
- EJB Project
- Java Application Project
- Java Project
- Project from Existing Source
- Project from WAR File
- Project Template

3. Click on Finish to finish creating the project.

Figure 5–59 Name your Project



Project Name: com.ofss.fc.demo.ui.view.party

Directory: /_workspace/View/com.ofss.fc.demo.ui.view.party

Available:

- ADF Business Components
- ADF Desktop Integration
- ADF Library Web Application Supp
- ADF Mobile Browser
- ADF Swing
- Ant
- Database (Offline)
- EJB
- JavaBeans

Selected:

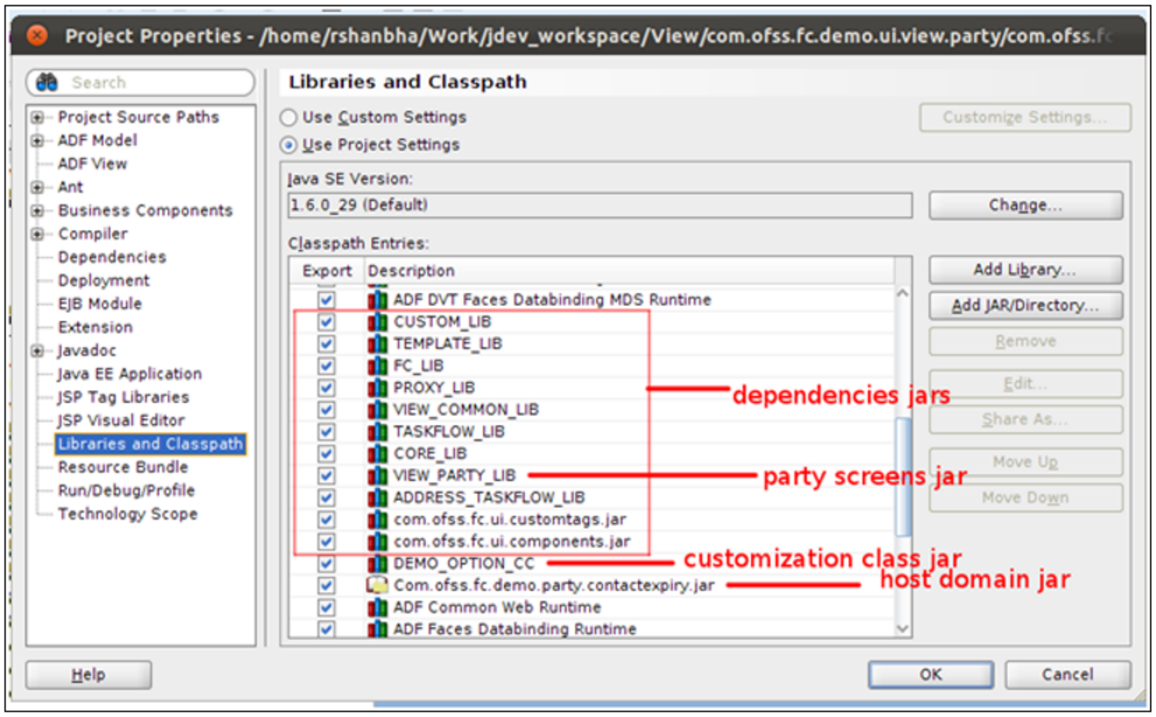
- ADF Faces
- ADF Page Flow
- HTML
- Java
- JSF
- JSP and Servlets
- XML

Technology Description:

ADF Faces adds very high quality components, a dialog framework, as well as personalization and skinning capabilities. ADF Faces features include: file upload support, client-side validation, partial rendering of a

4. Right click on the project and go to Project Properties. In the Libraries and Classpath tab, add the following:
5. The Jar containing the screen to be customized (com.ofss.fc.ui.view.party.jar).
6. The Jar containing the domain objects and services for Contact Expiry (com.ofss.fc.demo.party.contactexpiry.jar) as created in host application project.
7. All the required dependent Jars for the above Jars.
8. The Jar containing the customization class (com.ofss.fc.demo.ui.OptionCC.jar).
9. In the Dependencies tab, browse for and add the previously created adf model project (com.ofss.fc.demo.ui.model.party).
10. In the ADF View tab, check the Enable Seeded Customizations option to enable this project for customizations.

Figure 5–60 Libraries and Classpath


Project Properties - /home/rshanbha/Work/jdev_workspace/View/com.ofss.fc.demo.ui.view.party/com.ofss.f

Libraries and Classpath

Use Custom Settings

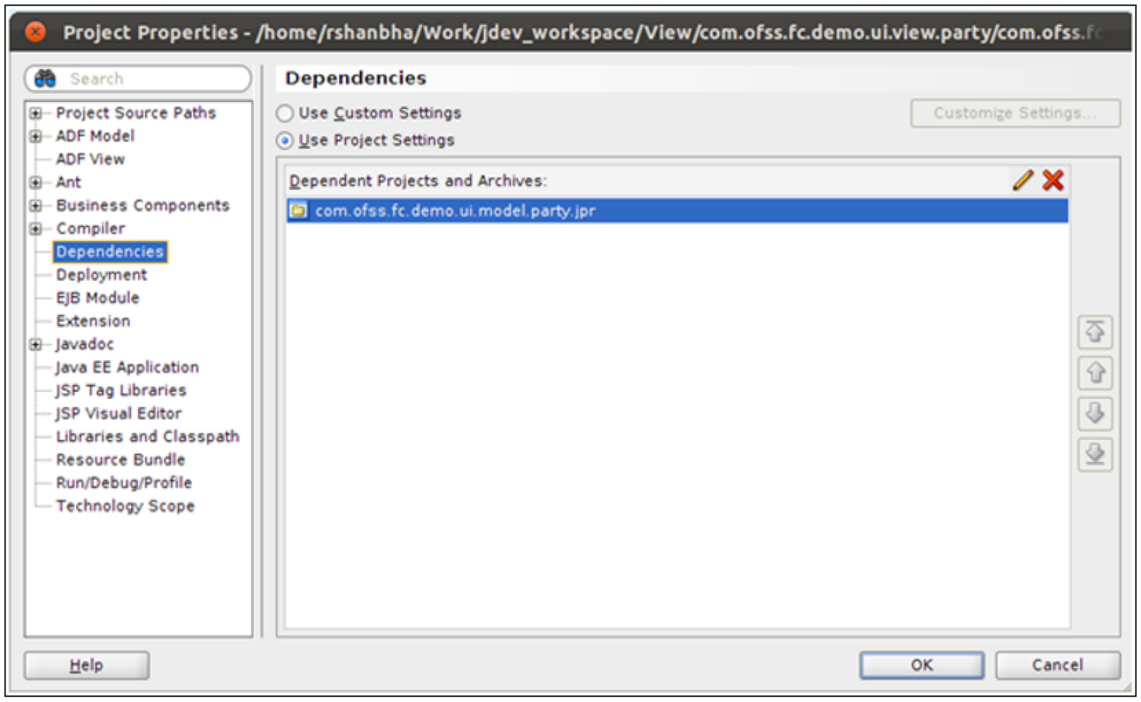
[✓ Use Project Settings]

Java SE Version: 1.6.0_29 (Default)

Classpath Entries:

Export	Description
✓	ADF DVT Faces Databinding MDS Runtime
✓	CUSTOM_LIB
✓	TEMPLATE_LIB
✓	FC_LIB
✓	PROXY_LIB
✓	VIEW_COMMON_LIB
✓	TASKFLOW_LIB
✓	CORE_LIB
✓	VIEW_PARTY_LIB
✓	ADDRESS_TASKFLOW_LIB
✓	com.ofss.fc.ui.customtags.jar
✓	com.ofss.fc.ui.components.jar
✓	DEMO_OPTION_CC
✓	Com.ofss.fc.demo.party.contactexpiry.jar
✓	ADF Common Web Runtime
✓	ADF Faces Databinding Runtime

Figure 5–61 Dependencies


Project Properties - /home/rshanbha/Work/jdev_workspace/View/com.ofss.fc.demo.ui.view.party/com.ofss.f

Dependencies

Use Custom Settings

[✓ Use Project Settings]

Dependent Projects and Archives:

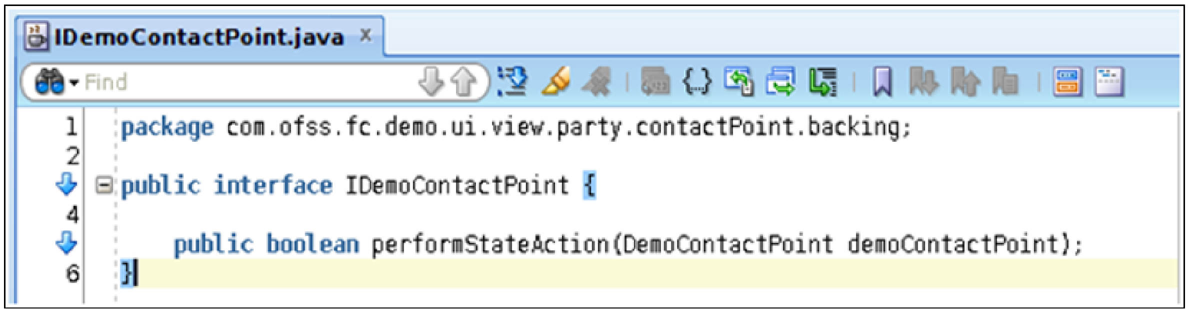
com.ofss.fc.demo.ui.model.party.jpr

11. Save the changes by clicking OK and rebuild the project.

Step 16 Create Maintenance State Action Interface

Create an interface containing the method definition for a maintenance action. This interface is implemented by the required maintenance state actions classes for the screen to be customized. The state action method will take the instance of the backing bean as a parameter.

Figure 5–62 Create Maintenance State Action Interface



```
package com.ofss.fc.demo.ui.view.party.contactPoint.backing;

public interface IDemoContactPoint {

    public boolean performStateAction(DemoContactPoint demoContactPoint);
}
```

Step 17 Create State Action Class

You need to create a class which will contain the business logic for the create transaction for this screen. This class should have following features:

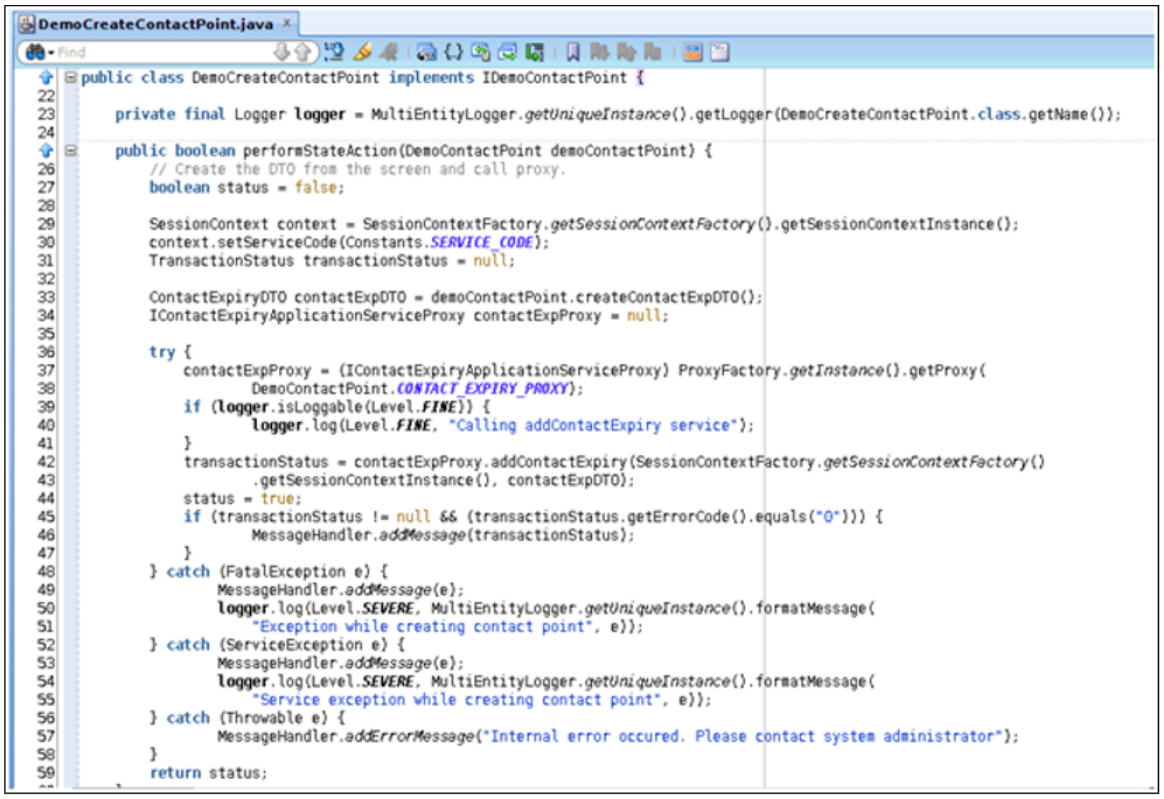
- Implements the previously created state action interface.
- Creates the Contact Point DTO from the users input.
- Creates an instance of the Contact Point service proxy.
- Calls the add method of the service passing the DTO.

Step 18 Create Update State Action Class

You will need to create a class which will contain the business logic for the update transaction for this screen. This class should have following features:

- Implements the previously created state action interface.
- Creates the Contact Point DTO from the users input.
- Creates an instance of the Contact Point service proxy.
- Calls the update method of the service passing the DTO.

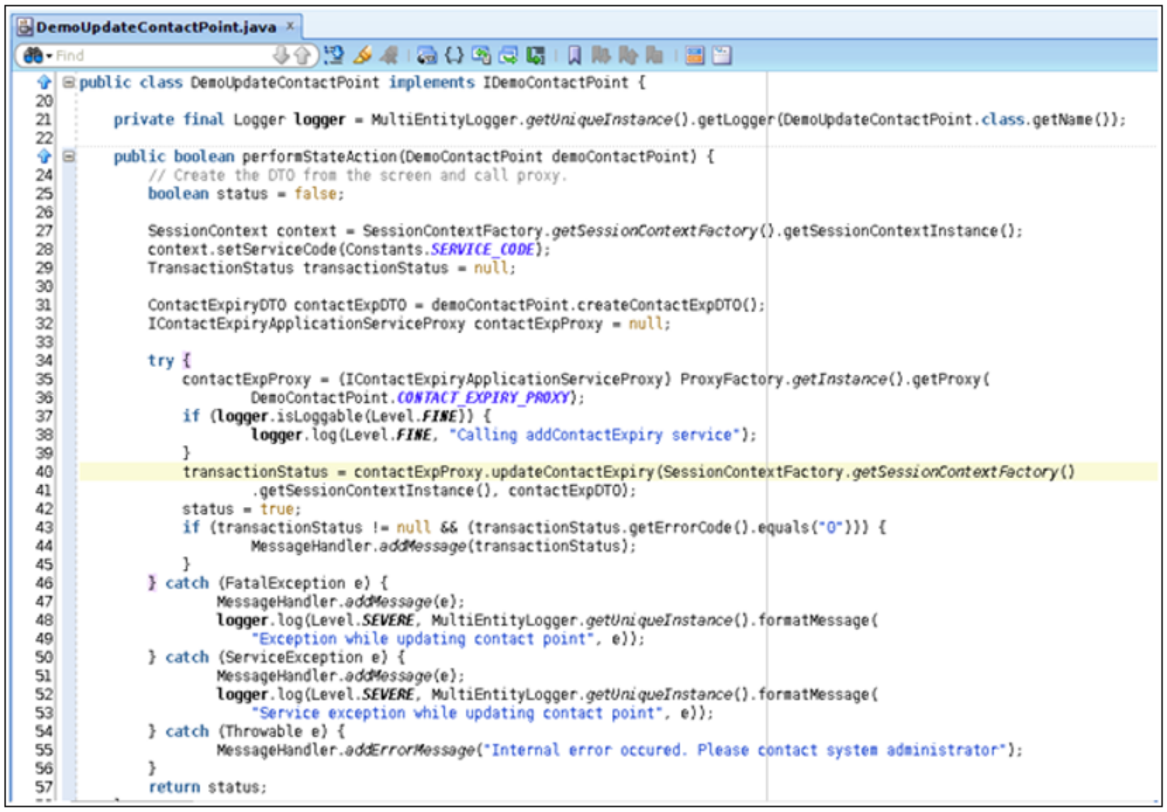
Figure 5–63 Create Update State Action Class



DemoCreateContactPoint.java

```
public class DemoCreateContactPoint implements IDemoContactPoint {

    private final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(DemoCreateContactPoint.class.getName());

    public boolean performStateAction(DemoContactPoint demoContactPoint) {
        // Create the DTO from the screen and call proxy.
        boolean status = false;

        SessionContext context = SessionContextFactory.getSessionContextFactory().getSessionContextInstance();
        context.setServiceCode(Constants.SERVICE_CODE);
        TransactionStatus transactionStatus = null;

        ContactExpiryDTO contactExpDTO = demoContactPoint.createContactExpDTO();
        IContactExpiryApplicationServiceProxy contactExpProxy = null;

        try {
            contactExpProxy = (IContactExpiryApplicationServiceProxy) ProxyFactory.getInstance().getProxy(
                    DemoContactPoint.CONTACT_EXPIRY_PROXY);
            if (logger.isLoggable(Level.FINE)) {
                    logger.log(Level.FINE, "Calling addContactExpiry service");
            }
            transactionStatus = contactExpProxy.addContactExpiry(SessionContextFactory.getSessionContextFactory()
                    .getSessionContextInstance(), contactExpDTO);
            status = true;
            if (transactionStatus != null && (transactionStatus.getErrorCode().equals("0"))) {
                    MessageHandler.addMessage(transactionStatus);
            }
        } catch (FatalException e) {
                MessageHandler.addMessage(e);
                logger.log(Level.SEVERE, MultiEntityLogger.getUniqueInstance().formatMessage(
                    "Exception while creating contact point", e));
        } catch (ServiceException e) {
                MessageHandler.addMessage(e);
                logger.log(Level.SEVERE, MultiEntityLogger.getUniqueInstance().formatMessage(
                    "Service exception while creating contact point", e));
        } catch (Throwable e) {
                MessageHandler.addErrorMessage("Internal error occured. Please contact system administrator");
        }
        return status;
```

Figure 5–64 Create Update State Action Class



DemoUpdateContactPoint.java

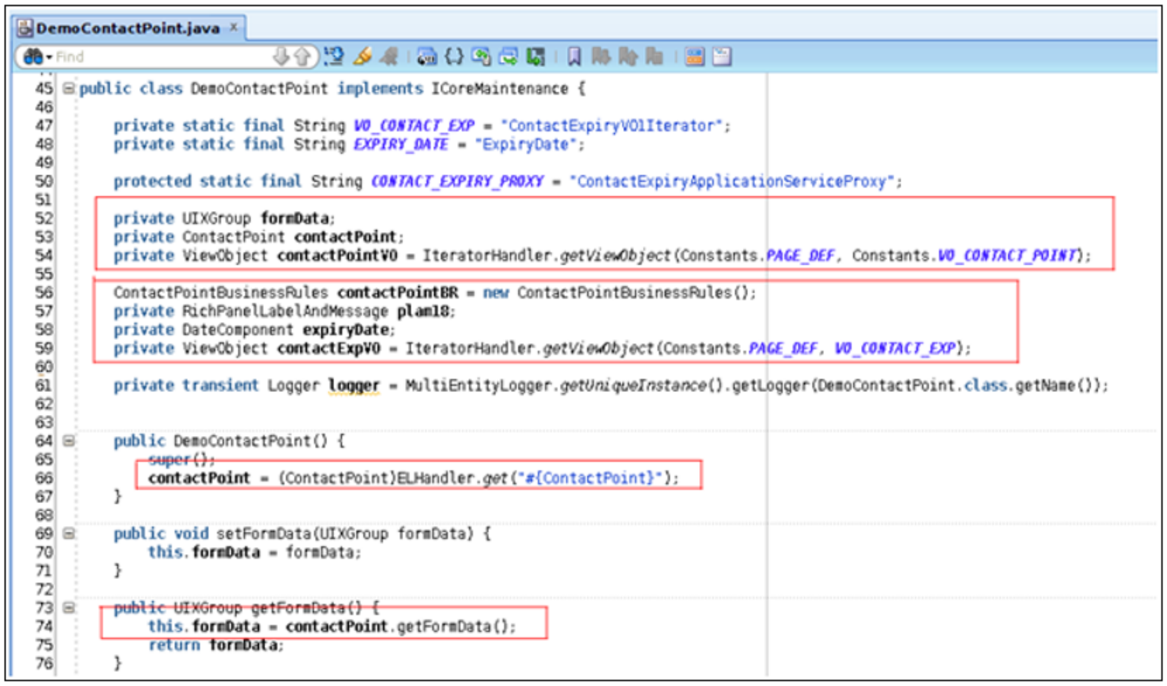
```java
public class DemoUpdateContactPoint implements IDemoContactPoint {

    private final Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(DemoUpdateContactPoint.class.getName());

    public boolean performStateAction(DemoContactPoint demoContactPoint) {
        // Create the DTO from the screen and call proxy.
        boolean status = false;

        SessionContext context = SessionContextFactory.getSessionContextFactory().getSessionContextInstance();
        context.setServiceCode(Constants.SERVICE_CODE);
        TransactionStatus transactionStatus = null;

        ContactExpiryDTO contactExpDTO = demoContactPoint.createContactExpDTO();
        IContactExpiryApplicationServiceProxy contactExpProxy = null;

        try {
            contactExpProxy = (IContactExpiryApplicationServiceProxy) ProxyFactory.getInstance().getProxy(
                    DemoContactPoint.CONTACT_EXPIRY_PROXY);
            if (logger.isLoggable(Level.FINE)) {
                    logger.log(Level.FINE, "Calling addContactExpiry service");
            }
            transactionStatus = contactExpProxy.updateContactExpiry(SessionContextFactory.getSessionContextFactory()
                    .getSessionContextInstance(), contactExpDTO);
            status = true;
            if (transactionStatus != null && (transactionStatus.getErrorCode().equals("0"))) {
                    MessageHandler.addMessage(transactionStatus);
            }
        } catch (FatalException e) {
                MessageHandler.addMessage(e);
                logger.log(Level.SEVERE, MultiEntityLogger.getUniqueInstance().formatMessage(
                    "Exception while updating contact point", e));
        } catch (ServiceException e) {
                MessageHandler.addMessage(e);
                logger.log(Level.SEVERE, MultiEntityLogger.getUniqueInstance().formatMessage(
                    "Service exception while updating contact point", e));
        } catch (Throwable e) {
                MessageHandler.addErrorMessage("Internal error occured. Please contact system administrator");
        }
        return status;
```

Step 19 Create Backing Bean

You need to create a backing bean class for the screen to be customized. This class should have the following features:

- Should implement the interface ICoreMaintenance.
- Private members for the to be added UI Components in customization and public accessors for the same.
- Private member for the backing bean of the original backing bean of the screen (ContactPoint) which is initialized in the constructor of this class.
- Private member for the parent UI Component of the newly added UI components and public accessors which returns the corresponding component of the backing bean.
- Private member for the newly added view object (ContactExpiryVO) and the current view objects present on the screen.

