

**Oracle® Retail Accelerators for WebLogic Server
11g**

Micro-Applications Development Tutorial

October 2010

Note: The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Contents

1	Introduction	
	How to Use this Document.....	1-2
	Creating a Regular Item in RMS	1-3
	About the Create Regular Item Micro-Application.....	1-3
	Accessing the Sample JDeveloper Project.....	1-4
	Using the Sample JDeveloper Project.....	1-4
2	Creating the Create Regular Item Micro-Application	
	Creating a new ADF application.....	2-1
	Creating Model Layer Objects.....	2-4
	Creating the Department View Object.....	2-5
	Setting Up the DepartmentRVO View Object for LOV	2-7
	Setting Up the View Criteria	2-8
	Setting Up the DepartmentID Attribute as an LOV	2-9
	Creating the Class View Object.....	2-13
	Internationalization of Labels	2-14
	Adding View Objects to the Application Module.....	2-18
	Creating SQL Object Types in Java	2-21
	Creating the User Interface.....	2-24
	Creating a Page Fragment.....	2-24
	Constructing the User Interface	2-26
	Setting Up a Managed Bean	2-28
	Calling X API in the Micro-Application	2-31
	Creating a Business Service User Interface Button	2-34
	PL/SQL Wrapper for Item Sequence Generator in RMS	2-37
	Creating an ADF Task Flow	2-41
	Packaging the Application	2-43
	Subclass View Object.....	A-1
	SystemOptions View Object.....	A-2
	Supplier View Object.....	A-2
	Supplier Site View Object.....	A-3
	Supplier Country View Object.....	A-3
	Item Number Type View Object	A-3
	Cost Zone Group View Object.....	A-4
	Retail Price Zone Group View Object.....	A-4

Standard Unit of Measure View Object..... A-5

Introduction

Oracle Retail Micro-Applications are ADF taskflows that are designed to carry out a single or multiple activities of an Oracle Retail application from outside the application. Micro-Applications can be packaged in an ADF JAR library. Once packaged in an ADF JAR library, they can be easily incorporated in dashboards or workspaces. They can also be designed to use the business services or PL/SQL procedures defined by an application to complete a task.

This tutorial describes how you can create a Micro-Application for the *Creating a Regular Item* task in the Retail Merchandising System (RMS) application using Oracle JDeveloper and ADF. To complete the task, the Micro-Application will be designed to use the same APIs and RIB objects used by RMS.

This document includes the following topics:

- [Chapter 1, "Introduction"](#) – This chapter introduces you to the concept of Micro-Applications and business process illustrated in this tutorial.
- [Chapter 2, "Creating the Create Regular Item Micro-Application"](#) – This chapter describes how you can create the Create Regular Item Micro-Application.
- [Appendix A, "View Objects"](#) – This appendix provides information on the read-only view objects defined in the Create Regular Item Micro-Application.

Before proceeding, ensure that you have Oracle JDeveloper 11g (Release 11.1.1.3.0) installed on your system. Oracle JDeveloper is available for download on Oracle Technology Network. You must also have access to an Oracle Retail Merchandising System Release 13.1 environment.

Audience

This document is intended for application integrators and implementation personnel who are familiar with the Oracle Application Development Framework (ADF), specifically ADF Faces, ADF Business Components, and ADF Task Flows. Knowledge of Oracle Retail Merchandising System (RMS) and Retail Integration Bus (RIB), specifically X APIs and RIB objects, is also required.

Additional Reference Documentation

- To get acquainted with the X APIs and RIB objects used in this document, refer to the *Oracle Retail Merchandising System Operations Guide, Release 13.1 (Volume 2 – Message Publication and Subscription Designs)*. Specifically, refer to the section *Subscription Designs Item Subscription API*.
- To get acquainted with the RMS database schema used in this document, refer to the *Oracle Retail Merchandising System Data Model (Release 13.1)*. The Data Model is available on My Oracle Support with the Note ID 945584.1.

- To integrate the Micro-Applications with Oracle WebCenter Spaces and customize WebCenter Spaces to include custom developed content, refer to the *My Oracle Support Note ID 1189403.1 – Oracle Retail Accelerators Guide for WebCenter 11g*.

About Business Services in RMS (X API)

As business services, RMS provides PL/SQL procedures, called as X APIs, that provide an interface for external applications to initiate business processes such as creating an item, creating purchase orders, and so on. Micro-Applications use these PL/SQL procedures to carry out a specific task. These PL/SQL procedures take SQL objects, called RIB objects, as parameters. These RIB objects encapsulate and provide the user inputs to the procedure.

About Oracle Retail Accelerators Guide for WebCenter 11g

Oracle Retail Accelerators Guide for WebCenter 11g enables you to leverage the Oracle WebCenter infrastructure to create and modify workspaces specific to your business need. Apart from the capabilities that are available out-of-the-box, this document enables you to create, customize, and deploy expanded functionality by combining assets from the Oracle Fusion Middleware suite.

How to Use this Document

This document is a tutorial that will enable you to create a Micro-Application. A typical Micro-Application consists of several different components such as view objects, application module, page fragments, managed beans, and so on.

This document attempts to teach the concepts behind creating these components. All important components are illustrated in detail with examples. Once you know all the information and procedures to create a particular type of component, you can similarly create rest of the components. The document also includes hints that will guide you through the steps.

This document is also accompanied by a JDeveloper project with sample code for the CreateRegularItem Micro-Application. The JDeveloper project and components included follow the structure mentioned in this document. If you follow the same packaging structure as mentioned in this document, you can then choose to copy the source code from the relevant files in the same JDeveloper project. It is recommended that you create all the components mentioned in this document. This ensures that all the required directory structure are created automatically in your working project. You can then easily copy the source code to your working project. For more information on accessing the sample JDeveloper project, see [Accessing the Sample JDeveloper Project](#).

Important: The accompanied JDeveloper project with the sample code for the CreateRegularItem Micro-Application is available for illustration purposes only.

The code snippets or screenshots included in this document may not match the sample code included in the JDeveloper project. Since the sample code was intended for illustration purposes only, it was not constructed as production code and is not robust enough to validate all inputs or handle all exceptions.

Creating a Regular Item in RMS

In RMS items are categorized in a hierarchy structure. At the top-most level of the hierarchy is Department, followed by Class, Subclass, and at the lowest level is the item (Department > Class > Subclass > Item). Also, each item in RMS contains a lot of other mandatory information.

To create a new item in RMS, all the mandatory information about the item must be provided. You must first define the hierarchy structure for the item. This can be done by selecting an existing **Department** in RMS. You must then select a **Class** within the selected department, followed by a **Subclass** in the selected Class. This becomes the hierarchy of the newly created item. Once the hierarchy is defined, you must set up the other mandatory information. Each item must have a **Supplier** or a **Supplier site** that exists in RMS. This