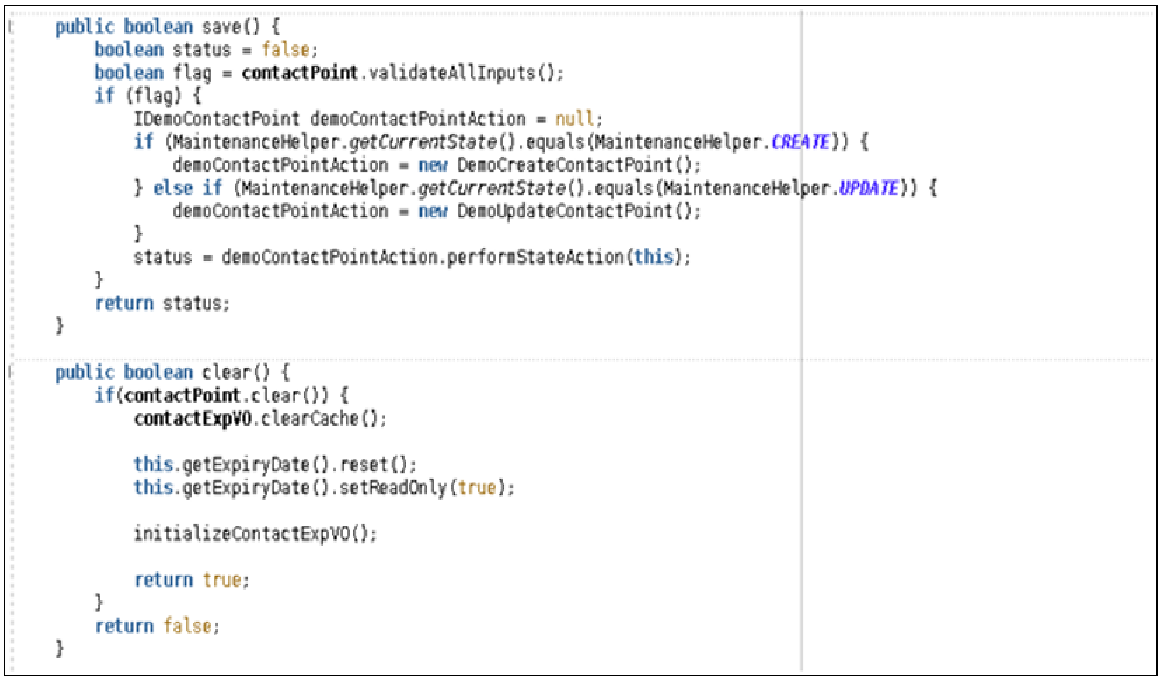
Figure 5–65 DemoContactPoint.java displays the View Objects



```java
public class DemoContactPoint implements ICoreMaintenance {

    private static final String VO_CONTACT_EXP = "ContactExpiryVO1Iterator";
    private static final String EXPIRY_DATE = "ExpiryDate";

    protected static final String CONTACT_EXPIRY_PROXY = "ContactExpiryApplicationServiceProxy";

    private UIXGroup formData;
    private ContactPoint contactPoint;
    private ViewObject contactPointVO = IteratorHandler.getViewObject(Constants.PAGE_DEF, Constants.VO_CONTACT_POINT);

    ContactPointBusinessRules contactPointBR = new ContactPointBusinessRules();
    private RichPanelLabelAndMessage plam18;
    private DateComponent expiryDate;
    private ViewObject contactExpVO = IteratorHandler.getViewObject(Constants.PAGE_DEF, VO_CONTACT_EXP);

    private transient Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(DemoContactPoint.class.getName());


    public DemoContactPoint() {
        super();
        contactPoint = (ContactPoint)ELHandler.get("#{ContactPoint}");
    }

    public void setFormData(UIXGroup formData) {
        this.formData = formData;
    }

    public UIXGroup getFormData() {
        this.formData = contactPoint.getFormData();
        return formData;
    }
```

- clear() method which handles the user action Clear.
- save() method which handles the maintenance state actions Create and Update.
- Depending on the current state action, the save() method should instantiate either DemoCreateContactPoint or DemoUpdateContactPoint and perform the corresponding state action methods.

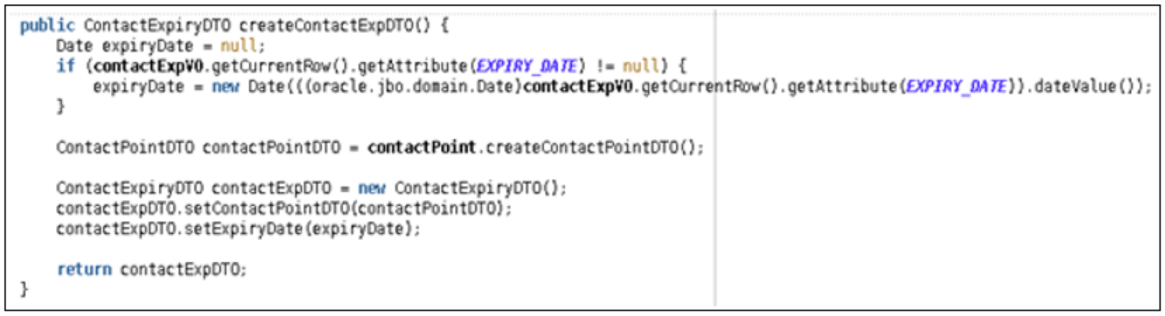
Figure 5–66 DemoCreateContactPoint / DemoUpdateContactPoint



```
public boolean save() {
    boolean status = false;
    boolean flag = contactPoint.validateAllInputs();
    if (flag) {
        IDemoContactPoint demoContactPointAction = null;
        if (MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.CREATE)) {
            demoContactPointAction = new DemoCreateContactPoint();
        } else if (MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.UPDATE)) {
            demoContactPointAction = new DemoUpdateContactPoint();
        }
        status = demoContactPointAction.performStateAction(this);
    }
    return status;
}

public boolean clear() {
    if(contactPoint.clear()) {
        contactExpVO.clearCache();

        this.getExpiryDate().reset();
        this.getExpiryDate().setReadOnly(true);

        initializeContactExpVO();

        return true;
    }
    return false;
}
```

- A public method to create the Contact Expiry DTO from the user's input on the screen.

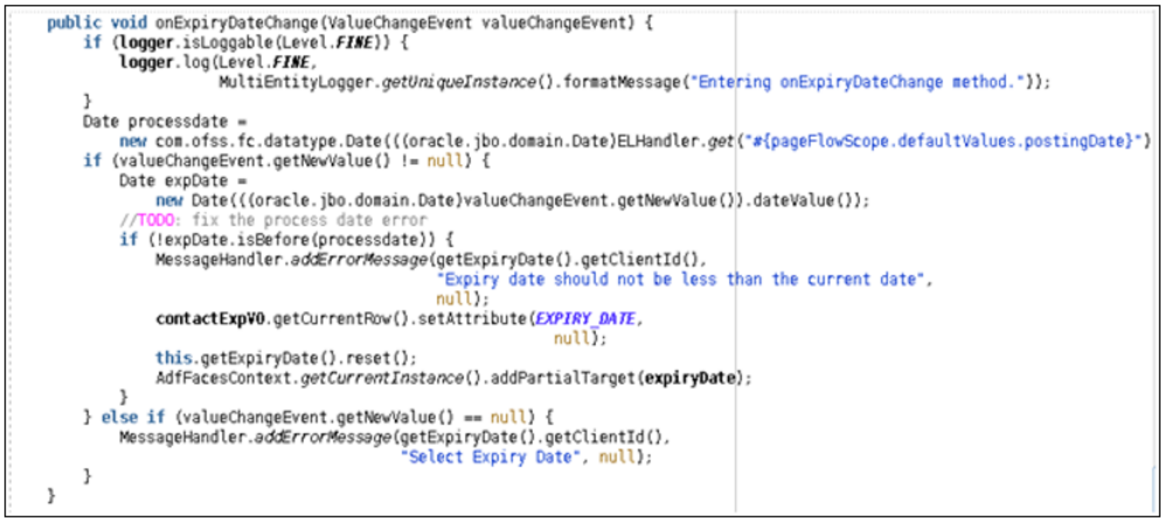
Figure 5–67 Create Contact Expiry DTO



```
public ContactExpiryDTO createContactExpDTO() {
    Date expiryDate = null;
    if (contactExpVO.getCurrentRow().getAttribute(EXPIRY_DATE) != null) {
        expiryDate = new Date(((oracle.jbo.domain.Date)contactExpVO.getCurrentRow().getAttribute(EXPIRY_DATE)).dateValue());
    }

    ContactPointDTO contactPointDTO = contactPoint.createContactPointDTO();

    ContactExpiryDTO contactExpDTO = new ContactExpiryDTO();
    contactExpDTO.setContactPointDTO(contactPointDTO);
    contactExpDTO.setExpiryDate(expiryDate);

    return contactExpDTO;
}
```

- A value change event handler for the Expiry Date UI Component.

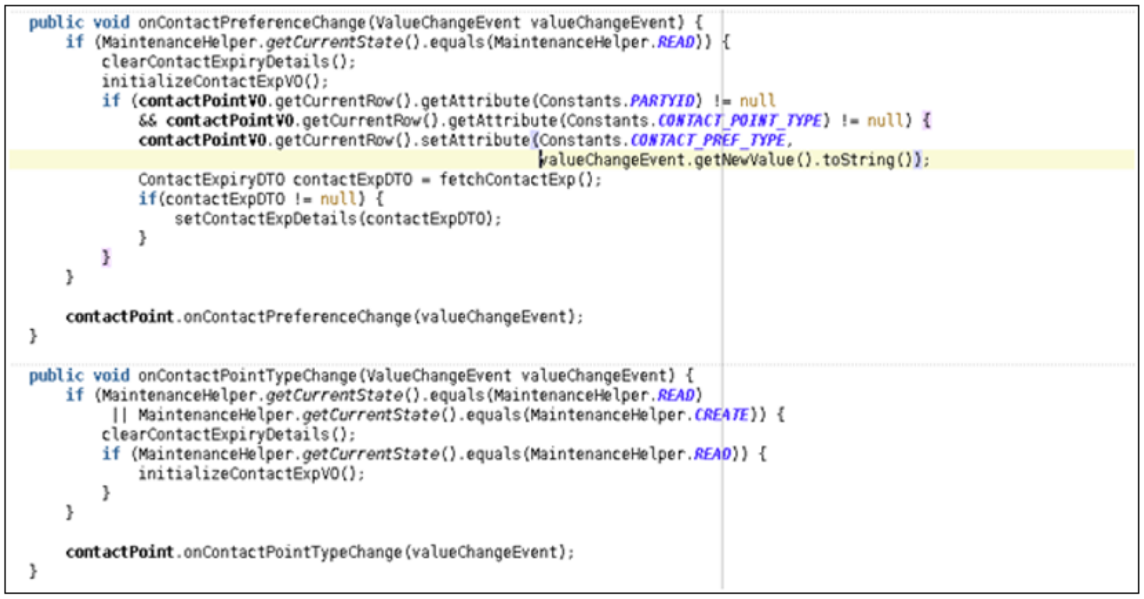
Figure 5–68 Value Change Event Handler for the Expiry Date UI Component



```java
public void onExpiryDateChange(ValueChangeEvent valueChangeEvent) {
    if (logger.isLoggable(Level.FINE)) {
        logger.log(Level.FINE,
                   MultiEntityLogger.getUniqueInstance().formatMessage("Entering onExpiryDateChange method."));
    }
    Date processdate =
        new com.ofss.fc.datatype.Date(((oracle.jbo.domain.Date)ELHandler.get("#{pageFlowScope.defaultValues.postingDate}")
    if (valueChangeEvent.getNewValue() != null) {
        Date expDate =
            new Date(((oracle.jbo.domain.Date)valueChangeEvent.getNewValue()).dateValue());
        //TODO: fix the process date error
        if (!expDate.isBefore(processdate)) {
            MessageHandler.addErrorMessage(getExpiryDate().getClientId(),
                                           "Expiry date should not be less than the current date",
                                           null);
            contactExpVO.getCurrentRow().setAttribute(EXPIRY_DATE,
                                                      null);
            this.getExpiryDate().reset();
            AdfFacesContext.getCurrentInstance().addPartialTarget(expiryDate);
        }
    } else if (valueChangeEvent.getNewValue() == null) {
        MessageHandler.addErrorMessage(getExpiryDate().getClientId(),
                                       "Select Expiry Date", null);
    }
}
```

- Value change event handlers for the existing UI Components on change of which the screen data is to be fetched.

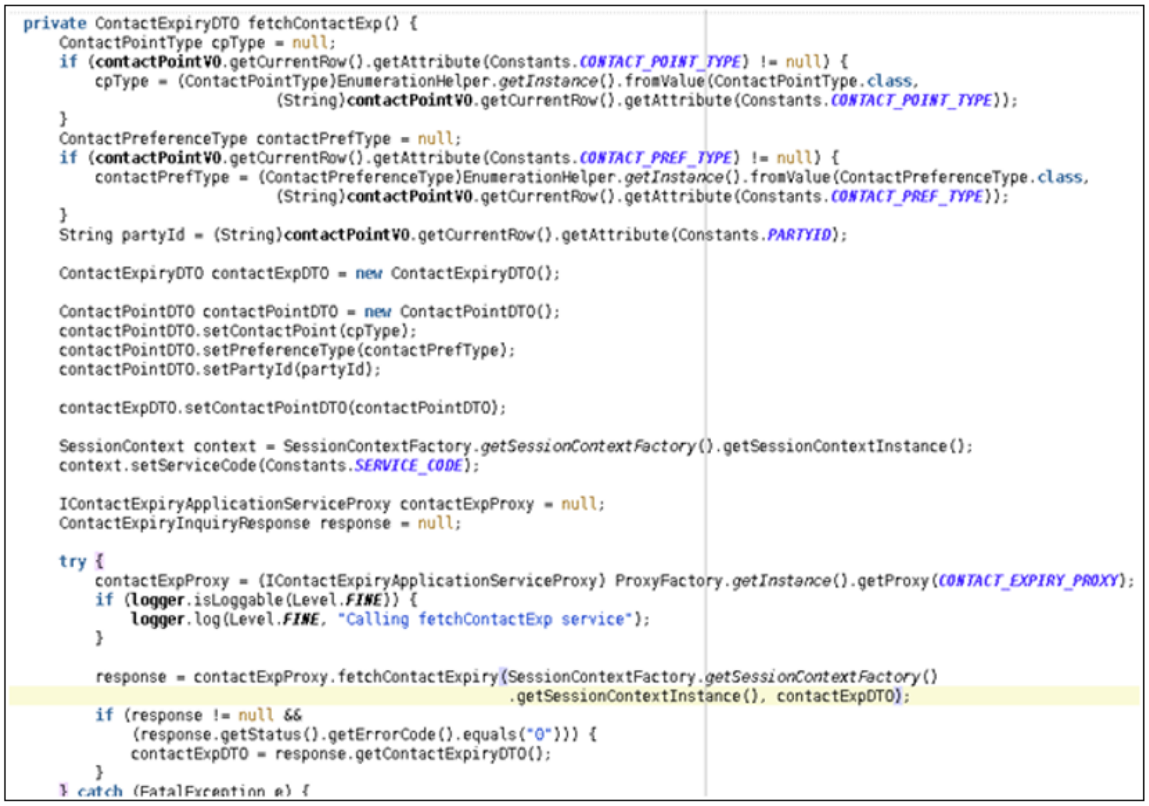
Figure 5–69 Value Change Event Handlers for Existing UI Components



```java
public void onContactPreferenceChange(ValueChangeEvent valueChangeEvent) {
    if (MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.READ)) {
        clearContactExpiryDetails();
        initializeContactExpVO();
        if (contactPointVO.getCurrentRow().getAttribute(Constants.PARTYID) != null
            && contactPointVO.getCurrentRow().getAttribute(Constants.CONTACT_POINT_TYPE) != null) {
            contactPointVO.getCurrentRow().setAttribute(Constants.CONTACT_PREF_TYPE,
                                                        valueChangeEvent.getNewValue().toString());
            ContactExpiryDTO contactExpDTO = fetchContactExp();
            if(contactExpDTO != null) {
                setContactExpDetails(contactExpDTO);
            }
        }
    }

    contactPoint.onContactPreferenceChange(valueChangeEvent);
}

public void onContactPointTypeChange(ValueChangeEvent valueChangeEvent) {
    if (MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.READ)
        || MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.CREATE)) {
        clearContactExpiryDetails();
        if (MaintenanceHelper.getCurrentState().equals(MaintenanceHelper.READ)) {
            initializeContactExpVO();
        }
    }

    contactPoint.onContactPointTypeChange(valueChangeEvent);
}
```

- Method containing the business logic to fetch screen data using Contact Expiry proxy service.

Figure 5–70 Method to fetch Screen Data using Contact Expiry Proxy Service



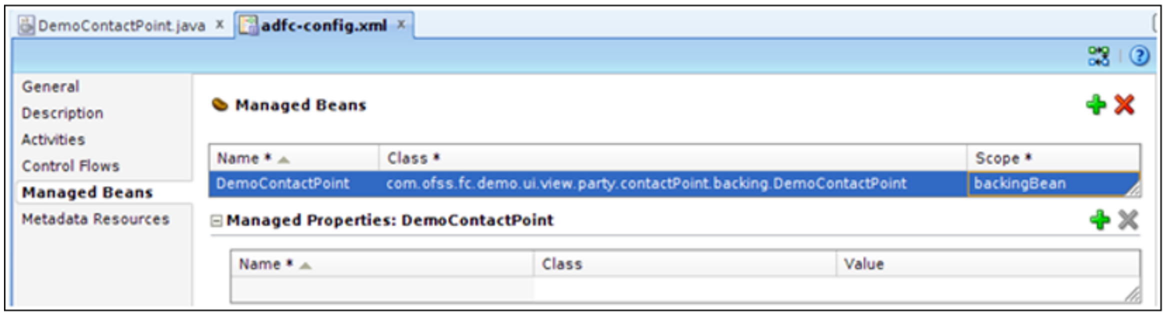
```
private ContactExpiryDTO fetchContactExp() {
    ContactPointType cpType = null;
    if (contactPointVO.getCurrentRow().getAttribute(Constants.CONTACT_POINT_TYPE) != null) {
        cpType = (ContactPointType)EnumerationHelper.getInstance().fromValue(ContactPointType.class,
                    (String)contactPointVO.getCurrentRow().getAttribute(Constants.CONTACT_POINT_TYPE));
    }
    ContactPreferenceType contactPrefType = null;
    if (contactPointVO.getCurrentRow().getAttribute(Constants.CONTACT_PREF_TYPE) != null) {
        contactPrefType = (ContactPreferenceType)EnumerationHelper.getInstance().fromValue(ContactPreferenceType.class,
                    (String)contactPointVO.getCurrentRow().getAttribute(Constants.CONTACT_PREF_TYPE));
    }
    String partyId = (String)contactPointVO.getCurrentRow().getAttribute(Constants.PARTYID);

    ContactExpiryDTO contactExpDTO = new ContactExpiryDTO();

    ContactPointDTO contactPointDTO = new ContactPointDTO();
    contactPointDTO.setContactPoint(cpType);
    contactPointDTO.setPreferenceType(contactPrefType);
    contactPointDTO.setPartyId(partyId);

    contactExpDTO.setContactPointDTO(contactPointDTO);

    SessionContext context = SessionContextFactory.getSessionContextFactory().getSessionContextInstance();
    context.setServiceCode(Constants.SERVICE_CODE);

    IContactExpiryApplicationServiceProxy contactExpProxy = null;
    ContactExpiryInquiryResponse response = null;

    try {
        contactExpProxy = (IContactExpiryApplicationServiceProxy) ProxyFactory.getInstance().getProxy(CONTACT_EXPIRY_PROXY);
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE, "Calling fetchContactExp service");
        }

        response = contactExpProxy.fetchContactExpiry(SessionContextFactory.getSessionContextFactory()
                                        .getSessionContextInstance(), contactExpDTO);
        if (response != null &&
            (response.getStatus().getErrorCode().equals("0"))) {
            contactExpDTO = response.getContactExpiryDTO();
        }
    } catch (FatalException e) {
```

Step 20 Create Managed Bean

You need to register the DemoContactPoint backing bean as a managed bean with a backing bean scope.

1. Open the project's adfc-config.xml which is present in the WEB-INF folder.
2. In the Managed Beans tab, add the binding bean class as a managed bean with backing bean scope as follows:

Figure 5–71 Create Managed Bean



Managed Beans

Name *	Class *	Scope *
DemoContactPoint	com.ofss.fc.demo.ui.view.party.contactPoint.backing.DemoContactPoint	backingBean

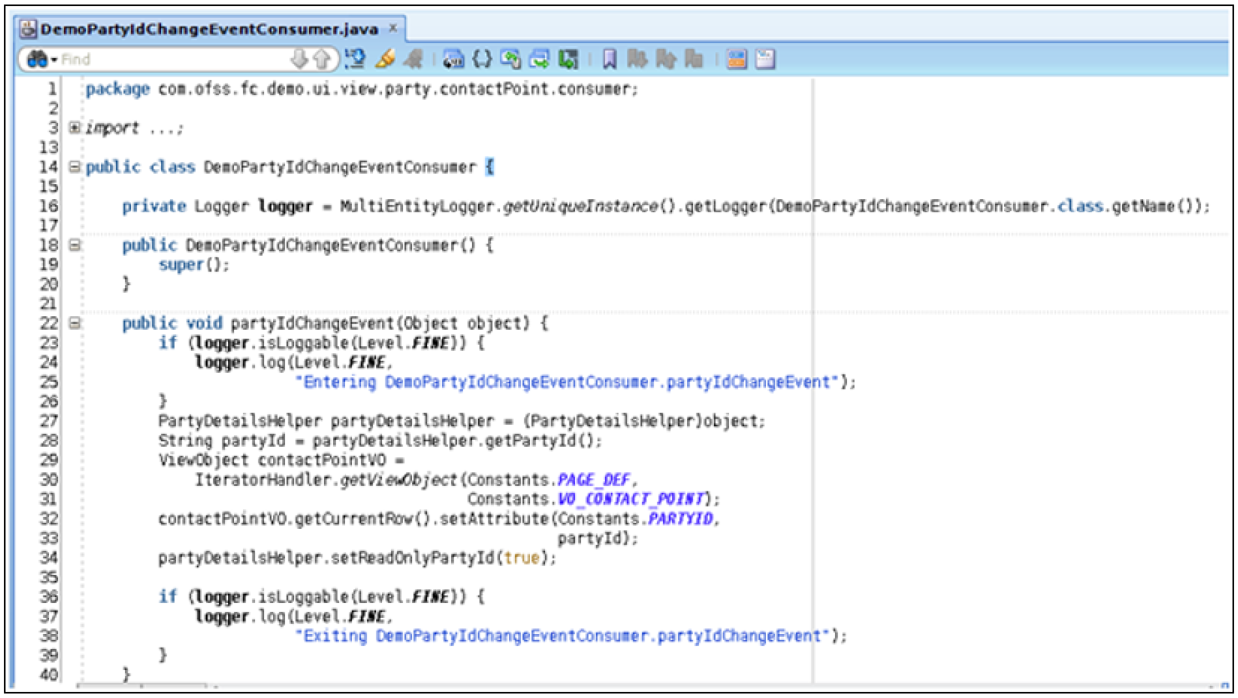
Managed Properties: DemoContactPoint

Name *	Class	Value

Step 21 Create Event Consumer Class

You need to create an event consumer class to consume the Party Id Change event. When the user inputs a party id on the screen, the business logic in this event consumer class will be executed automatically.

Figure 5–72 Create Event Customer Class



DemoPartyIdChangeEventConsumer.java

```java
package com.ofss.fc.demo.ui.view.party.contactPoint.consumer;

import ...;

public class DemoPartyIdChangeEventConsumer {

    private Logger logger = MultiEntityLogger.getUniqueInstance().getLogger(DemoPartyIdChangeEventConsumer.class.getName());

    public DemoPartyIdChangeEventConsumer() {
        super();
    }

    public void partyIdChangeEvent(Object object) {
        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE,
                       "Entering DemoPartyIdChangeEventConsumer.partyIdChangeEvent");
        }
        PartyDetailsHelper partyDetailsHelper = (PartyDetailsHelper)object;
        String partyId = partyDetailsHelper.getPartyId();
        ViewObject contactPointVO =
            IteratorHandler.getViewObject(Constants.PAGE_DEF,
                                          Constants.VO_CONTACT_POINT);
        contactPointVO.getCurrentRow().setAttribute(Constants.PARTYID,
                                                    partyId);
        partyDetailsHelper.setReadOnlyPartyId(true);

        if (logger.isLoggable(Level.FINE)) {
            logger.log(Level.FINE,
                       "Exiting DemoPartyIdChangeEventConsumer.partyIdChangeEvent");
        }
    }
```

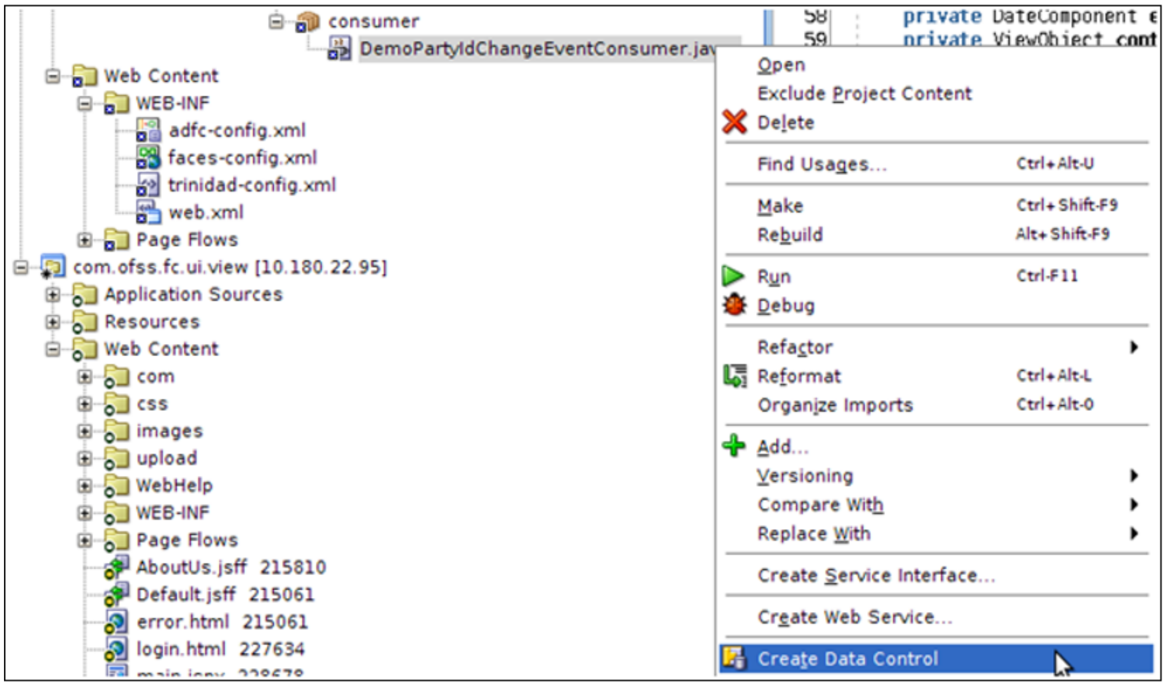
Step 22 Create Data Control

For the event consumer class's method to be exposed as an event handler, you need to create a data control for this class.

1. In the Application Navigator, right click on the event consumer Java file and create data control.

   On creation of data control, an XML file is generated for the class and a DataControls.dcx file is generated containing the information about the data controls present in the project. You will be able to see the event consumer data control in the Data Controls tab.

Figure 5–73 Create Data Control



```
private DateComponent e
private ViewObject cont
```

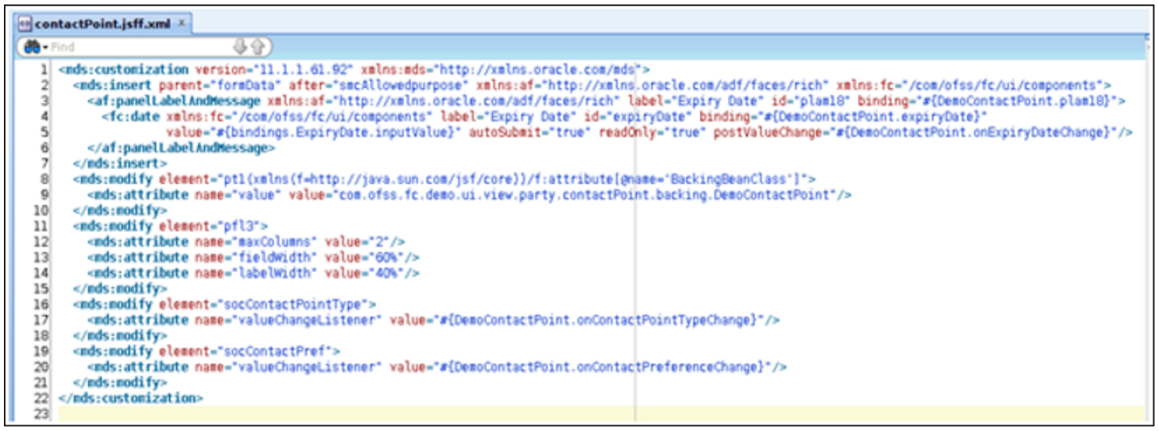
2. You should now restart JDeveloper in the Customization Developer Role to edit the customizations. Ensure that the appropriate Customization Context is selected.

Step 23 Add UI Components to Screen

Browse and locate the JSFF for the screen to be customized (com.ofss.fc.ui.view.party.contactPoint.contactPoint.jsff) inside the JAR (com.ofss.fc.ui.view.party.jar). Open the JSFF and do the required changes as follows:

1. Drag and drop the Panel Label & Message and Date UI components at the required position on the screen.
2. For each component, set the required attributes using the Property Inspector panel of JDeveloper.
3. Modify the containing Panel's width and number of columns attributes as required.
4. For each component, add the binding to the DemoContactPoint backing bean's corresponding members.
5. Add the value change event binding for the Expiry Date UI component to the backing bean's corresponding method.
6. Change the value change event binding for the existing UI component on change of which the screen data is fetched.
7. Change the backing bean attribute of the screen to the previously created DemoContactPoint backing bean.
8. Save the changes. You will notice that JDeveloper has created a customization XML in the ADF Library Customizations folder to save the differences between the base JSFF and the customized JSFF. The generated contactPoint.jsff.xml should look similar to as shown below.

Figure 5–74 Generated contactPoint.jsff.xml



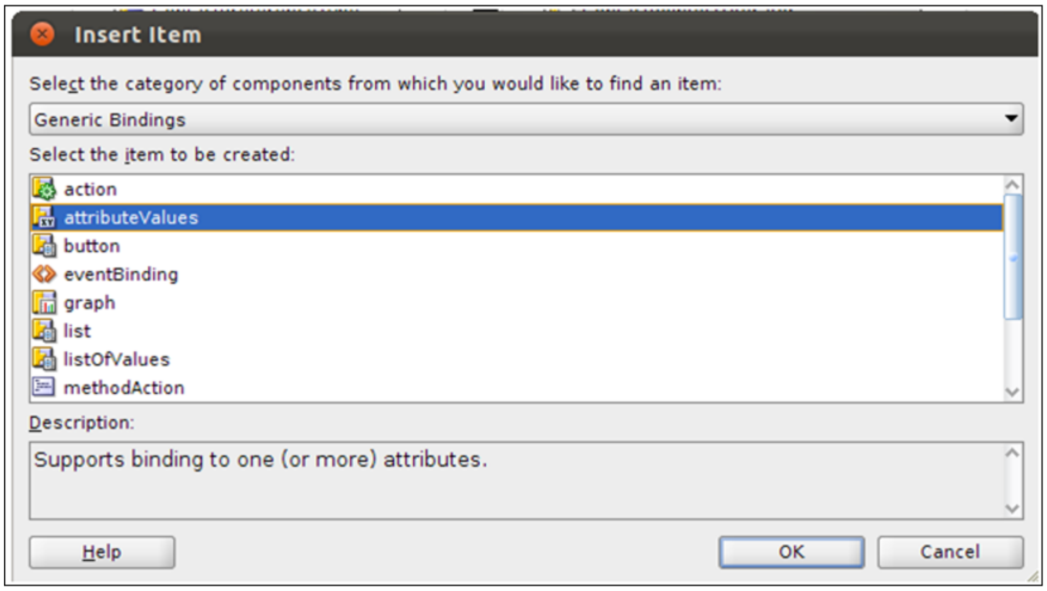
```
<mds:customization version="11.1.1.61.92" xmlns:mds="http://xmlns.oracle.com/mds">
  <mds:insert parent="formData" after="smcAllowedpurpose" xmlns:af="http://xmlns.oracle.com/adf/faces/rich" xmlns:fc="/com/ofss/fc/ui/components">
    <af:panelLabelAndMessage xmlns:af="http://xmlns.oracle.com/adf/faces/rich" label="Expiry Date" id="plam18" binding="#{DemoContactPoint.plam18}">
      <fc:date xmlns:fc="/com/ofss/fc/ui/components" label="Expiry Date" id="expiryDate" binding="#{DemoContactPoint.expiryDate}"
               value="#{bindings.ExpiryDate.inputValue}" autoSubmit="true" readOnly="true" postValueChange="#{DemoContactPoint.onExpiryDateChange}"/>
    </af:panelLabelAndMessage>
  </mds:insert>
  <mds:modify element="pt1(xmlns(f=http://java.sun.com/jsf/core))/f:attribute[@name='BackingBeanClass']">
    <mds:attribute name="value" value="com.ofss.fc.demo.ui.view.party.contactPoint.backing.DemoContactPoint"/>
  </mds:modify>
  <mds:modify element="pfl3">
    <mds:attribute name="maxColumns" value="2"/>
    <mds:attribute name="fieldWidth" value="60%"/>
    <mds:attribute name="labelWidth" value="40%"/>
  </mds:modify>
  <mds:modify element="socContactPointType">
    <mds:attribute name="valueChangeListener" value="#{DemoContactPoint.onContactPointTypeChange}"/>
  </mds:modify>
  <mds:modify element="socContactPref">
    <mds:attribute name="valueChangeListener" value="#{DemoContactPoint.onContactPreferenceChange}"/>
  </mds:modify>
</mds:customization>
```

Step 24 Add View Object Binding to Page Definition

You need to add the view object binding for the previously created ContactExpiryVO view object to the page definition of the screen to be customized.

1. Browse and locate the page definition for the screen to be customized (com.ofss.fc.ui.view.party.contactPoint.pageDef.ContactPointPageDef.xml) and open it.
2. Add an attributeValues binding as shown below.

Figure 5–75 Add an attributeValues binding



Insert Item

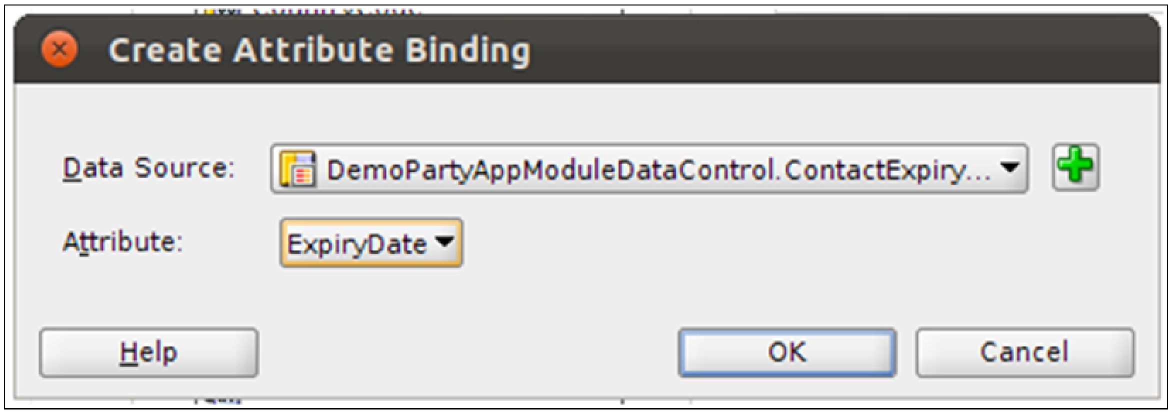
Select the category of components from which you would like to find an item: Generic Bindings

Select the item to be created: attributeValues

Description: Supports binding to one (or more) attributes.

3. For Data Source option, locate the previously created ContactExpiryVO view object present in the DemoPartyAppModule.
4. For Attribute option, choose the ExpiryDate attribute present in the view object.

Figure 5–76 Create Attribute Binding



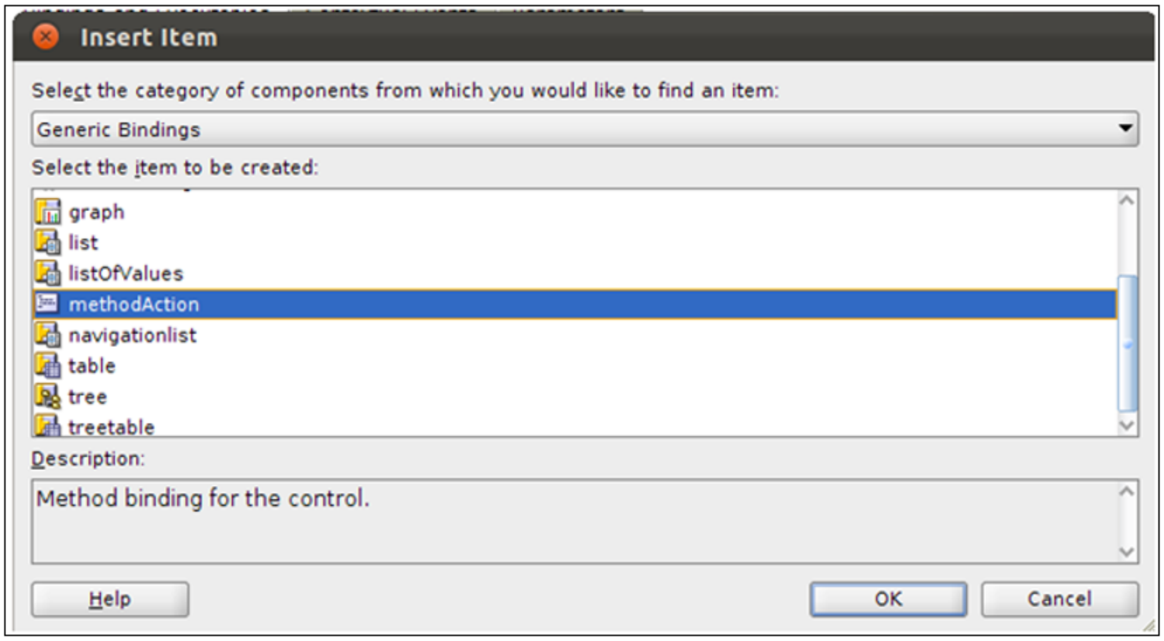
Data Source: DemoPartyAppModuleDataControl.ContactExpiry...

Attribute: ExpiryDate

Step 25 Add Method Action Binding to Page Definition
You need to add the method action binding for the previously created DemoPartyIdEventChangeConsumer event consumer class to the page definition of the screen to be customized.

1. Add a methodAction binding as shown below.

Figure 5–77 Add a methodAction binding



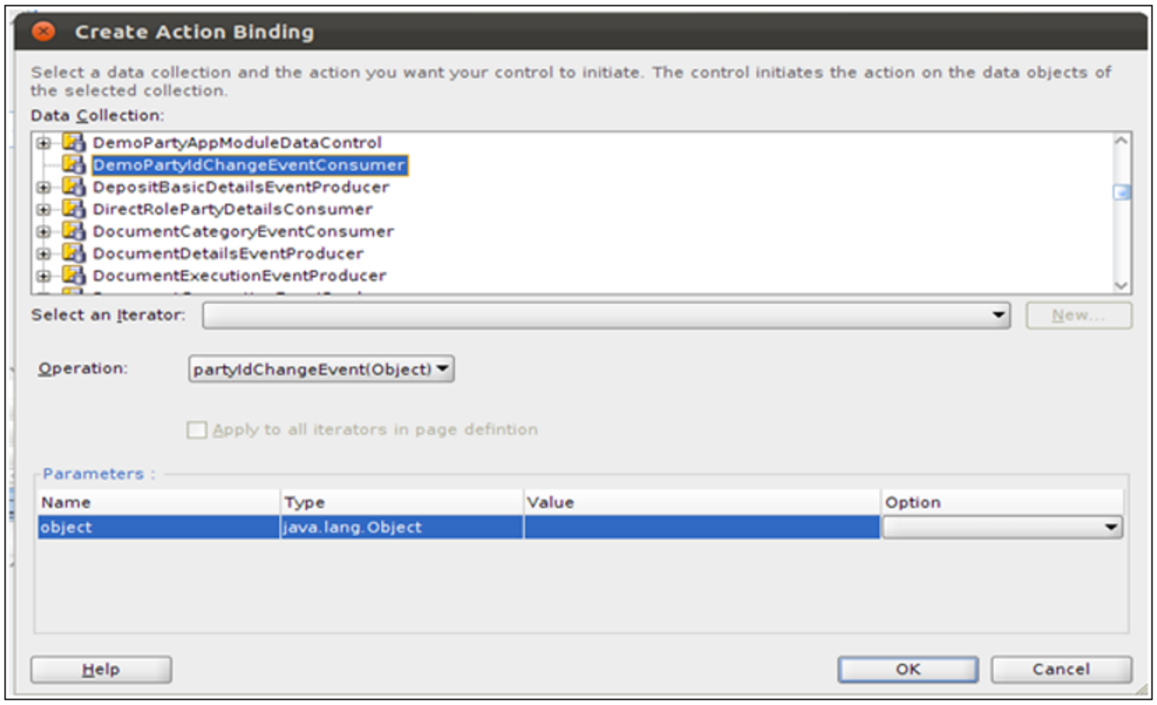
Select the category of components from which you would like to find an item: Generic Bindings

Select the item to be created: methodAction

Description: Method binding for the control.

2. For the Data Collection option, locate the previously created DemoPartyIdChangeEventConsumer data control.

Figure 5–78 Create Action Binding


Create Action Binding

Select a data collection and the action you want your control to initiate. The control initiates the action on the data objects of the selected collection.

Data Collection:

- DemoPartyAppModuleDataControl
- DemoPartyIdChangeEventConsumer
- DepositBasicDetailsEventProducer
- DirectRolePartyDetailsConsumer
- DocumentCategoryEventConsumer
- DocumentDetailsEventProducer
- DocumentExecutionEventProducer

Select an Iterator:

Operation: partyIdChangeEvent(Object)

Apply to all iterators in page defintion

Parameters :

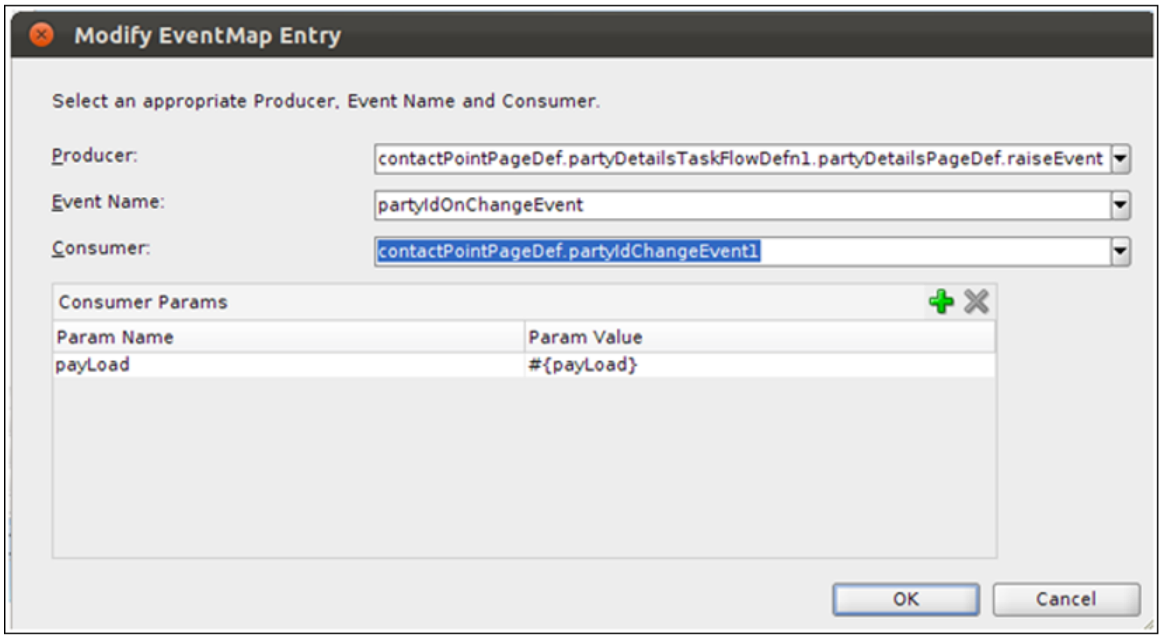
Name	Type	Value	Option
object	java.lang.Object		

Step 26 Edit Event Map of Page Definition

You need to map the Event Producer for the party id change event to the previously created Event Consumer.

1. In the Structure panel of JDeveloper, right click on the page definition and select Edit Event Map.
2. In the Event Map Editor dialog that opens, edit the mapping for the party id change event. Select the previously created Event Consumer's method.

Figure 5–79 Select the Event Consumer Method



Select an appropriate Producer, Event Name and Consumer.

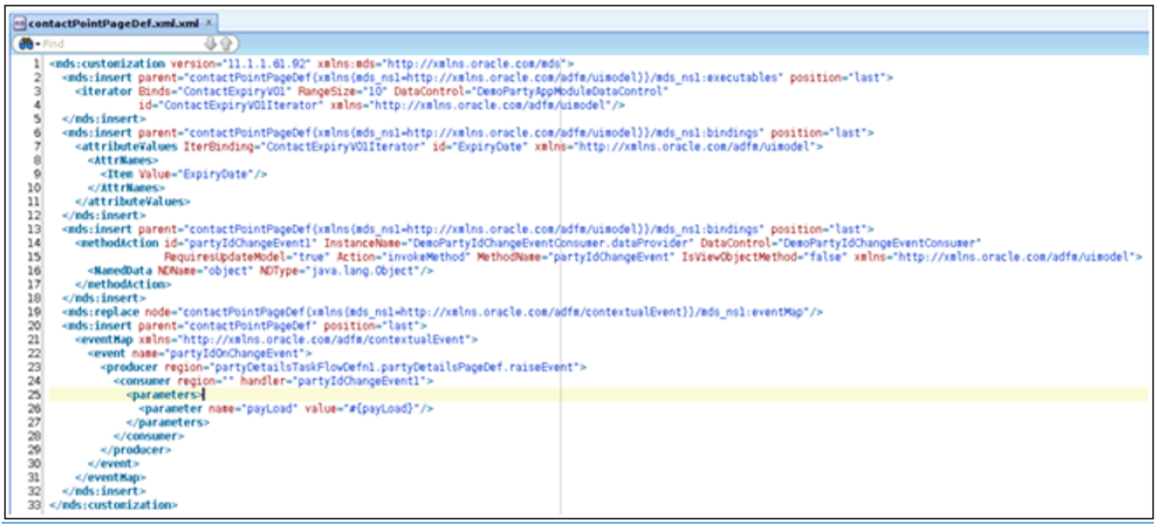
Producer:	contactPointPageDef.partyDetailsTaskFlowDefn1.partyDetailsPageDef.raiseEvent
Event Name:	partyIdOnChangeEvent
Consumer:	contactPointPageDef.partyIdChangeEvent1

Consumer Params

Param Name	Param Value
payLoad	#{payLoad}

3. Save the changes. You will notice that JDeveloper has created a customization XML in the ADF Library Customizations folder to save the differences between the base JSFF and the customized JSFF. The generated contactPoint.jsff.xml should look similar to as shown below.

Figure 5–80 Generated contactPoint.jsff.xml



```
<mds:customization version="11.1.1.61.92" xmlns:mds="http://xmlns.oracle.com/mds">
  <mds:insert parent="contactPointPageDef(xmlns(mds_ns1=http://xmlns.oracle.com/adfm/uimodel))/mds_ns1:executables" position="last">
    <iterator Binds="ContactExpiryVO1" RangeSize="10" DataControl="DemoPartyAppModuleDataControl"
              id="ContactExpiryVO1Iterator" xmlns="http://xmlns.oracle.com/adfm/uimodel"/>
  </mds:insert>
  <mds:insert parent="contactPointPageDef(xmlns(mds_ns1=http://xmlns.oracle.com/adfm/uimodel))/mds_ns1:bindings" position="last">
    <attributeValues IterBinding="ContactExpiryVO1Iterator" id="ExpiryDate" xmlns="http://xmlns.oracle.com/adfm/uimodel">
      <AttrNames>
        <Item Value="ExpiryDate"/>
      </AttrNames>
    </attributeValues>
  </mds:insert>
  <mds:insert parent="contactPointPageDef(xmlns(mds_ns1=http://xmlns.oracle.com/adfm/uimodel))/mds_ns1:bindings" position="last">
    <methodAction id="partyIdChangeEvent1" InstanceName="DemoPartyIdChangeEventConsumer.dataProvider" DataControl="DemoPartyIdChangeEventConsumer"
                  RequiresUpdateModel="true" Action="invokeMethod" MethodName="partyIdChangeEvent" IsViewObjectMethod="false" xmlns="http://xmlns.oracle.com/adfm/uimodel">
      <NamedData NDName="object" NDType="java.lang.Object"/>
    </methodAction>
  </mds:insert>
  <mds:replace node="contactPointPageDef(xmlns(mds_ns1=http://xmlns.oracle.com/adfm/contextualEvent))/mds_ns1:eventMap"/>
  <mds:insert parent="contactPointPageDef" position="last">
    <eventMap xmlns="http://xmlns.oracle.com/adfm/contextualEvent">
      <event name="partyIdOnChangeEvent">
        <producer region="partyDetailsTaskFlowDefn1.partyDetailsPageDef.raiseEvent">
          <consumer region="" handler="partyIdChangeEvent1">
            <parameters>
              <parameter name="payLoad" value="#{payLoad}"/>
            </parameters>
          </consumer>
        </producer>
      </event>
    </eventMap>
  </mds:insert>
</mds:customization>
```

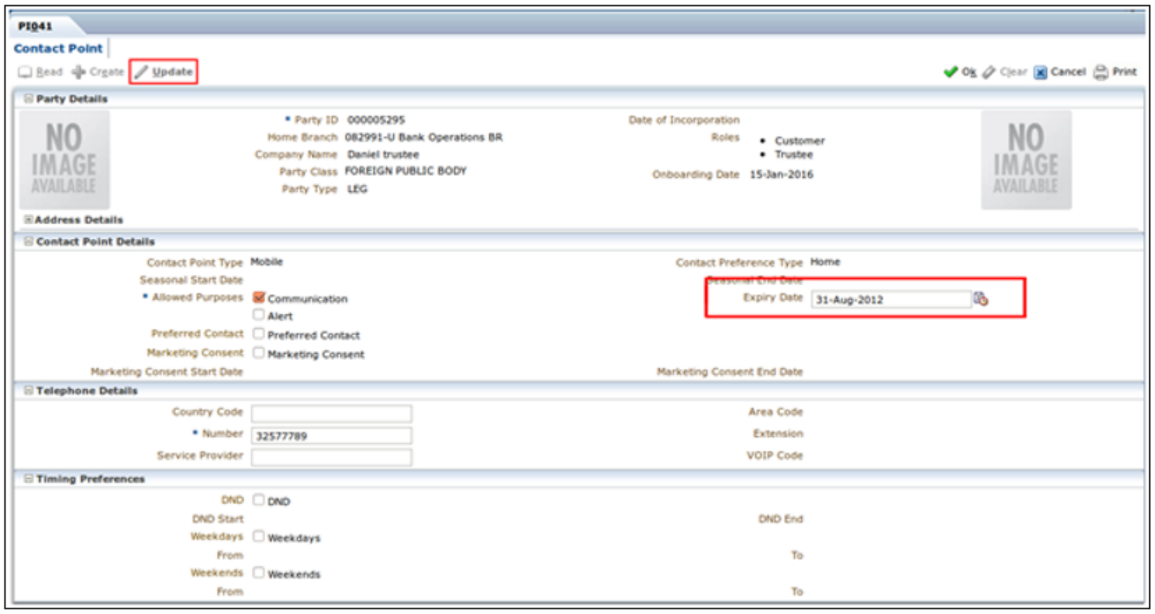
Step 27 Deploy Customization Project

After finishing the customization changes, exit the Customization Developer Role and start JDeveloper in Default Role. Deploy the view controller project as an ADF Library Jar (com.ofss.fc.demo.ui.view.party.jar).

Go to Project Properties of the main application project and in the Libraries and Classpath, add the following:

1. View controller project Jar (com.ofss.fc.demo.ui.view.party.jar)
2. Host domain Jar (com.ofss.fc.demo.party.contactexpiry.jar)
3. Customization Class Jar (com.ofss.fc.demo.ui.OptionCC.jar)
4. All dependency libraries and Jars for the project.
5. Start the application and navigate to Party → Contact Information → Contact Point screen. Input a party id on the screen and perform the read, create and update actions on Contact Point. You need to input data and fetch value for the newly added Expiry Date field.

Figure 5–81 PI041 - Contact Point Screen



Party Details

• Party ID	000005295	Date of Incorporation	
Home Branch	082991-U Bank Operations BR	Roles	• Customer • Trustee
Company Name	Daniel trustee	Onboarding Date	15-Jan-2016
Party Class	FOREIGN PUBLIC BODY		
Party Type	LEG		

Address Details

Contact Point Details

Contact Point Type	Mobile	Contact Preference Type	Home
Seasonal Start Date		Seasonal End Date	
• Allowed Purposes	✓ Communication ☐ Alert	Expiry Date	31-Aug-2012
Preferred Contact	☐ Preferred Contact		
Marketing Consent	☐ Marketing Consent		
Marketing Consent Start Date		Marketing Consent End Date	

Telephone Details

Country Code		Area Code	
• Number	32577789	Extension	
Service Provider		VOIP Code	

Timing Preferences

DND	☐ DND		
DND Start		DND End	
Weekdays	☐ Weekdays		
From		To	
Weekends	☐ Weekends		
From		To	

5.7.3 Override the product managedBean

Screen customizations could be used to handle a product code which does not serve the necessary functionality and needs to be re-written.

6 Human Task Screen Extension

6.1 Introduction

Human task screen uses internally ADF taskflows except the primary worklist actions. Human screen extensions work similar to ADF screen extensions, where the backing bean and controller classes of human task have the provision for extension, similar to the UI extension as explained in Section 4.1 UI Extension Interface, Section 4.2 Default UI Extension, Section 4.3 UI Extension Executor, and Section 4.4 Extension Configuration. From the perspective of customization, the recommendation is to do customization on the taskflows used internally. For CSS customization, see the below the details.

6.2 Custom CSS Skin

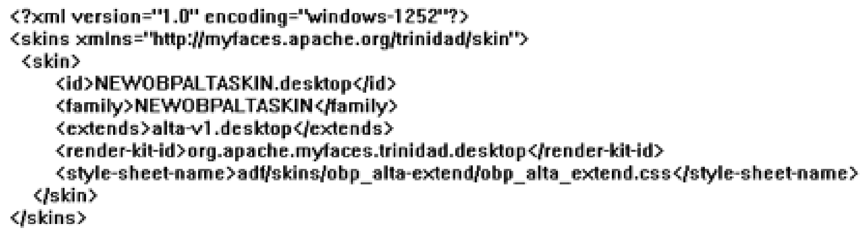
If ADF skin is already available and you just want to change the skin family reference in “trinidad-config.xml”, directly apply the Section 6.2.2 Apply New Skin . If you want to create a new ADF skin and merge with the existing skin families to “trinidad-skins.xml”, start from Section 6.2.1 Create New ADF Skin .

6.2.1 Create New ADF Skin

To create a new ADF skin and package it as library to refer to new skin families files, perform the following steps:

1. Create ADF skin project and skin families as given in https://docs.oracle.com/cd/E16162_01/user.1112/e17456/adfsg_project.htm#ADFSG490:

   Figure 6–1 Sample trinidad-skins.xml

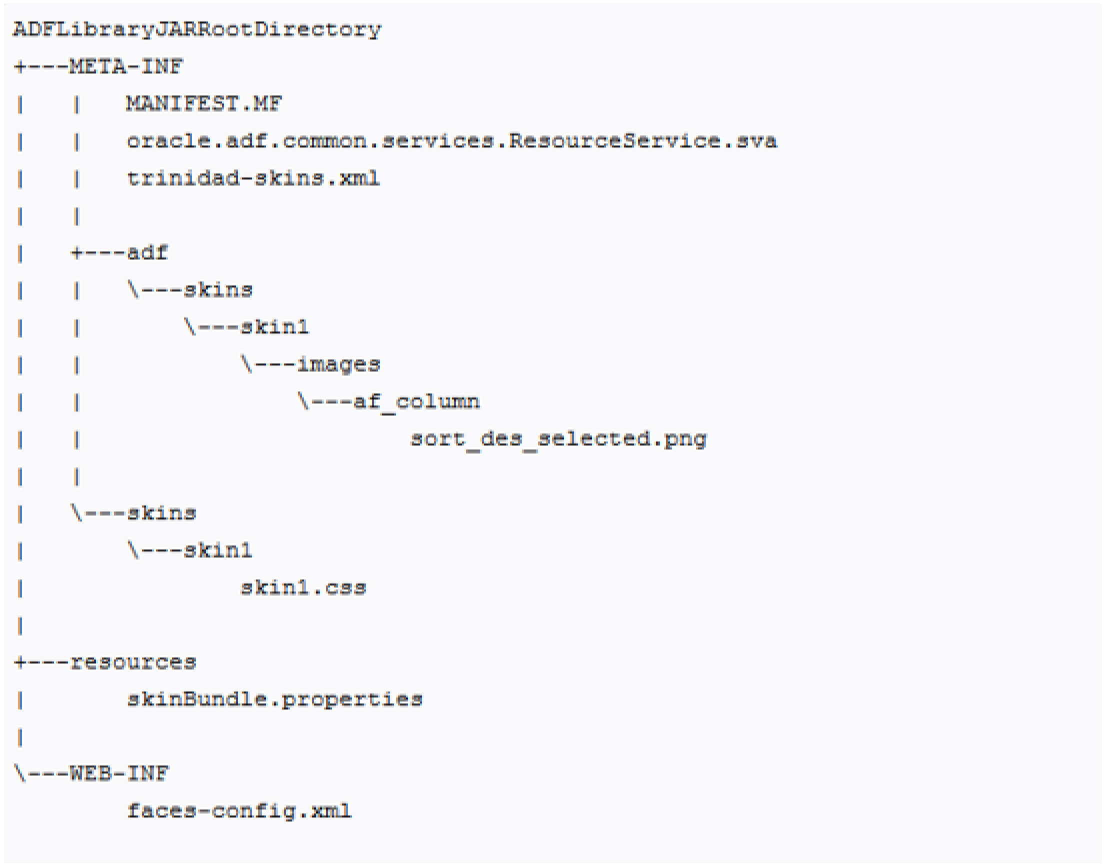


```
<?xml version="1.0" encoding="windows-1252"?>
<skins xmlns="http://myfaces.apache.org/trinidad/skin">
  <skin>
      <id>NEWOBPALTASKIN.desktop</id>
      <family>NEWOBPALTASKIN</family>
      <extends>alta-v1.desktop</extends>
      <render-kit-id>org.apache.myfaces.trinidad.desktop</render-kit-id>
      <style-sheet-name>adf/skins/obp_alta-extend/obp_alta_extend.css</style-sheet-name>
    </skin>
</skins>
```

2. Package the above project as given in https://docs.oracle.com/cd/E16162_01/user.1112/e17456/adfsg_apply.htm#ADFSG404.

   After deployment, the package structure should be as shown in the following screenshot:

Figure 6–2 Package Structure



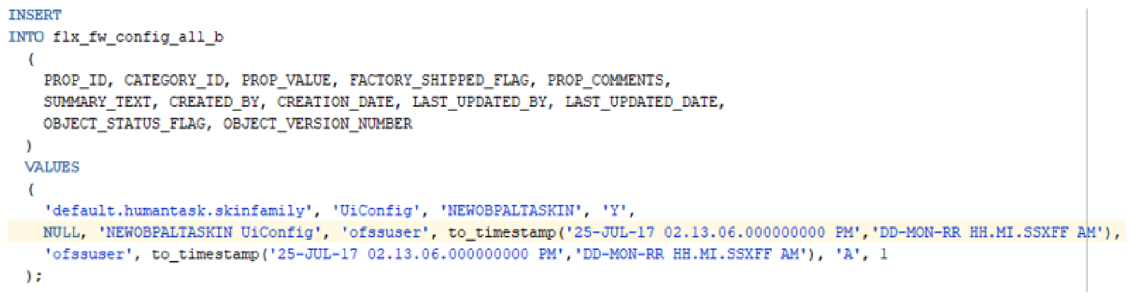
```
ADFLibraryJARRootDirectory
+---META-INF
|    |    MANIFEST.MF
|    |    oracle.adf.common.services.ResourceService.sva
|    |    trinidad-skins.xml
|    |
|    +---adf
|    |    \---skins
|    |        \---skin1
|    |            \---images
|    |                \---af_column
|    |                        sort_des_selected.png
|    |
|    \---skins
|        \---skin1
|                skin1.css
|
+---resources
|       skinBundle.properties
|
\---WEB-INF
        faces-config.xml
```

3. Deploy or add the deployed jar to obp ui library references.
4. Reference implementation can be provided on request.

6.2.2 Apply New Skin

- **Option #1:** The skin id can be configured with the help of property 'default.humantask.skinfamily' in seed data available as part of FLX_FW_CONFIG_ALL_B.

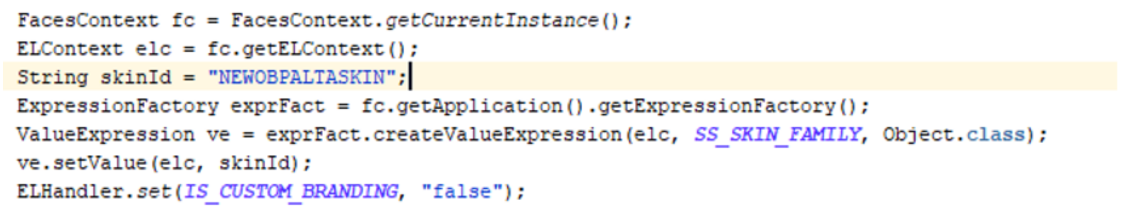
Figure 6–3 Sample Data



```
INSERT
INTO flx_fw_config_all_b
  (
    PROP_ID, CATEGORY_ID, PROP_VALUE, FACTORY_SHIPPED_FLAG, PROP_COMMENTS,
    SUMMARY_TEXT, CREATED_BY, CREATION_DATE, LAST_UPDATED_BY, LAST_UPDATED_DATE,
    OBJECT_STATUS_FLAG, OBJECT_VERSION_NUMBER
  )
  VALUES
  (
    'default.humantask.skinfamily', 'UiConfig', 'NEWOBPALTASKIN', 'Y',
    NULL, 'NEWOBPALTASKIN UiConfig', 'ofssuser', to_timestamp('25-JUL-17 02.13.06.000000000 PM','DD-MON-RR HH.MI.SSXFF AM'),
    'ofssuser', to_timestamp('25-JUL-17 02.13.06.000000000 PM','DD-MON-RR HH.MI.SSXFF AM'), 'A', 1
  );
```

- **Option #2:** Over and above the above the option 1, the extension can be implemented for “postCustomBranding” operation of CommonTaskFormPagePhaseListenerImpl [extension class: com.ofss.fc.workflow.ui.common.ext.ICommonTaskFormPagePhaseListenerImplUIExt]

Figure 6–4 Sample Implementation



```
FacesContext fc = FacesContext.getCurrentInstance();
ELContext elc = fc.getELContext();
String skinId = "NEWOBPALTASKIN";
ExpressionFactory exprFact = fc.getApplication().getExpressionFactory();
ValueExpression ve = exprFact.createValueExpression(elc, SS_SKIN_FAMILY, Object.class);
ve.setValue(elc, skinId);
ELHandler.set(IS_CUSTOM_BRANDING, "false");
```

7 Receipt Printing

OBP has many transaction screens in different modules where it is desired to print the receipt with different details about the transaction. This functionality provides the print receipt button on the top right corner of the screen which gets enabled on the completion of the transaction and can be used for printing of receipt of the transaction details.

For example, if the customer is funding his term deposit account, the print receipt option will print the receipt with the details like Payin Amount, Deposit Term etc at the end of the transaction. The steps to configure this option in the OBP application are given in the following section.

7.1 Prerequisite

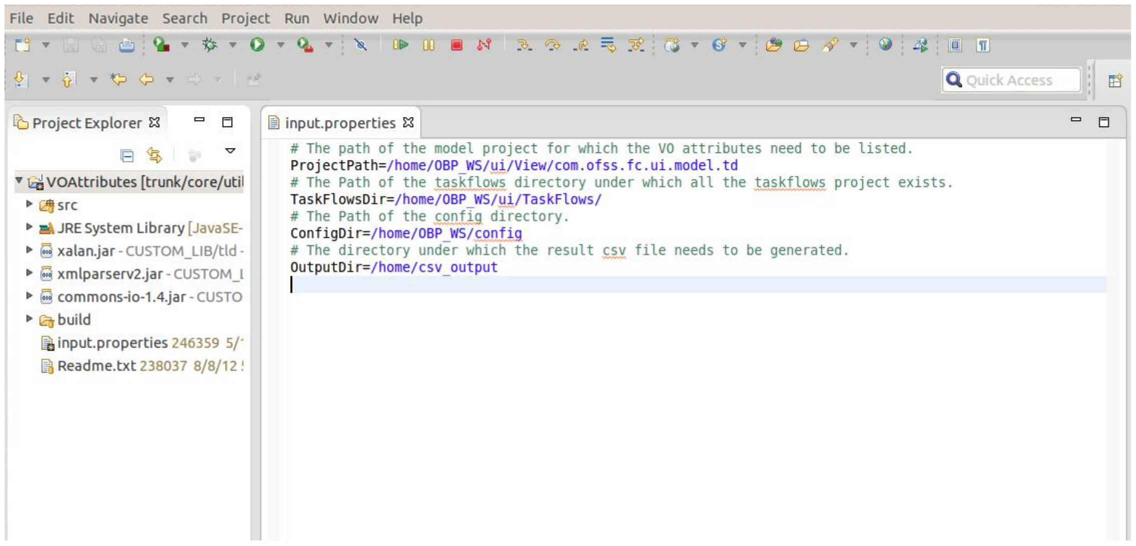
Following are the prerequisites for receipt printing.

7.1.1 Identify Node Element for Attributes in Print Receipt Template

The list of all the elements that are present in the particular task code screen and need to be displayed in the printed receipt can be identified with the help of the VO object utility. This utility helps in identifying all the node elements which are available on the screen and can be used in the print receipt template. This utility VOObjectUtility can be used to generate the data required for the functionality to work.

Once the utility is imported in the workspace, the input.properties file needs to be updated with the location of module's UI, location of task flow directory, location of config directory and the output directory where you want the output of the utility.

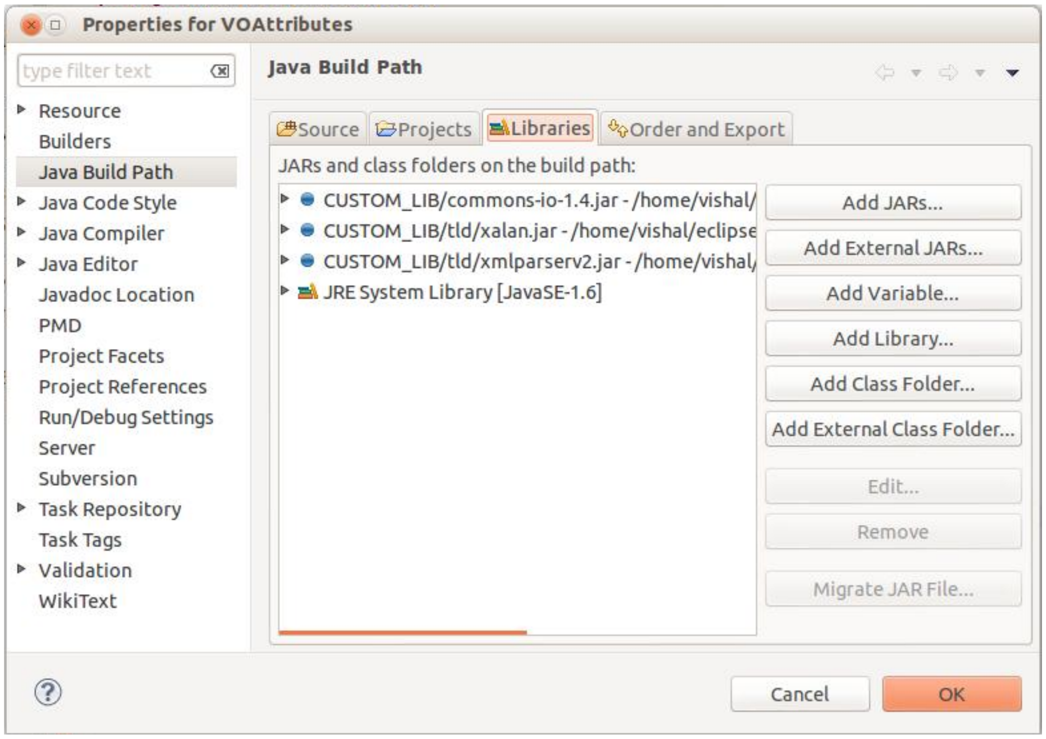
Figure 7–1 Input Property Files


File Edit Navigate Search Project Run Window Help
Quick Access
Project Explorer
VOAttributes [trunk/core/util
src
JRE System Library [JavaSE-
xalan.jar - CUSTOM_LIB/tld -
xmlparserv2.jar - CUSTOM_L
commons-io-1.4.jar - CUSTO
build
input.properties 246359 5/
Readme.txt 238037 8/8/12
input.properties

```
# The path of the model project for which the VO attributes need to be listed.
ProjectPath=/home/OBP_WS/ui/View/com.ofss.fc.ui.model.td
# The Path of the taskflows directory under which all the taskflows project exists.
TaskFlowsDir=/home/OBP_WS/ui/TaskFlows/
# The Path of the config directory.
ConfigDir=/home/OBP_WS/config
# The directory under which the result csv file needs to be generated.
OutputDir=/home/csv_output
```

In the build path of the utility, three libraries (commons-io, xalan and xmlparserv2) need to be added as they are required for execution of the utility.

Figure 7–2 Build Path of Utility

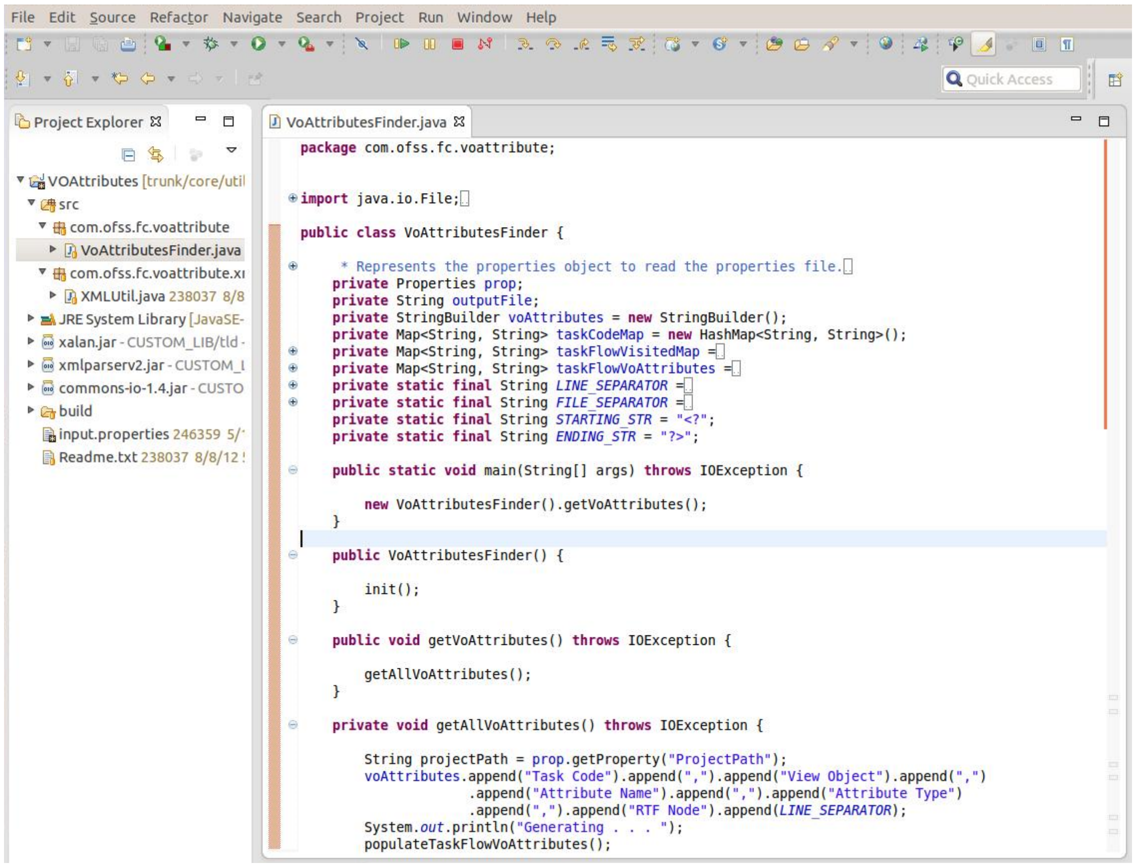


JARs and class folders on the build path:

- CUSTOM_LIB/commons-io-1.4.jar - /home/vishal/
- CUSTOM_LIB/tld/xalan.jar - /home/vishal/eclipse
- CUSTOM_LIB/tld/xmlparserv2.jar - /home/vishal/
- JRE System Library [JavaSE-1.6]

Then the main method of the VOAttributesFinder.java class in the utility is executed.

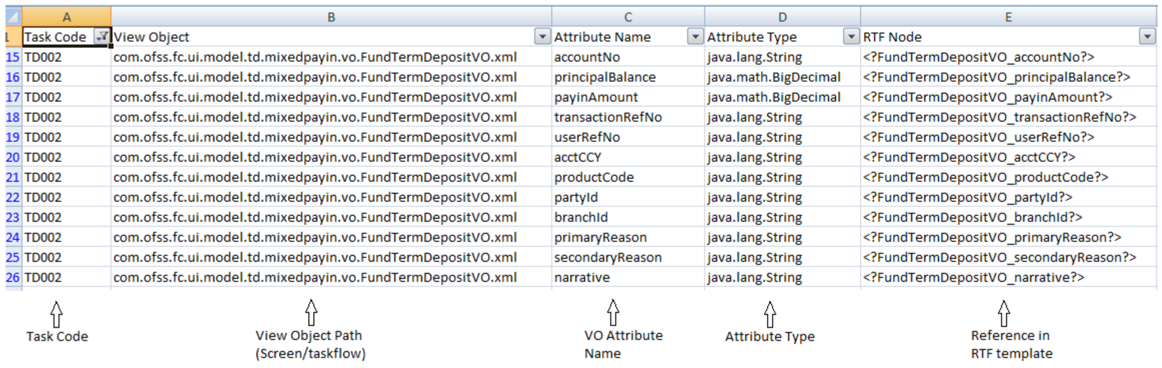
Figure 7–3 Utility Execution


File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Project Explorer
VoAttributesFinder.java
VOAttributes [trunk/core/util
src
com.ofss.fc.voattribute
VoAttributesFinder.java
com.ofss.fc.voattribute.xı
XMLUtil.java 238037 8/8
JRE System Library [JavaSE-
xalan.jar - CUSTOM_LIB/tld -
xmlparserv2.jar - CUSTOM_l
commons-io-1.4.jar - CUSTO
build
input.properties 246359 5/
Readme.txt 238037 8/8/12

```
package com.ofss.fc.voattribute;

import java.io.File;

public class VoAttributesFinder {

    * Represents the properties object to read the properties file.
    private Properties prop;
    private String outputFile;
    private StringBuilder voAttributes = new StringBuilder();
    private Map<String, String> taskCodeMap = new HashMap<String, String>();
    private Map<String, String> taskFlowVisitedMap =
    private Map<String, String> taskFlowVoAttributes =
    private static final String LINE_SEPARATOR =
    private static final String FILE_SEPARATOR =
    private static final String STARTING_STR = "<?";
    private static final String ENDING_STR = "?>";

    public static void main(String[] args) throws IOException {

        new VoAttributesFinder().getVoAttributes();
    }

    public VoAttributesFinder() {

        init();
    }

    public void getVoAttributes() throws IOException {

        getAllVoAttributes();
    }

    private void getAllVoAttributes() throws IOException {

        String projectPath = prop.getProperty("ProjectPath");
        voAttributes.append("Task Code").append(",").append("View Object").append(",")
                     .append("Attribute Name").append(",").append("Attribute Type")
                     .append(",").append("RTF Node").append(LINE_SEPARATOR);
        System.out.println("Generating . . . ");
        populateTaskFlowVoAttributes();
```

On the execution of the utility, the Excel file is generated. The task codes can be filtered in the Excel file for viewing different RTF node value of different attributes available on the particular screen.

Figure 7–4 Excel Generation



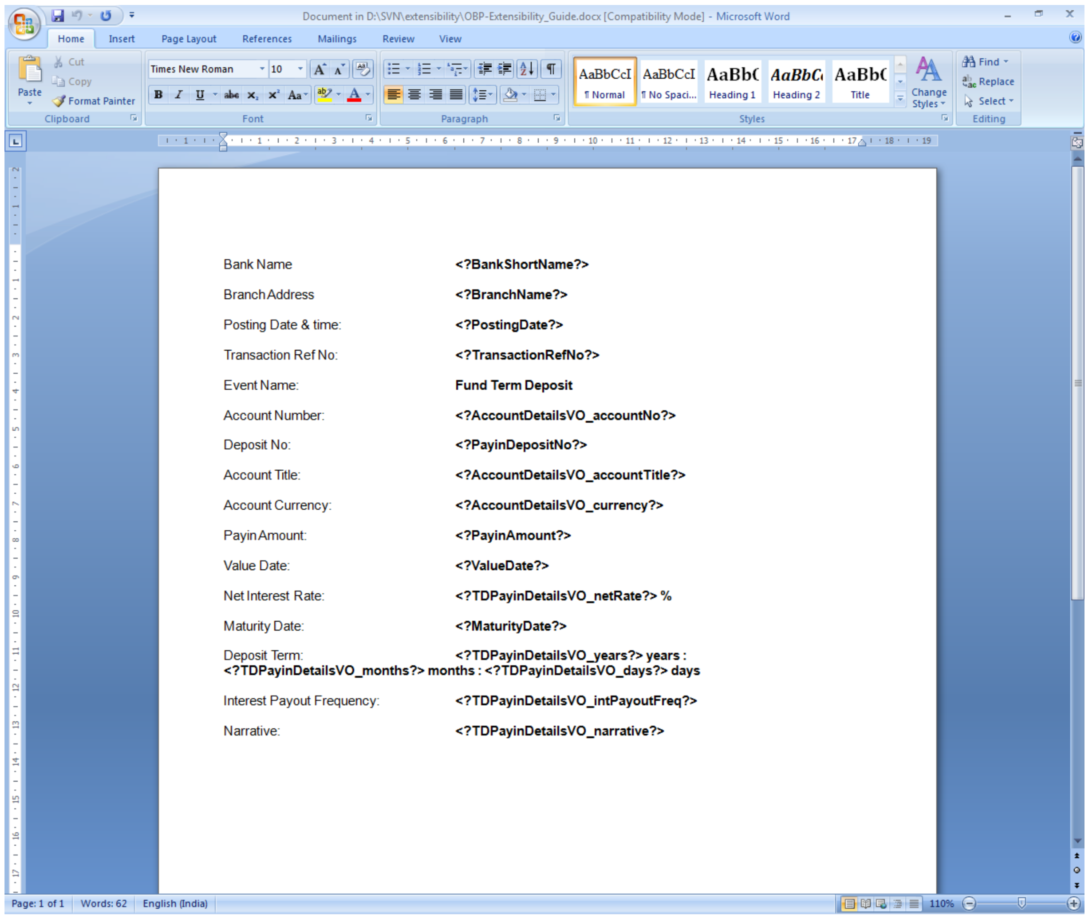
	A	B	C	D	E
1	Task Code	View Object	Attribute Name	Attribute Type	RTF Node
15	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	accountNo	java.lang.String	<?FundTermDepositVO_accountNo?>
16	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	principalBalance	java.math.BigDecimal	<?FundTermDepositVO_principalBalance?>
17	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	payinAmount	java.math.BigDecimal	<?FundTermDepositVO_payinAmount?>
18	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	transactionRefNo	java.lang.String	<?FundTermDepositVO_transactionRefNo?>
19	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	userRefNo	java.lang.String	<?FundTermDepositVO_userRefNo?>
20	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	acctCCY	java.lang.String	<?FundTermDepositVO_acctCCY?>
21	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	productCode	java.lang.String	<?FundTermDepositVO_productCode?>
22	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	partyId	java.lang.String	<?FundTermDepositVO_partyId?>
23	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	branchId	java.lang.String	<?FundTermDepositVO_branchId?>
24	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	primaryReason	java.lang.String	<?FundTermDepositVO_primaryReason?>
25	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	secondaryReason	java.lang.String	<?FundTermDepositVO_secondaryReason?>
26	TD002	com.ofss.fc.ui.model.td.mixedpayin.vo.FundTermDepositVO.xml	narrative	java.lang.String	<?FundTermDepositVO_narrative?>

7.1.2 Receipt Format Template (.rtf)

This template is used for defining the format of the output receipt. Different data elements which need to be shown in the output receipt are mentioned in this RTF report format template. The node will be taken from the above generated Excel file from 'RTF Node' column for specifying the output value in the final output RTF.

The sample rtf template is shown below:

Figure 7–5 Receipt Format Template



Bank Name	**<?BankShortName?>**
Branch Address	**<?BranchName?>**
Posting Date & time:	**<?PostingDate?>**
Transaction Ref No:	**<?TransactionRefNo?>**
Event Name:	**Fund Term Deposit**
Account Number:	**<?AccountDetailsVO_accountNo?>**
Deposit No:	**<?PayinDepositNo?>**
Account Title:	**<?AccountDetailsVO_accountTitle?>**
Account Currency:	**<?AccountDetailsVO_currency?>**
Payin Amount:	**<?PayinAmount?>**
Value Date:	**<?ValueDate?>**
Net Interest Rate:	**<?TDPayinDetailsVO_netRate?> %**
Maturity Date:	**<?MaturityDate?>**
Deposit Term:	**<?TDPayinDetailsVO_years?> years : <?TDPayinDetailsVO_months?> months : <?TDPayinDetailsVO_days?> days**
Interest Payout Frequency:	**<?TDPayinDetailsVO_intPayoutFreq?>**
Narrative:	**<?TDPayinDetailsVO_narrative?>**

7.2 Configuration

This section describes the configuration details.

7.2.1 Parameter Configuration in the BROPConfig.properties

Following configuration parameters are required to be set in the BROPConfig.properties file.

- receipt.print.copy: Set to 'S' (default) if Single receipt is required. Else, set to 'M' for multiple receipts. The receipt will be stored in current posting date 'month/date' folder structure.
- receipt.base.in.location: Location for the RTF templates, which is configured as 'config\receipt\basein\template\' structure on the UI server. (For RTF development purpose this location will also have the XML generated while processing receipt.)
- receipt.base.out.location: Location for generated receipt, which is configured as 'config\receipt\baseout\' structure on the UI server.
- ui.service.url : UI URL http://IP:port format.

7.2.2 Configuration in the ReceiptPrintReports.properties

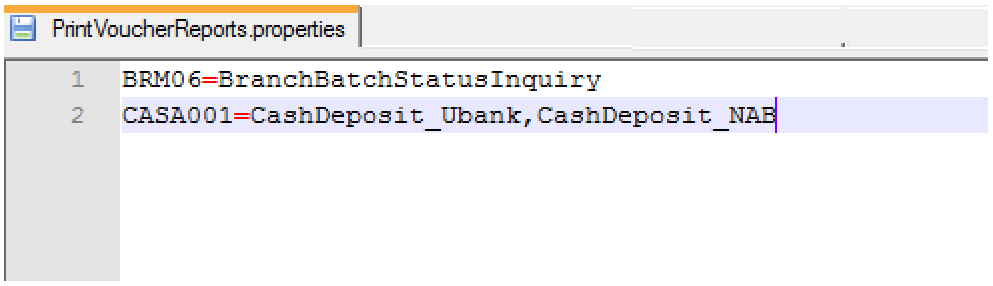
This file contains the key value pair of the Task Code of the screen and the corresponding template names, comma separated if more than 1 template is referred by screen.

TaskCode=RTF Filename

Where TaskCode: task code of screen for which receipt print will be enabled and RTF Filename: name of the RTF template which will be used for the creation of the output with the same filename.

For example, TD002=FundTermDeposit

Figure 7–6 Receipt Print Reports



```
BRM06=BranchBatchStatusInquiry
CASA001=CashDeposit_Ubank,CashDeposit_NAB
```

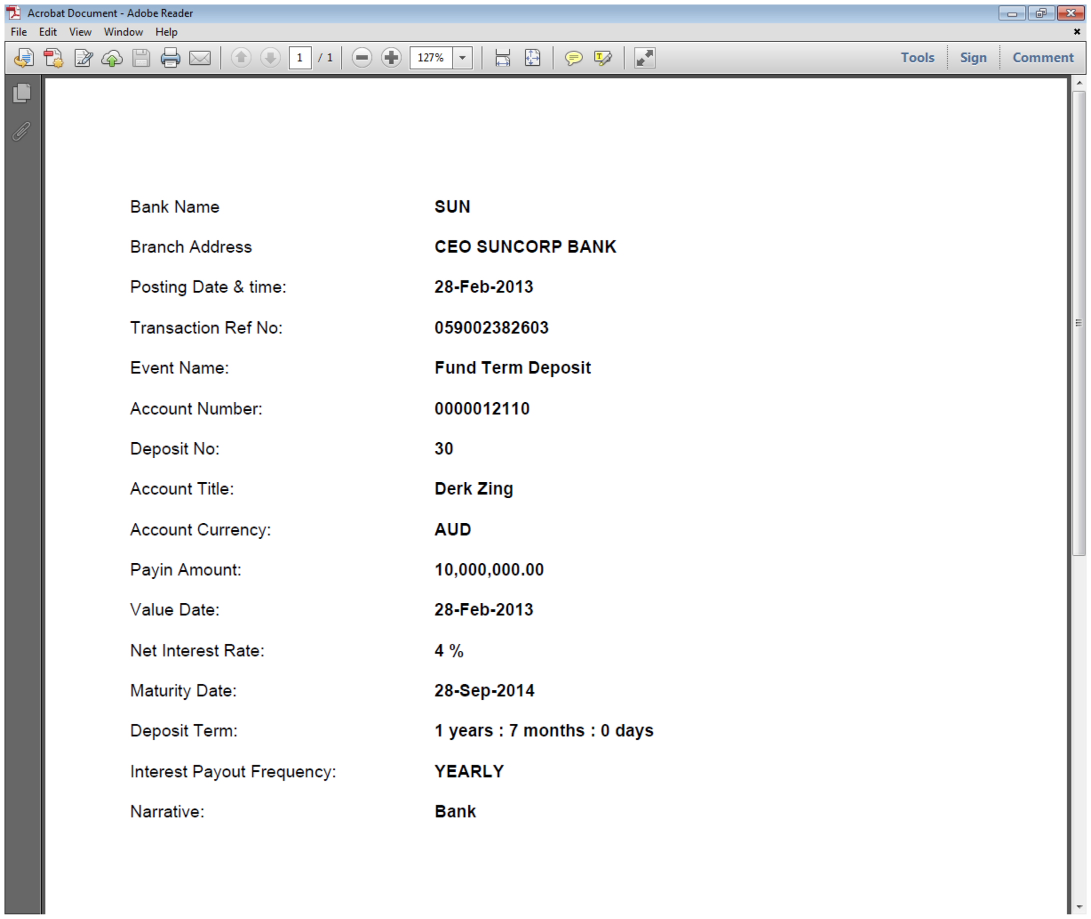
7.3 Implementation

The implementation for the print receipt functionality is explained in the following steps:

1. Once the screen is opened, Template checks '*ReceiptPrintReports.properties*' file if the Task code of the opened screen is present in the property file. The 'Receipt Print' button will be rendered in a disabled state.
2. On successful completion of transaction (successful Ok click), Receipt Print button gets enabled.
3. On click of Receipt Print button, all the VO's on current screen are fetched and created as a XML with data (for RTF development reference, this XML is not deleted at the moment but on environments these will be deleted). The RTF and XML merge up to create and open the receipt in the pdf format.
4. Receipt will be stored with the file name as <Logged in userId_TemplateName>

The sample output receipt in the PDF form is shown below:

Figure 7–7 Sample of Print Receipt



Bank Name	SUN
Branch Address	CEO SUNCORP BANK
Posting Date & time:	28-Feb-2013
Transaction Ref No:	059002382603
Event Name:	Fund Term Deposit
Account Number:	0000012110
Deposit No:	30
Account Title:	Derk Zing
Account Currency:	AUD
Payin Amount:	10,000,000.00
Value Date:	28-Feb-2013
Net Interest Rate:	4 %
Maturity Date:	28-Sep-2014
Deposit Term:	1 years : 7 months : 0 days
Interest Payout Frequency:	YEARLY
Narrative:	Bank

7.3.1 Default Nodes

As per the functional specification requirement, some default nodes are already added in the generated XML. The list of those nodes are as follows:

- BankCode
- BankShortName
- BranchName
- PostingDate
- UserName
- BankAddress
- BranchAddress
- LocalDateTimeText

7.4 Special Scenarios

There are some cases, where some of the attributes are not available in the VOs of the screen and the value needs to be picked from the response of the transaction. There are also some data values which need to be formatted first and then published in the PDF.

These values can be added to the pageFlowScope Map variable 'receiptPrintOtherDetailsMap'.

The population of those values needs to be done in the Backing Bean, after getting the response of the transaction in the following manner:

```
MessageHandler.addMessage(payinResponse.getStatus());
receiptDetails.put("TransactionRefNo",payinResponse.getStatus
().getInternalReferenceNumber());
SimpleDateFormat receiptTimeFormat = new SimpleDateFormat("hh:mm:ss
a");
SimpleDateFormat receiptDateFormat = new SimpleDateFormat("dd-MMM-
yyyy");
HashMap<String,String> receiptDetails = new HashMap<String, String>
();
Date date=new Date(getSessionContext().getLocalDateTimeText());
receiptDetails.put("PostingTime", receiptTimeFormat.format
(date.getSQLTimestamp()));
if(payinResponse!=null && payinResponse.getValueDate()!=null) {
receiptDetails.put("ValueDate",receiptDateFormat.format
(payinResponse.getValueDate().getSqlDate()));
}
ELHandler.set("#{pageFlowScope.receiptPrintOtherDetailsMap}",
receiptDetails);
```

Internally, the functionality adds all the details in map variable, other than VO's data. While receipt printing, template checks the Map variable and if not null, it gets all the key-value from the map and show them in XML which is used later on for generation of receipt.

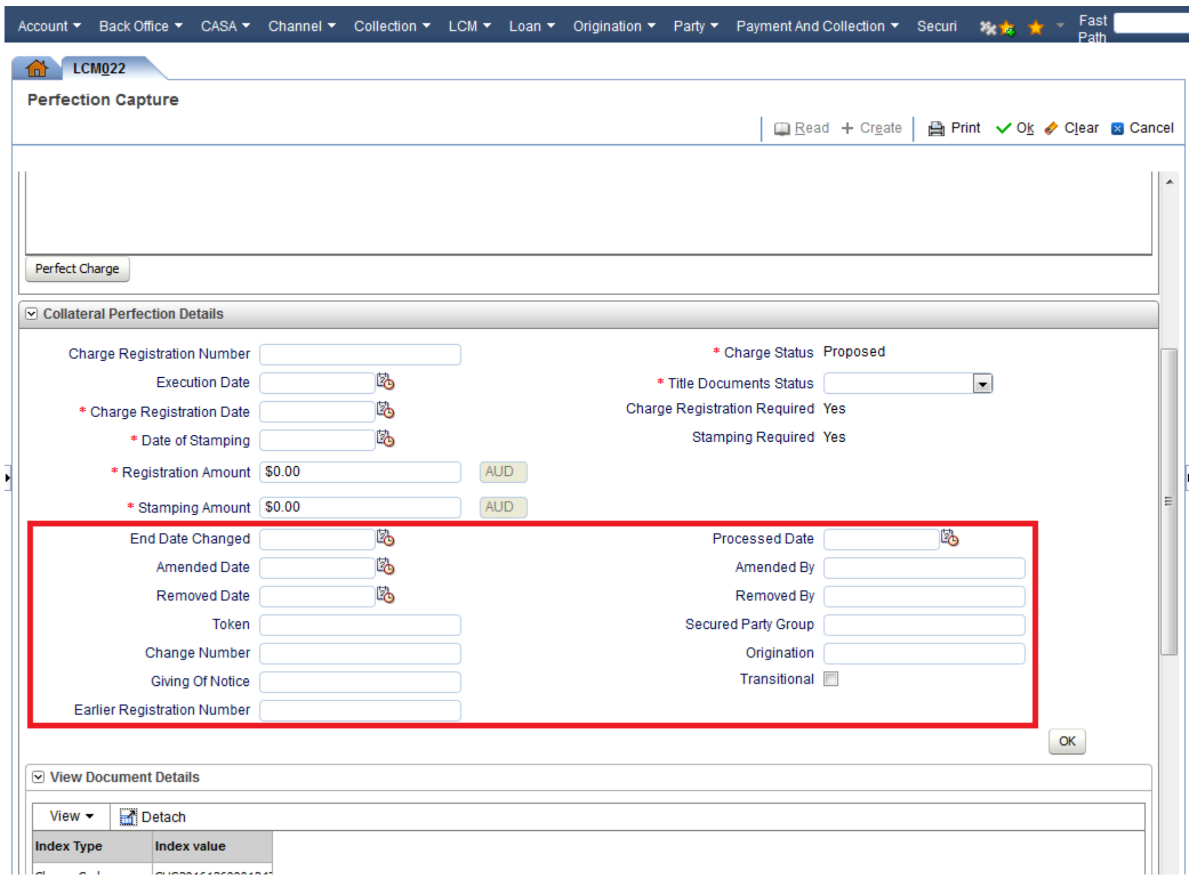
8 Extensibility Usage – OBP Localization Pack

OBP shall be releasing localization pack which ensures an optimized implementation period by adapting the product to different regions by implementing common region specific features pre-built and shipped. Every bank in different regions have different tax laws, different financial policies and so on. The policies in US will be different from those in Australia.

The localization packs leverage OBP extensibility to incorporate regional features and requirements by implementing different extension hooks for host and / or different JDeveloper customization functionalities for UI layer. This section presents a use case from OBP localization pack as implemented using the extensibility guidelines as a sample which can be referred to and followed as a guideline. Customization developers can implement bank's specific requirements on similar lines.

For example, in LCM022 'Perfection Capture' screen, the details section is shown with the additional fields which are defined for a particular location.

Figure 8–1 Perfection Capture Screen



Perfection Capture

Collateral Perfection Details

Charge Registration Number		* Charge Status	Proposed
Execution Date		* Title Documents Status	
* Charge Registration Date		Charge Registration Required	Yes
* Date of Stamping		Stamping Required	Yes
* Registration Amount	$0.00 AUD		
* Stamping Amount	$0.00 AUD		
End Date Changed		Processed Date	
Amended Date		Amended By	
Removed Date		Removed By	
Token		Secured Party Group	
Change Number		Origination	
Giving Of Notice		Transitional	☐
Earlier Registration Number			

View Document Details

Index Type	Index value

8.1 Localization Implementation Architectural Change

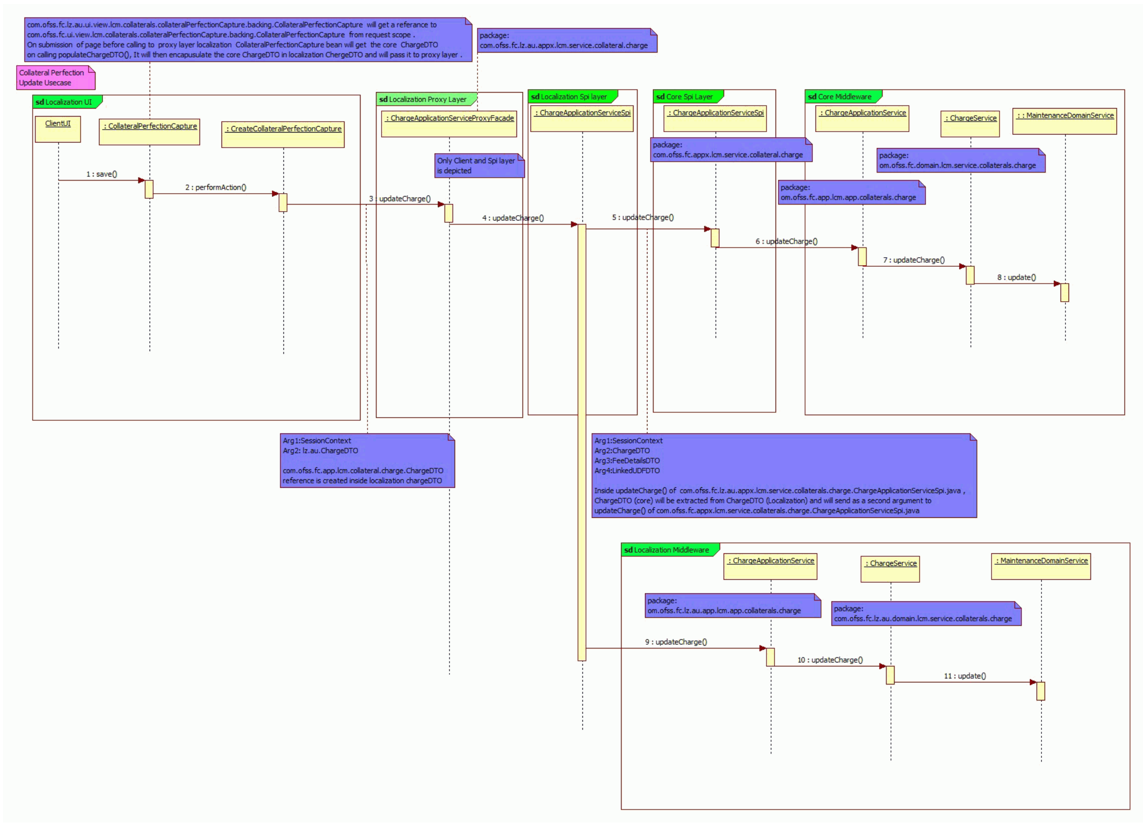
Architecturally, the following points are considered:

- Localization package will be over and above the product.
- Customization packages will be over the Localization and the Product.
- Any changes done for Localization should ensure that future product changes as well as customization changes will work seamlessly without any impact.

The additional fields which get identified and developed as part of localization requirements are in its own project, package, configuration, constant files and tables.

For example, the typical flow of the above mentioned perfection attributes added as part of localization requirement is shown below:

Figure 8–2 Localization Implementation Architectural Change


Collateral Perfection Update Usecase
sd Localization UI
ClientUI
: CollateralPerfectionCapture
: CreateCollateralPerfectionCapture
1 : save()
2 : performAction()
3 : updateCharge()
sd Localization Proxy Layer
: ChargeApplicationServiceProxyFacade
Only Client and Spi layer is depicted
4 : updateCharge()
sd Localization Spi layer
: ChargeApplicationServiceSpi
5 : updateCharge()
sd Core Spi Layer
: ChargeApplicationServiceSpi
package: com.ofss.fc.appx.lcm.service.collateral.charge
6 : updateCharge()
sd Core Middleware
: ChargeApplicationService
: ChargeService
: : MaintenanceDomainService
package: om.ofss.fc.domain.lcm.service.collaterals.charge
package: om.ofss.fc.app.lcm.app.collaterals.charge
7 : updateCharge()
8 : update()
package: com.ofss.fc.lz.au.appx.lcm.service.collateral.charge
sd Localization Middleware
: ChargeApplicationService
: ChargeService
: MaintenanceDomainService
package: om.ofss.fc.lz.au.app.lcm.app.collaterals.charge
package: com.ofss.fc.lz.au.domain.lcm.service.collaterals.charge
9 : updateCharge()
10 : updateCharge()
11 : update()

com.ofss.fc.lz.au.ui.view.lcm.collaterals.collateralPerfectionCapture.backing.CollateralPerfectionCapture will get a referance to com.ofss.fc.ui.view.lcm.collaterals.collateralPerfectionCapture.backing.CollateralPerfectionCapture from request scope .
On submission of page before calling to proxy layer localization CollateralPerfectionCapture bean will get the core ChargeDTO on calling populateChargeDTO(), It will then encapusulate the core ChargeDTO in localization ChergeDTO and will pass it to proxy layer .

Arg1:SessionContext
Arg2: lz.au.ChargeDTO

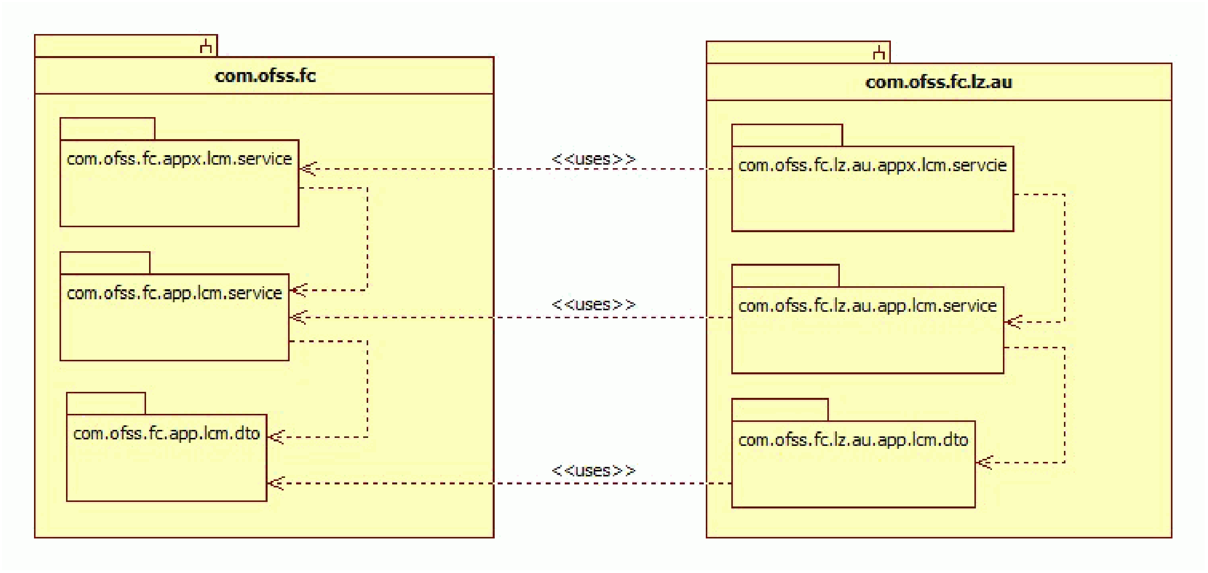
com.ofss.fc.app.lcm.collateral.charge.ChargeDTO reference is created inside localization chargeDTO

Arg1:SessionContext
Arg2:ChargeDTO
Arg3:FeeDetailsDTO
Arg4:LinkedUDFDTO

Inside updateCharge() of com.ofss.fc.lz.au.appx.lcm.service.collaterals.charge.ChargeApplicationServiceSpi.java , ChargeDTO (core) will be extracted from ChargeDTO (Localization) and will send as a second argument to updateCharge() of com.ofss.fc.appx.lcm.service.collaterals.charge.ChargeApplicationServiceSpi.java

The Package structure for the implementation is shown below:

Figure 8–3 Package Structure


com.ofss.fc
com.ofss.fc.appx.lcm.service
com.ofss.fc.app.lcm.service
com.ofss.fc.app.lcm.dto
<<uses>>
<<uses>>
<<uses>>
com.ofss.fc.lz.au
com.ofss.fc.lz.au.appx.lcm.servcie
com.ofss.fc.lz.au.app.lcm.service
com.ofss.fc.lz.au.app.lcm.dto

8.2 Customizing UI Layer

This section explains the customization of UI layer.

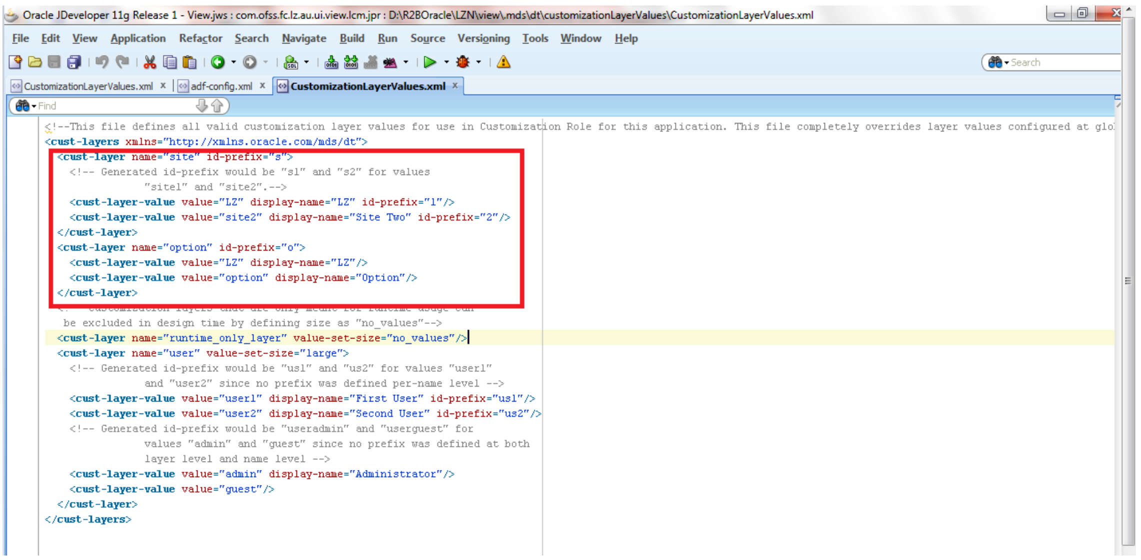
8.2.1 JDeveloper and Project Customization

For the customization of the UI layer, JDeveloper needs to be configured in the customizable mode as explained in the ADF Screen Customization Sections.

The example for the customization of the JDeveloper is described below:

CustomizationLayerValues.xml

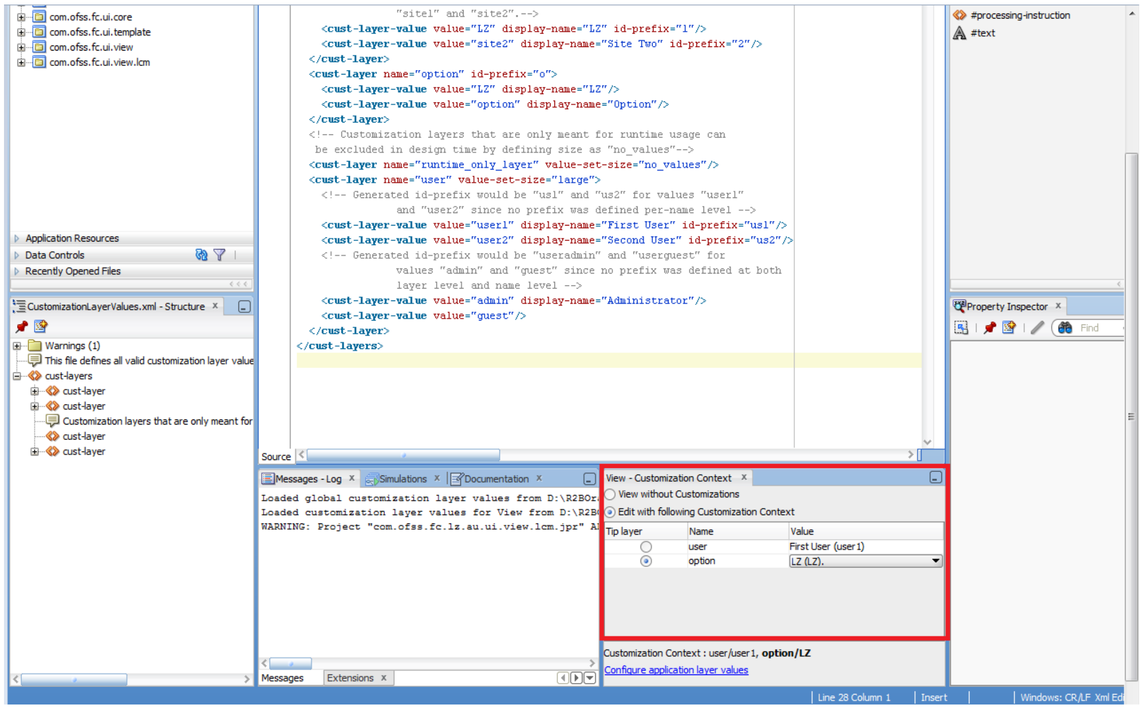
Figure 8–4 Customization of the JDeveloper



Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.lz.au.ui.view.lcm.jpr : D:\R2BOracle\LZN\view\.mds\dt\customizationLayerValues\CustomizationLayerValues.xml

```
<!--This file defines all valid customization layer values for use in Customization Role for this application. This file completely overrides layer values configured at glo
<cust-layers xmlns="http://xmlns.oracle.com/mds/dt">
  <cust-layer name="site" id-prefix="s">
    <!-- Generated id-prefix would be "s1" and "s2" for values
            "site1" and "site2".-->
    <cust-layer-value value="LZ" display-name="LZ" id-prefix="1"/>
    <cust-layer-value value="site2" display-name="Site Two" id-prefix="2"/>
  </cust-layer>
  <cust-layer name="option" id-prefix="o">
    <cust-layer-value value="LZ" display-name="LZ"/>
    <cust-layer-value value="option" display-name="Option"/>
  </cust-layer>
   be excluded in design time by defining size as "no_values"-->
  <cust-layer name="runtime_only_layer" value-set-size="no_values"/>
  <cust-layer name="user" value-set-size="large">
    <!-- Generated id-prefix would be "us1" and "us2" for values "user1"
            and "user2" since no prefix was defined per-name level -->
    <cust-layer-value value="user1" display-name="First User" id-prefix="us1"/>
    <cust-layer-value value="user2" display-name="Second User" id-prefix="us2"/>
    <!-- Generated id-prefix would be "useradmin" and "userguest" for
            values "admin" and "guest" since no prefix was defined at both
            layer level and name level -->
    <cust-layer-value value="admin" display-name="Administrator"/>
    <cust-layer-value value="guest"/>
  </cust-layer>
</cust-layers>
```

Figure 8–5 Customization of the JDeveloper



```
            "site1" and "site2".-->
    <cust-layer-value value="LZ" display-name="LZ" id-prefix="1"/>
    <cust-layer-value value="site2" display-name="Site Two" id-prefix="2"/>
  </cust-layer>
  <cust-layer name="option" id-prefix="o">
    <cust-layer-value value="LZ" display-name="LZ"/>
    <cust-layer-value value="option" display-name="Option"/>
  </cust-layer>
  <!-- Customization layers that are only meant for runtime usage can
   be excluded in design time by defining size as "no_values"-->
  <cust-layer name="runtime_only_layer" value-set-size="no_values"/>
  <cust-layer name="user" value-set-size="large">
    <!-- Generated id-prefix would be "us1" and "us2" for values "user1"
            and "user2" since no prefix was defined per-name level -->
    <cust-layer-value value="user1" display-name="First User" id-prefix="us1"/>
    <cust-layer-value value="user2" display-name="Second User" id-prefix="us2"/>
    <!-- Generated id-prefix would be "useradmin" and "userguest" for
            values "admin" and "guest" since no prefix was defined at both
            layer level and name level -->
    <cust-layer-value value="admin" display-name="Administrator"/>
    <cust-layer-value value="guest"/>
  </cust-layer>
</cust-layers>
```

```
Loaded global customization layer values from D:\R2BOr
Loaded customization layer values for View from D:\R2B
WARNING: Project "com.ofss.fc.lz.au.ui.view.lcm.jpr" A
```

View - Customization Context

( ) View without Customizations
(●) Edit with following Customization Context

Tip layer	Name	Value
○	user	First User (user1)
●	option	LZ (LZ).

Customization Context : user/user1, **option/LZ**

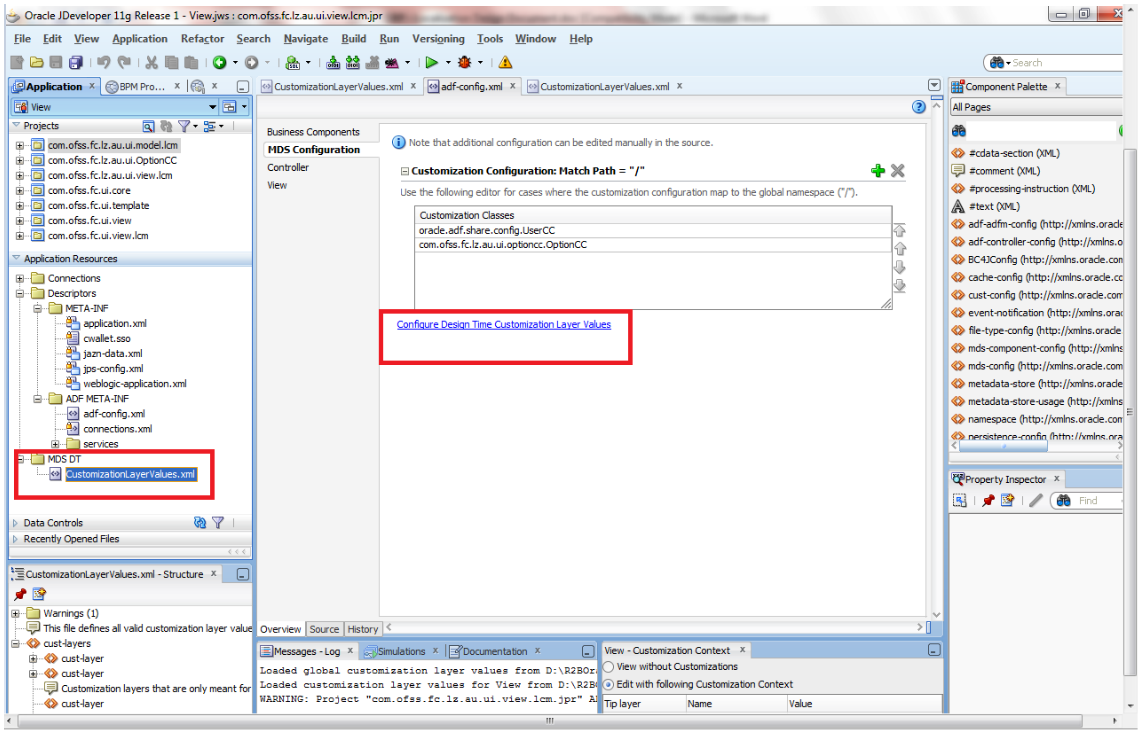
Configure application layer values

adf-config.xml

If the changes are not reflecting, adf-config.xml needs to be opened from the application resources and *Configure Design Time Customization layer values* highlighted in the below image needs to be clicked. It will create a CustomizationLayerValues.xml inside MDS DT folder in application resources. All the content from

<JDEVELOPER_HOME>/jdeveloper/jdev/CustomizationLayerValues.xml needs to be copied to this CustomizationLayerValues.xml. This is to ensure that the changes are reflected at the application level.

Figure 8–6 Configure Design Time Customization layer


Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.lz.au.ui.view.lcm.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Application
BPM Pro...
Projects
com.ofss.fc.lz.au.ui.model.lcm
com.ofss.fc.lz.au.ui.OptionCC
com.ofss.fc.lz.au.ui.view.lcm
com.ofss.fc.ui.core
com.ofss.fc.ui.template
com.ofss.fc.ui.view
com.ofss.fc.ui.view.lcm
Application Resources
Connections
Descriptors
META-INF
application.xml
cwallet.sso
jazn-data.xml
jps-config.xml
weblogic-application.xml
ADF META-INF
adf-config.xml
connections.xml
services
MDS DT
CustomizationLayerValues.xml
Data Controls
Recently Opened Files
CustomizationLayerValues.xml - Structure
Warnings (1)
This file defines all valid customization layer value
cust-layers
cust-layer
cust-layer
Customization layers that are only meant for
cust-layer
CustomizationLayerValues.xml
adf-config.xml
CustomizationLayerValues.xml
Business Components
MDS Configuration
Controller
View
Overview
Source
History
Messages - Log
Simulations
Documentation
Component Palette
All Pages
#cdata-section (XML)
#comment (XML)
#processing-instruction (XML)
#text (XML)
adf-adfm-config (http://xmlns.oracle
adf-controller-config (http://xmlns.o
BC4JConfig (http://xmlns.oracle.com
cache-config (http://xmlns.oracle.co
cust-config (http://xmlns.oracle.com
event-notification (http://xmlns.orac
file-type-config (http://xmlns.oracle
mds-component-config (http://xmlns
mds-config (http://xmlns.oracle.com
metadata-store (http://xmlns.oracle
metadata-store-usage (http://xmlns
namespace (http://xmlns.oracle.com
persistence-config (http://xmlns.ora
Property Inspector
Find
Search

Note that additional configuration can be edited manually in the source.

Customization Configuration: Match Path = "/"

Use the following editor for cases where the customization configuration map to the global namespace ("/").

Customization Classes
oracle.adf.share.config.UserCC
com.ofss.fc.lz.au.ui.optioncc.OptionCC

Configure Design Time Customization Layer Values

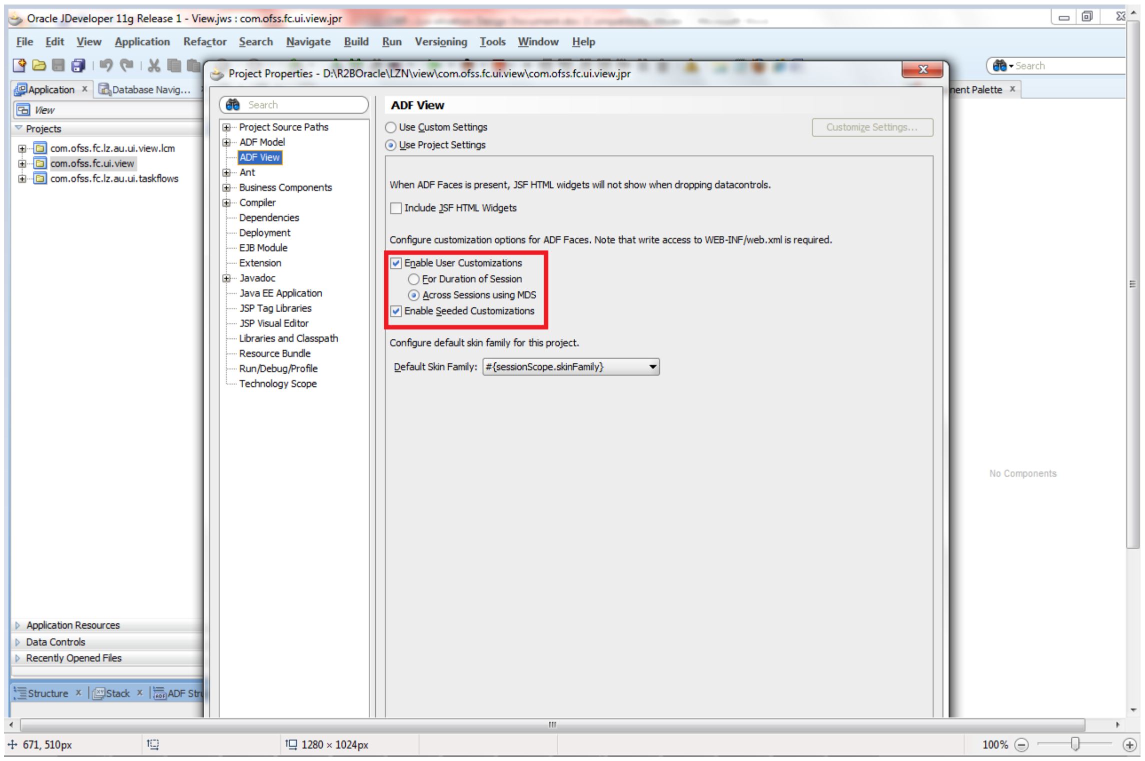
```
Loaded global customization layer values from D:\R2BOr
Loaded customization layer values for View from D:\R2B
WARNING: Project "com.ofss.fc.lz.au.ui.view.lcm.jpr" A
```

View - Customization Context

View without Customizations [✓ Edit with following Customization Context]

Tip layer	Name	Value

Figure 8–7 Enabling Seeded Customization


Project Properties - D:\R2BOracle\LZN\view\com.ofss.fc.ui.view\com.ofss.fc.ui.view.jpr

ADF View

[ ] Use Custom Settings
[● Use Project Settings]

When ADF Faces is present, JSF HTML widgets will not show when dropping datacontrols.

[ ] Include JSF HTML Widgets

Configure customization options for ADF Faces. Note that write access to WEB-INF/web.xml is required.

[✓ Enable User Customizations]
( ) For Duration of Session
[● Across Sessions using MDS]
[✓ Enable Seeded Customizations]

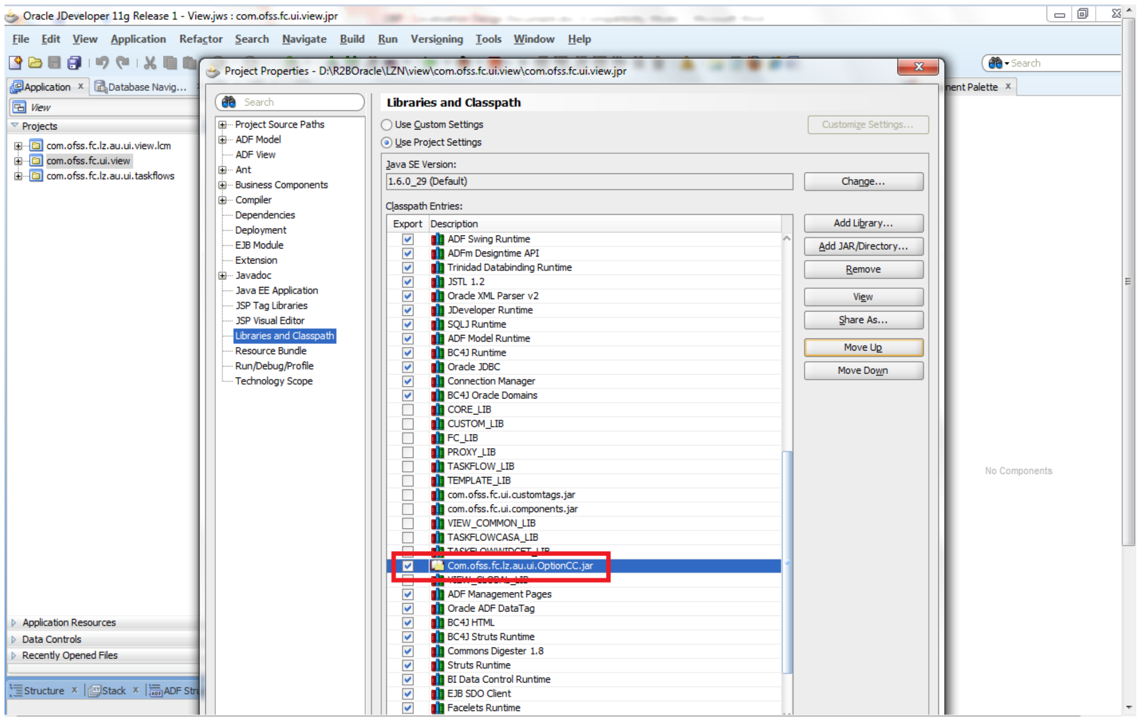
Configure default skin family for this project.

Default Skin Family: #{sessionScope.skinFamily}

Libraries and Classpath

In the "*Libraries and Classpath*" section, the previously deployed *com.ofss.fc.lz.au.ui.OptionCC.jar* containing the customization class then needs to be added.

Figure 8–8 Library and Class Path



Project Properties - D:\R2BOracle\LZN\view\com.ofss.fc.ui.view\com.ofss.fc.ui.view.jpr

Libraries and Classpath

( ) Use Custom Settings
(●) Use Project Settings

Java SE Version: 1.6.0_29 (Default)

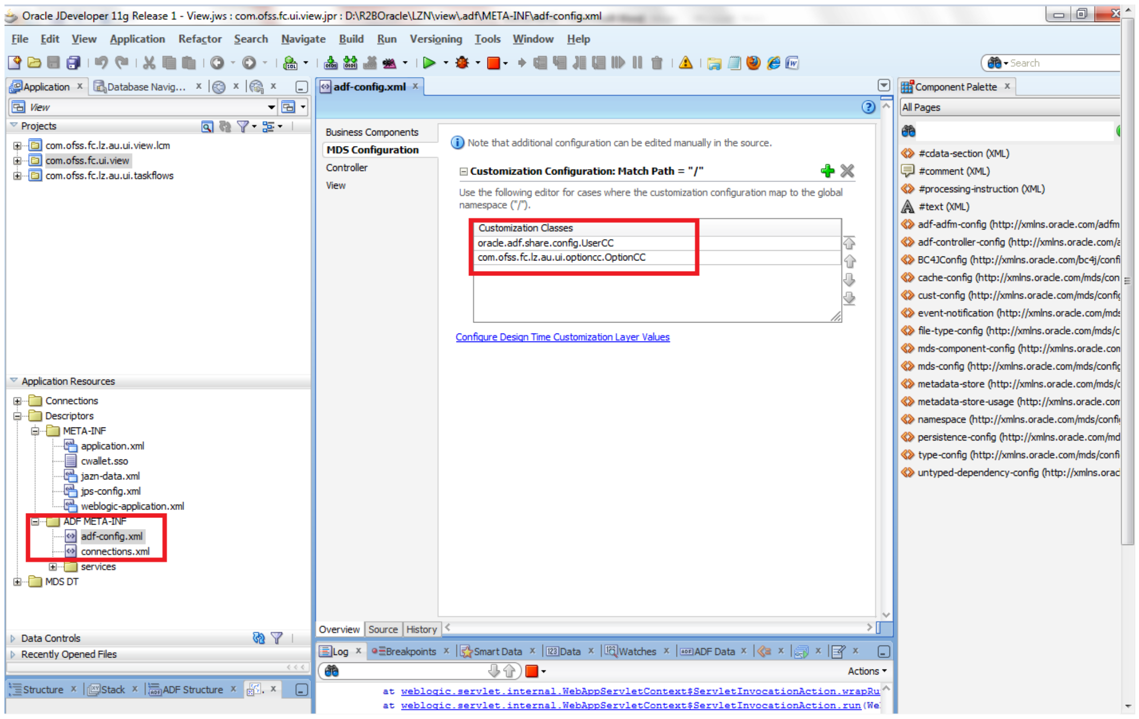
Classpath Entries:

Export	Description
✓	ADF Swing Runtime
✓	ADFm Designtime API
✓	Trinidad Databinding Runtime
✓	JSTL 1.2
✓	Oracle XML Parser v2
✓	JDeveloper Runtime
✓	SQLJ Runtime
✓	ADF Model Runtime
✓	BC4J Runtime
✓	Oracle JDBC
✓	Connection Manager
✓	BC4J Oracle Domains
	CORE_LIB
	CUSTOM_LIB
	FC_LIB
	PROXY_LIB
	TASKFLOW_LIB
	TEMPLATE_LIB
	com.ofss.fc.ui.customtags.jar
	com.ofss.fc.ui.components.jar
	VIEW_COMMON_LIB
	TASKFLOWCASA_LIB
	TASKFLOWWIDGET_LIB
✓	Com.ofss.fc.lz.au.ui.OptionCC.jar
	VIEW_GLOBAL_LIB
✓	ADF Management Pages
✓	Oracle ADF DataTag
✓	BC4J HTML
✓	BC4J Struts Runtime
✓	Commons Digester 1.8
✓	Struts Runtime
✓	BI Data Control Runtime
✓	EJB SDO Client
✓	Facelets Runtime

adf-config.xml

In the *Application Resources* tab, the *adf-config.xml* present in the *Descriptors/ADF META-INF* folder needs to be opened. In the list of *Customization Classes*, all the entries should not be removed and the *com.ofss.fc.lz.au.ui.OptionCC.OptionCC* class to this list needs to be added.

Figure 8–9 MDS Configuration



Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.ui.view.jpr : D:\R2BOracle\LZN\view\.adf\META-INF\adf-config.xml

adf-config.xml

Business Components
MDS Configuration
Controller
View

Note that additional configuration can be edited manually in the source.

Customization Configuration: Match Path = "/"

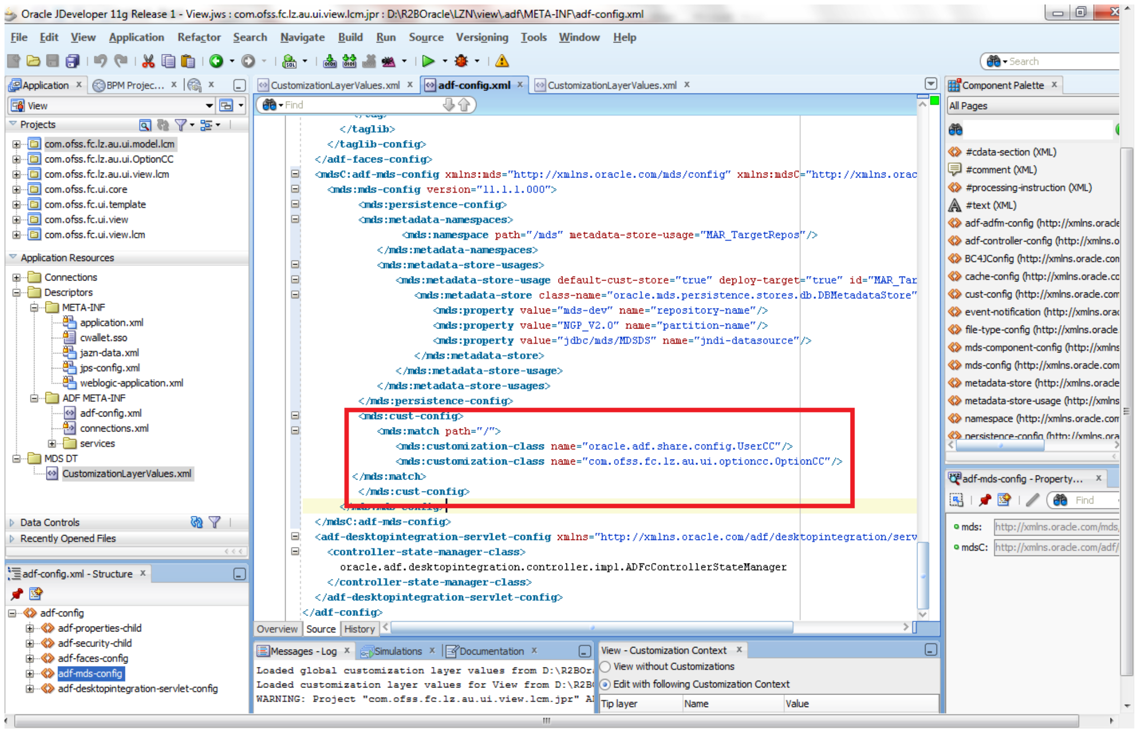
Use the following editor for cases where the customization configuration map to the global namespace ("/").

Customization Classes
oracle.adf.share.config.UserCC
com.ofss.fc.lz.au.ui.optioncc.OptionCC

Configure Design Time Customization Layer Values

Jdeveloper is then restarted and the entry needs to be checked for *com.ofss.fc.lz.au.ui.OptionCC*. If the jar entry is not reflecting, then source needs to be clicked and the entry as highlighted and shown in the below image needs to be manually added.

Figure 8–10 MDS Configuration


Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.lz.au.ui.view.lcm.jpr : D:\R2BOracle\LZN\view\.adf\META-INF\adf-config.xml
File Edit View Application Refactor Search Navigate Build Run Source Versioning Tools Window Help
CustomizationLayerValues.xml
adf-config.xml
Overview Source History
Messages - Log
Simulations
Documentation
View - Customization Context
Component Palette

```
          </taglib>
        </taglib-config>
      </adf-faces-config>
      <mdsC:adf-mds-config xmlns:mds="http://xmlns.oracle.com/mds/config" xmlns:mdsC="http://xmlns.orac
        <mds:mds-config version="11.1.1.000">
            <mds:persistence-config>
            <mds:metadata-namespaces>
                <mds:namespace path="/mds" metadata-store-usage="MAR_TargetRepos"/>
              </mds:metadata-namespaces>
              <mds:metadata-store-usages>
                <mds:metadata-store-usage default-cust-store="true" deploy-target="true" id="MAR_Tar
                  <mds:metadata-store class-name="oracle.mds.persistence.stores.db.DBMetadataStore"
                    <mds:property value="mds-dev" name="repository-name"/>
                    <mds:property value="NGP_V2.0" name="partition-name"/>
                    <mds:property value="jdbc/mds/MDSDS" name="jndi-datasource"/>
                  </mds:metadata-store>
                </mds:metadata-store-usage>
              </mds:metadata-store-usages>
            </mds:persistence-config>
            <mds:cust-config>
              <mds:match path="/">
                <mds:customization-class name="oracle.adf.share.config.UserCC"/>
                <mds:customization-class name="com.ofss.fc.lz.au.ui.optioncc.OptionCC"/>
           </mds:match>
            </mds:cust-config>
          </mds:mds-config>
      </mdsC:adf-mds-config>
      <adf-desktopintegration-servlet-config xmlns="http://xmlns.oracle.com/adf/desktopintegration/serv
        <controller-state-manager-class>
          oracle.adf.desktopintegration.controller.impl.ADFcControllerStateManager
        </controller-state-manager-class>
      </adf-desktopintegration-servlet-config>
    </adf-config>
```

```
Loaded global customization layer values from D:\R2BOr
Loaded customization layer values for View from D:\R2B
WARNING: Project "com.ofss.fc.lz.au.ui.view.lcm.jpr" A
```

View without Customizations [✓ Edit with following Customization Context]

Tip layer	Name	Value

8.2.2 Generic Project Creation

After creating the Customization Layer, Customization Class and enabling the application for Seeded Customizations, the next step is to create a project which will hold the customizations for the application. Generic project is then created with the following technologies:

- ADF Business Components
- Java
- JSF
- JSP and Servlets

Following jars must then be added to the Project Properties and in the classpath:

- Customization class JAR (com.ofss.fc.lz.au.ui.OptionCC.jar)
- The project JAR which contains the screen / component to be customized. For example, if you want to customize the Collateral Perfection Capture screen, the related project JAR is *com.ofss.fc.ui.view.lcm.jar.*
- All the dependent JARS / libraries for the project needs to added.
- Finally newly created project (example: *'com.ofss.fc.lz.au.view.lcm'*) needs to be enabled for Seeded Customizations.

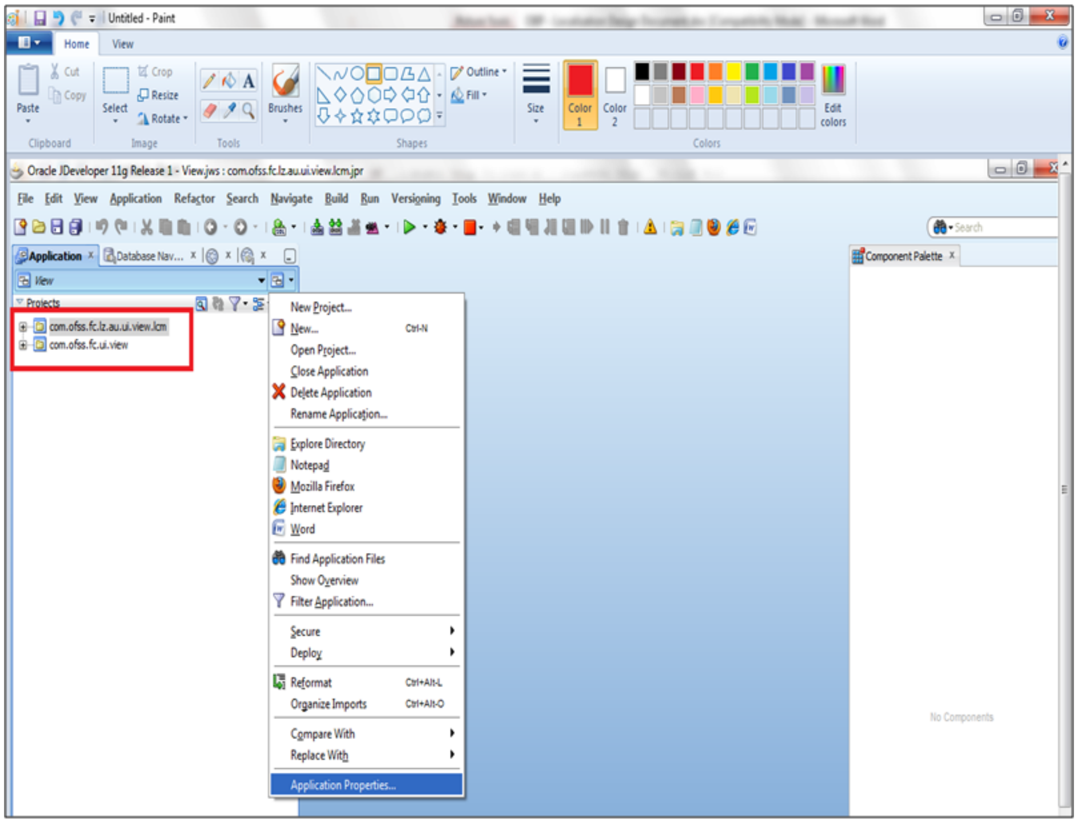
8.2.3 MAR Creation

After implementing customizations on objects from an ADF library, the customization metadata is stored by default in a subdirectory of the project called *libraryCustomizations*. Although ADF library customizations at the project level is created and merged together during packaging to be available at the application level at runtime. Essentially, ADF libraries are JARs that are added at the project level, which map to library customizations being created at the project level. However, although projects map to web applications at runtime, the MAR (which contains the library customizations) is at the EAR level, so the library customizations are seen from all web applications.

Therefore, an ADF library artifact are customized in only one place in an application for a given customization context (customization layer and layer value). Customizing the same library content in different projects for the same customization context would result in duplication in MAR packaging. To avoid duplicates that would cause packaging to fail, customizations are implemented for a given library in only one project in your application.

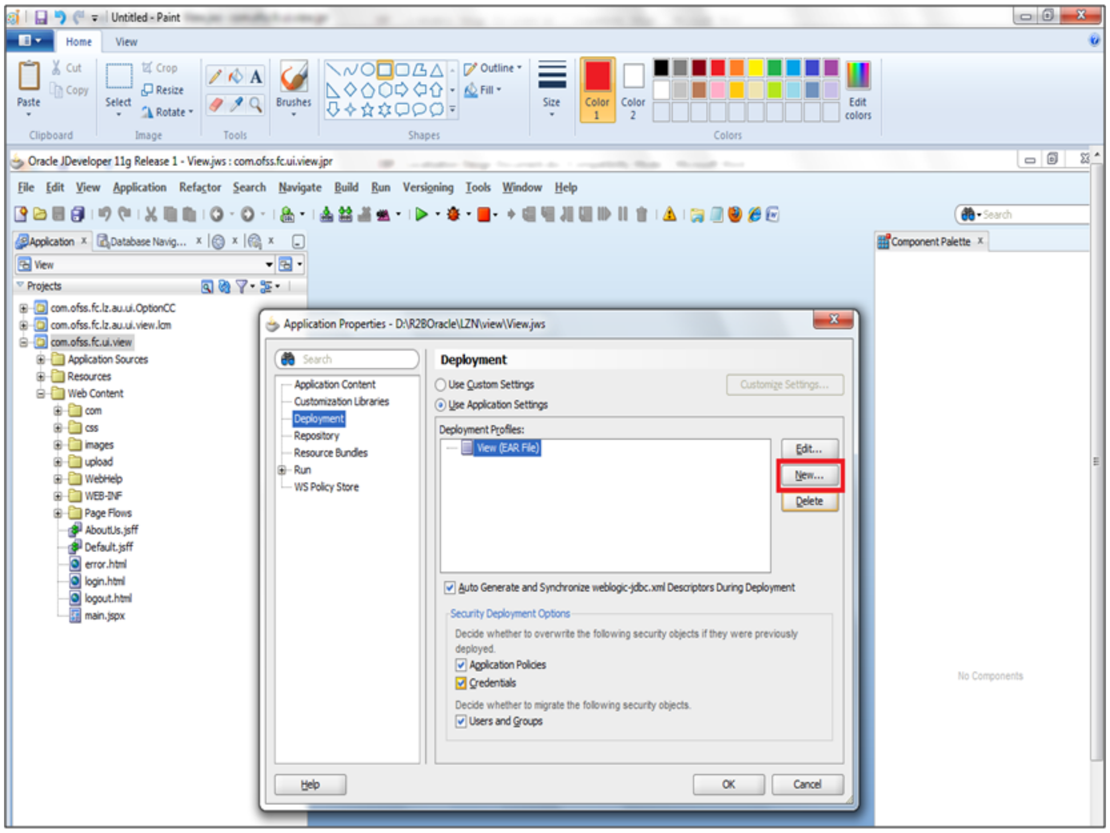
Step 1
Select the Application Properties.

Figure 8–11 MAR Creation



Step 2
Import com.ofss.fc.lz.au.ui.view.lcm project into application. Click Application Menu and select Application Properties.

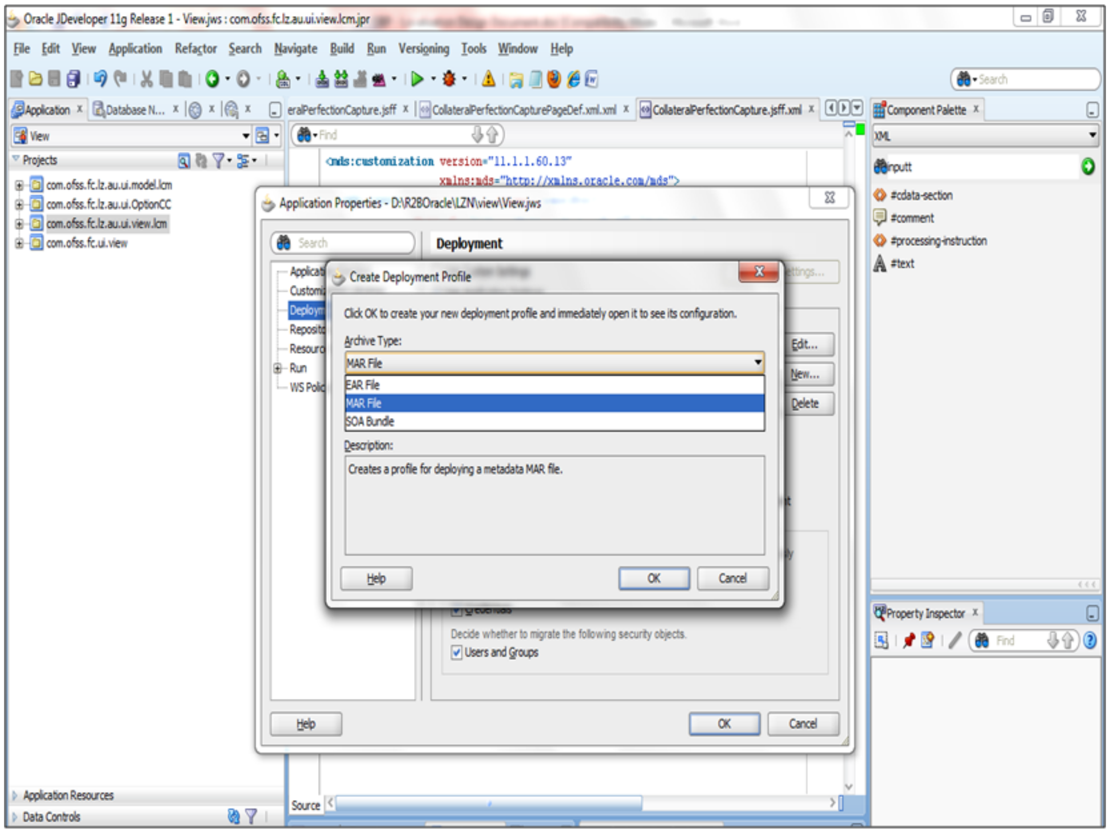
Figure 8–12 MAR Creation - Application Properties


Application Properties - D:\R2BOracle\LZN\view\View.jws
Search
Application Content
Customization Libraries
Deployment
Repository
Resource Bundles
Run
WS Policy Store
Deployment
Use Custom Settings
Customize Settings...
Use Application Settings
Deployment Profiles:
View (EAR File)
Edit...
New...
Delete
Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment
Security Deployment Options
Decide whether to overwrite the following security objects if they were previously deployed.
Application Policies
Credentials
Decide whether to migrate the following security objects.
Users and Groups
Help
OK
Cancel

Step 3

Select Deployment and click **New.**

Figure 8–13 MAR Creation - Create Deployment Profile



```
<mds:customization version="11.1.1.60.13"
                   xmlns:mds="http://xmlns.oracle.com/mds">
```

Create Deployment Profile

Click OK to create your new deployment profile and immediately open it to see its configuration.

Archive Type: MAR File

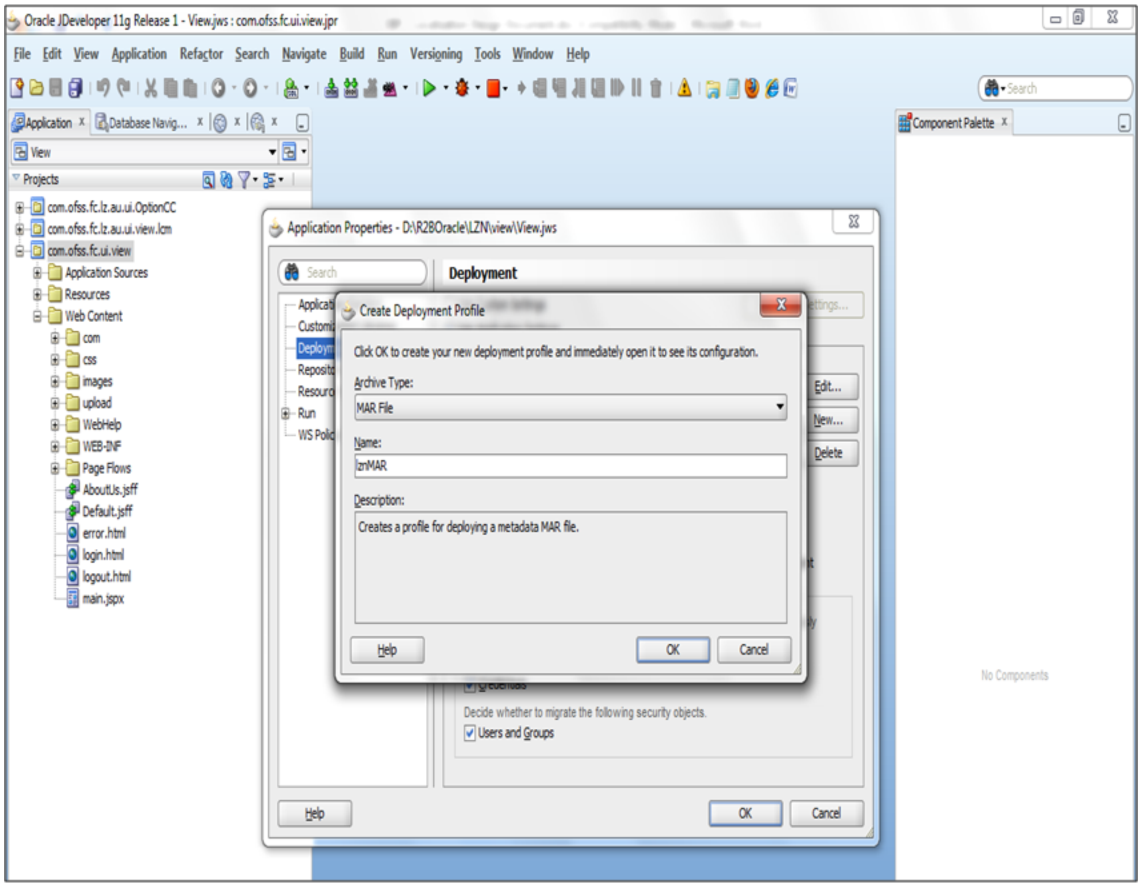
Description: Creates a profile for deploying a metadata MAR file.

Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.lz.au.ui.view.lcm.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Application Properties - D:\R2BOracle\LZN\view\View.jws
Deployment
EAR File
MAR File
SOA Bundle
Help
OK
Cancel
Decide whether to migrate the following security objects.
Users and Groups
Component Palette
Property Inspector

Step 4

Select the MAR File option.

Figure 8–14 MAR Creation - MAR File Selection


Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.ui.view.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Application Properties - D:\R2BOracle\LZN\view\View.jws
Deployment
Create Deployment Profile
Help
OK
Cancel
Edit...
New...
Delete
Component Palette
No Components

Click OK to create your new deployment profile and immediately open it to see its configuration.

Archive Type: MAR File

Name: lznMAR

Description: Creates a profile for deploying a metadata MAR file.

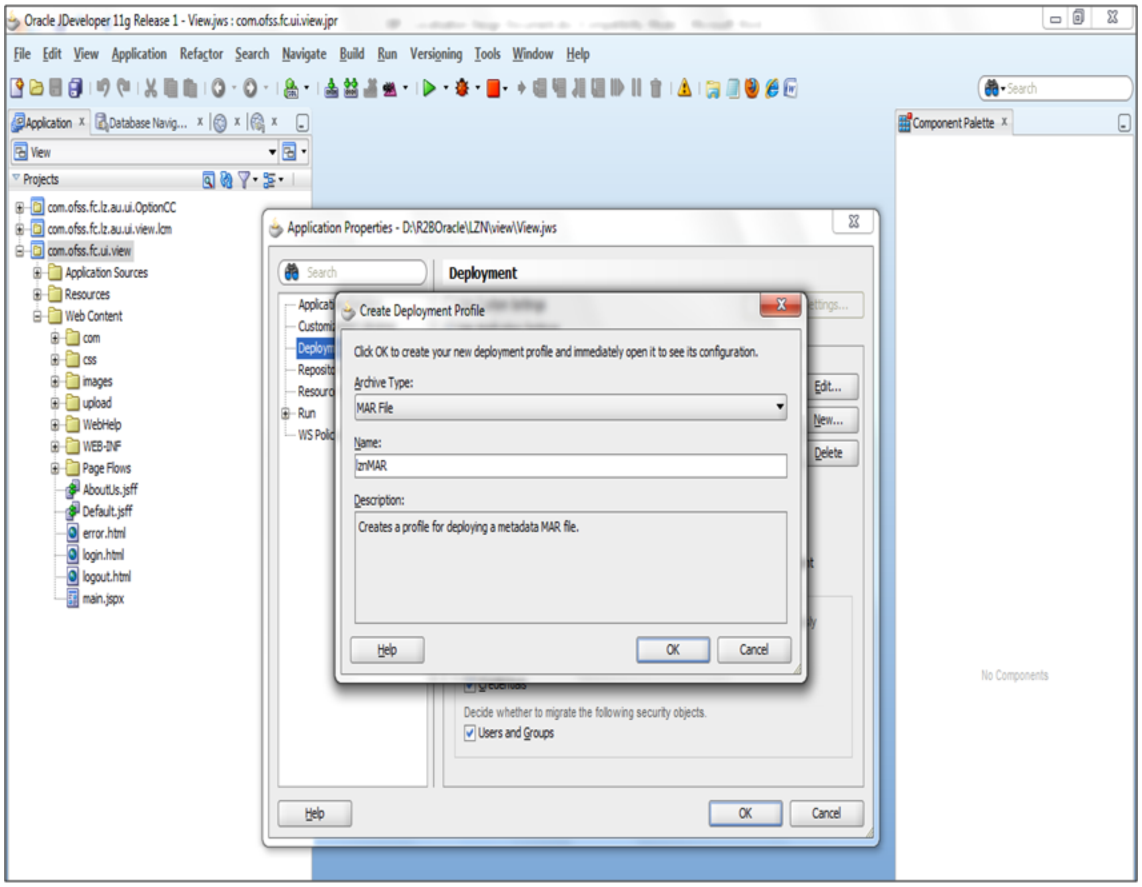
Decide whether to migrate the following security objects.

☑ Users and Groups

Step 5

Select MAR from Archive Type and give a name ending with MAR and click **Ok**.

Figure 8–15 MAR Creation - Enter Details


Create Deployment Profile

Click OK to create your new deployment profile and immediately open it to see its configuration.

Archive Type: MAR File

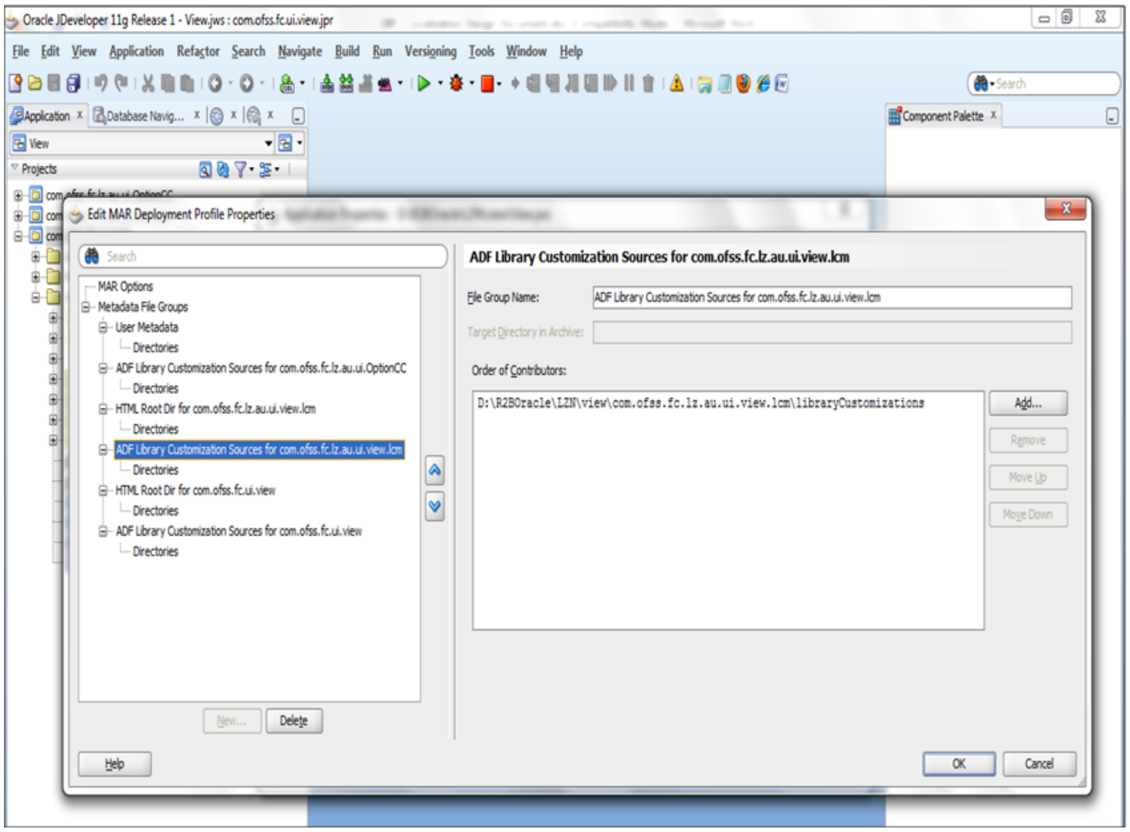
Name: lznMAR

Description: Creates a profile for deploying a metadata MAR file.

Step 6

Select the ADF Library Customization for com.ofss.fc.lz.au.ui.view.lcm.

Figure 8–16 MAR Creation - ADF Library Customization


Oracle JDeveloper 11g Release 1 - View.jws : com.ofss.fc.ui.view.jpr
File Edit View Application Refactor Search Navigate Build Run Versioning Tools Window Help
Application
Database Navig...
Component Palette
View
Projects
Edit MAR Deployment Profile Properties
Search
New...
Delete
Add...
Remove
Move Up
Move Down
Help
OK
Cancel

- MAR Options
- Metadata File Groups
  - User Metadata
    - Directories
  - ADF Library Customization Sources for com.ofss.fc.lz.au.ui.OptionCC
    - Directories
  - HTML Root Dir for com.ofss.fc.lz.au.ui.view.lcm
    - Directories
  - ADF Library Customization Sources for com.ofss.fc.lz.au.ui.view.lcm
    - Directories
  - HTML Root Dir for com.ofss.fc.ui.view
    - Directories
  - ADF Library Customization Sources for com.ofss.fc.ui.view
    - Directories

ADF Library Customization Sources for com.ofss.fc.lz.au.ui.view.lcm

File Group Name: ADF Library Customization Sources for com.ofss.fc.lz.au.ui.view.lcm

Target Directory in Archive:

Order of Contributors:

```
D:\R2BOracle\LZN\view\com.ofss.fc.lz.au.ui.view.lcm\libraryCustomizations
```

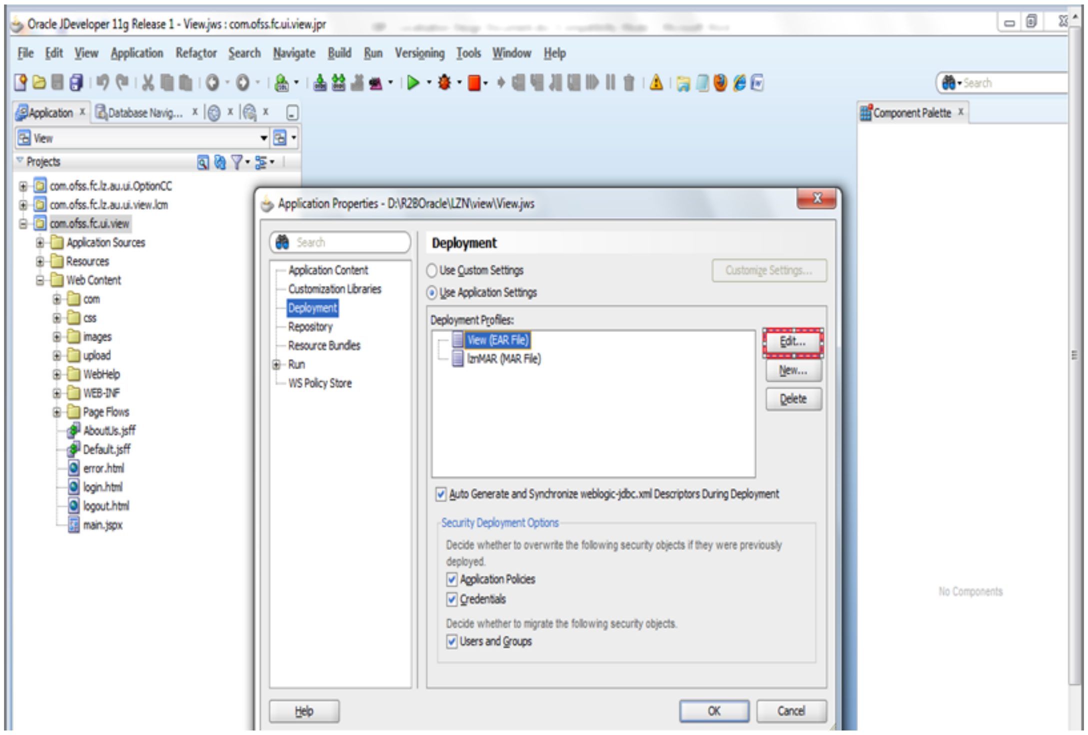
Step 7

Select the project for which Library Customization will be included in MAR (com.ofss.fc.lz.au.ui.view.lcm) and click **OK.**

Step 8

Select View (EAR File) and click Edit.

Figure 8–17 MAR Creation - Edit File



Application Properties - D:\R28Oracle\LZN\view\View.jws

- Application Content
- Customization Libraries
- Deployment
- Repository
- Resource Bundles
- Run
- WS Policy Store

Deployment

( ) Use Custom Settings
(●) Use Application Settings

Customize Settings...

Deployment Profiles:

- View (EAR File)
- lznMAR (MAR File)

Edit... New... Delete

[✓] Auto Generate and Synchronize weblogic-jdbc.xml Descriptors During Deployment

Security Deployment Options

Decide whether to overwrite the following security objects if they were previously deployed.

[✓] Application Policies
[✓] Credentials

Decide whether to migrate the following security objects.

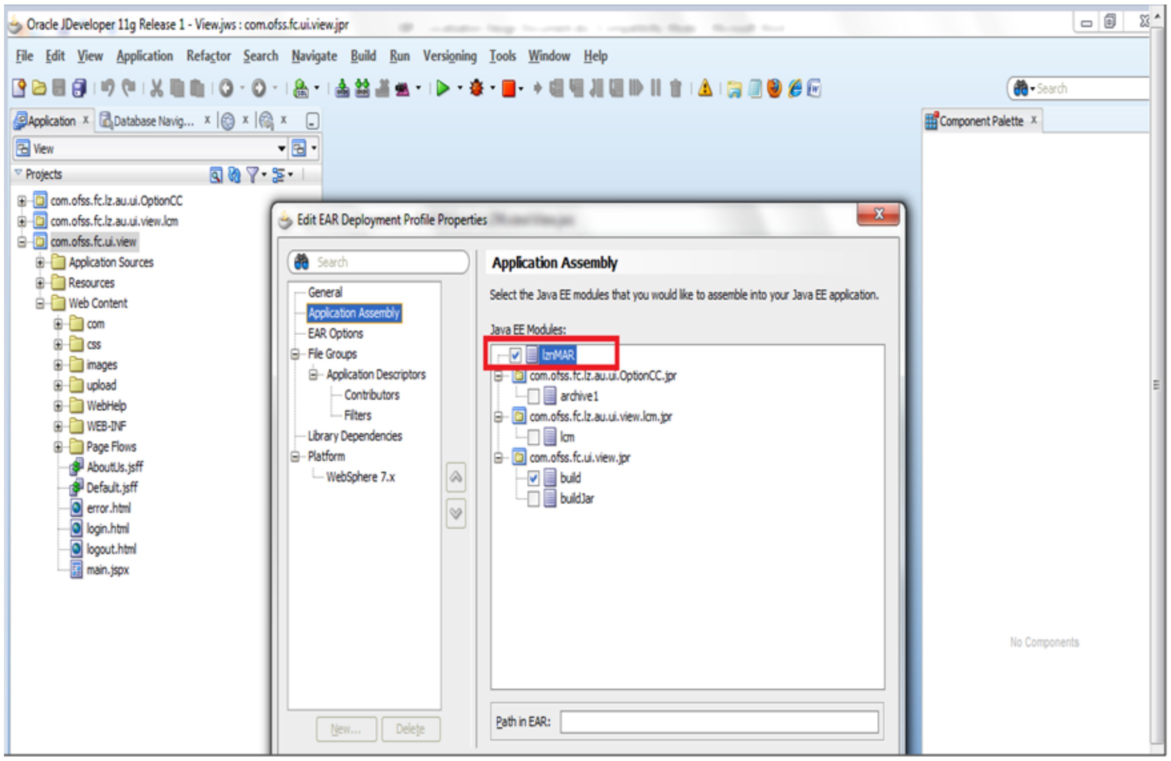
[✓] Users and Groups

Help OK Cancel

Step 9

Select Application Assembly and check the created MAR (lznMAR) and click ok on defaults.

Figure 8–18 MAR Creation - Application Assembly



8.3 Source Maintenance and Build

This section describes the source maintenance and build details.

8.3.1 Source Check-ins to SVN

Along with UI and middleware source maintenance, there is a set of metadata files required to be packaged in the deployable packages in order for customization. When performing any changes to a product screen in "customization mode" the corresponding <screen filename>.xml gets generated. In case of taskflows, the metadata file is <page definition filename>.xml. The path structure is provided in the below table.

Table 8–1 Path Structure

For page definition	
File name (with path)	adfmsrc/com/ofss/fc/ui/view/lcm/collaterals/collateralPerfectionCapture/ pageDefn/CollateralPerfectionCapturePageDef.xml
Meta-data file name	com\ofss\fc\ui\view\lcm\collaterals\collateralPerfectionCapture\ pageDefn\mdssys\cust\option\LZ\CollateralPerfectionCapturePageDef.xml.xml

(with path)	
For Screens	
File name (with path)	com/ofss/fc/ui/view/lcm/collaterals/collateralPerfectionCapture/form/CollateralPerfectionCapture.jsff
Meta-data file name (with path)	com\ofss\fc\ui\view\lcm\collaterals\collateralPerfectionCapture\form\mdssys\cust\option\LZ\CollateralPerfectionCapture.jsff.xml

These meta-data sources are checked into the METADATA folder in the product SVN under the localization path. During deployment, the EAR implementing these customizations must include these above mentioned sources in a .mar file.

8.3.2 .mar files Generated during Build

The localization specific build will include a last step, which is creation of .mar (metadata archive) file from the files checked-in the METADATA folder. This step will create separate .mar files, based on the modules which these represent. These MAR files are then packaged inside the deployable application EAR (com.ofss.fc.ui.view.ear).

Typical mar files generated during build will follow the naming convention com.ofss.fc.lz.au.ui.view.<module>.mar. Example, com.ofss.fc.lz.au.ui.view.pc.mar

8.3.3 adf-config.xml

adf-config.xml stores design time configuration information. The cust-config section (under mds-config) in the file contains a reference to the customization class. As part of the build activity, this file needs to be placed in the path com.ofss.fc.ui.view.ear@/adf/META-INF/. Also the customization class should be available in the classpath during deployment.

8.4 Packaging and Deployment of Localization Pack

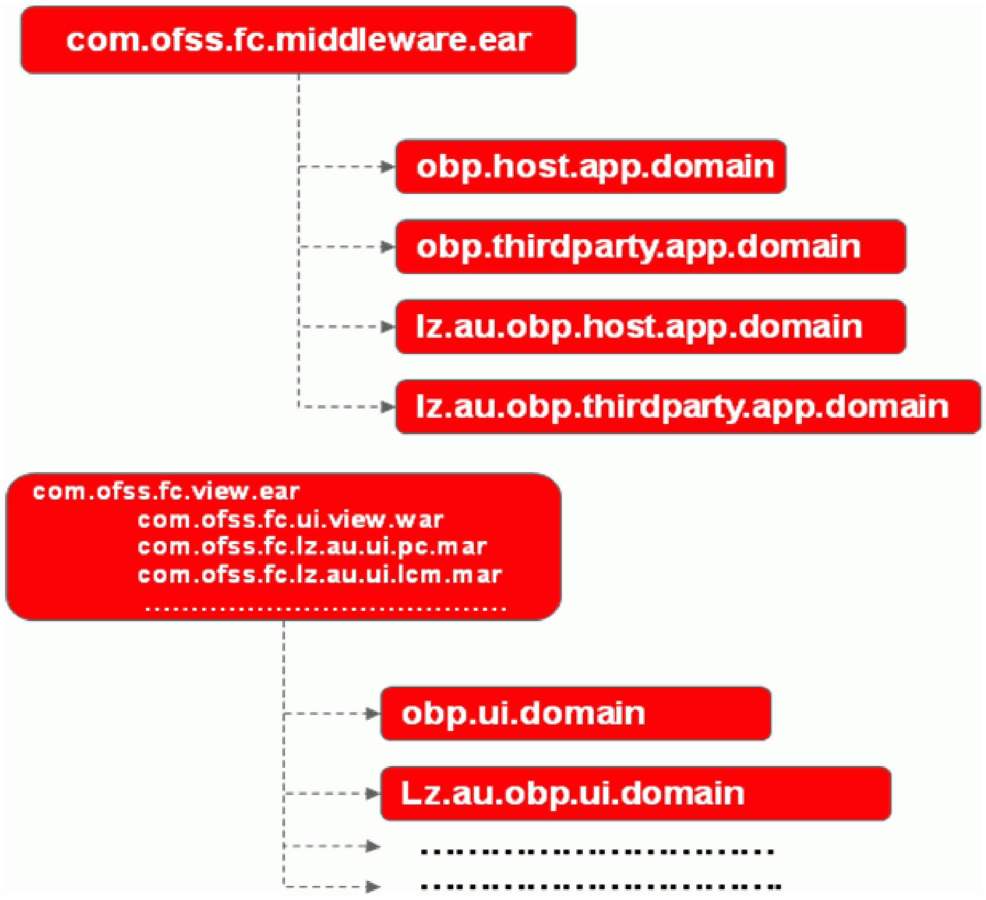
In the OBP application, different projects will be shipped in the form of library jars which can be customized and the new localization-specific application libraries can be created. In the application, the assembly has been specifically modularized to take care of multiple localizations by prevention of mix-up of jars. The naming convention for the jars can be defined for different clients differently.

The new customized jars for hosts and UI needs to be packed with the original jars in the EAR files which will be deployed on the server. Let's say, we are creating the extension hooks of 'obp.host.app.domain' jar, then the separate jars can be defined as 'lz.au.obp.host.app.domain' and 'lz.us.obp.host.app.domain' for Australia and US respectively.

The similar structure can also be maintained for the other applications across UI and SOA channels. 'lz.au.obp.ui.domain' can be defined for the customized jar of the project 'obp.ui.domain'.

The new customized jars for hosts and UI are packed below with the original jars in the EAR files which will be deployed on the servers.

Figure 8–19 Package Deployment


com.ofss.fc.middleware.ear
obp.host.app.domain
obp.thirdparty.app.domain
lz.au.obp.host.app.domain
lz.au.obp.thirdparty.app.domain
com.ofss.fc.view.ear
com.ofss.fc.ui.view.war
com.ofss.fc.lz.au.ui.pc.mar
com.ofss.fc.lz.au.ui.lcm.mar
obp.ui.domain
Lz.au.obp.ui.domain

9 Deployment Guideline

This chapter explains the deployment guidelines.

9.1 Customized Project Jars

The customized extension projects are to be bundled in the different extensibility jars which are required to be added in the extensibility.

9.2 Database Objects

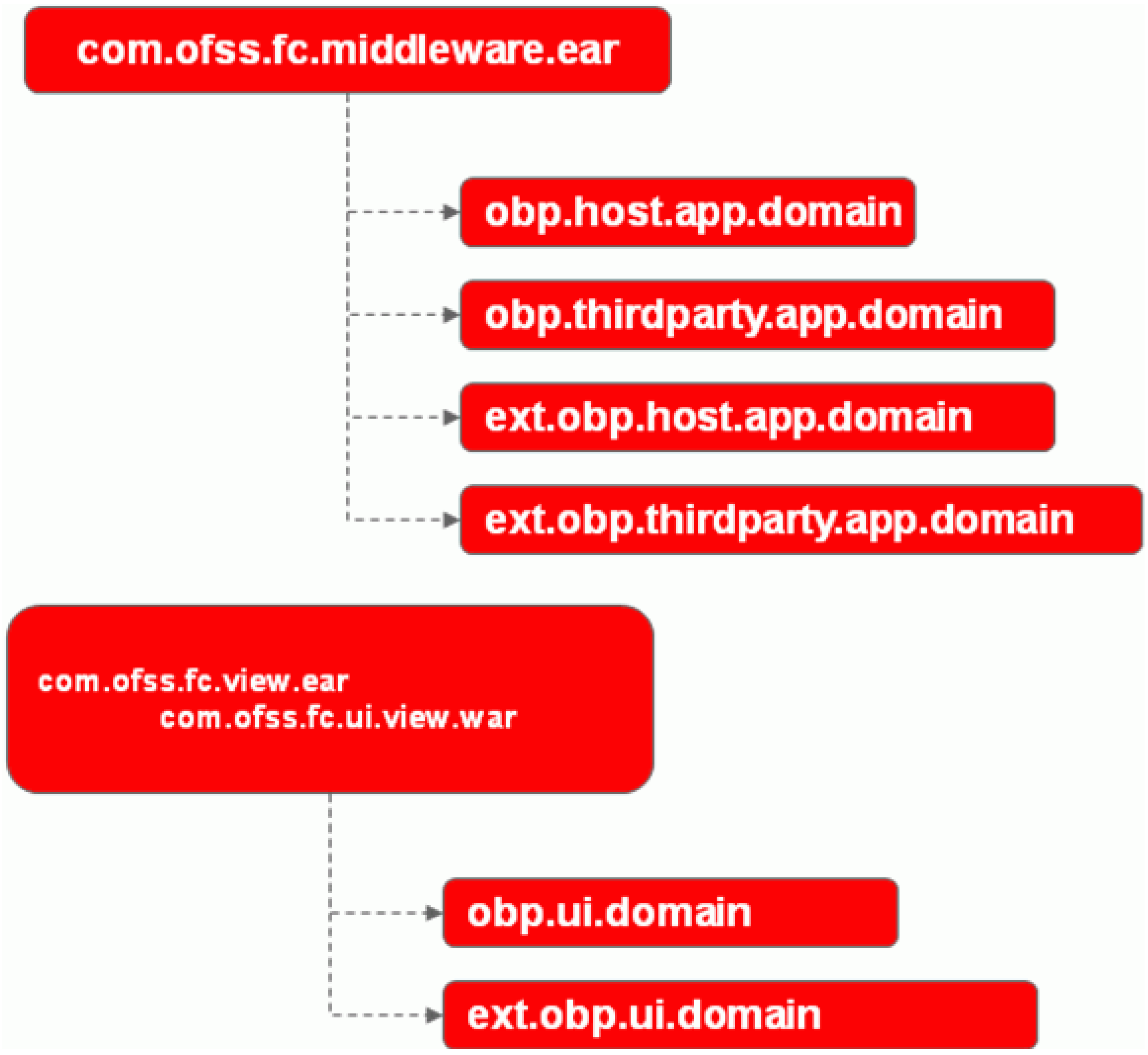
User has to update the corresponding seed data for the implementation of different extensibility features.

9.3 Extensibility Deployment

The new customized extensibility jars will be added in the extensibility libraries as ext.obp.host.domain for the host middleware layer, ext.obp.ui.domain for UI or presentation layer and ext.obp.soa.domain for the SOA layer. These extensibility application libraries will be packaged and shipped as the separate library folders along with the original library folders so that the extensibility feature can be added.

The OBP deployed applications shall reference these libraries so that customization jars included into these get automatically referenced in the corresponding EAR and WAR files.

Figure 9–1 Extensibility Deployment


com.ofss.fc.middleware.ear
obp.host.app.domain
obp.thirdparty.app.domain
ext.obp.host.app.domain
ext.obp.thirdparty.app.domain
com.ofss.fc.view.ear
com.ofss.fc.ui.view.war
obp.ui.domain
ext.obp.ui.domain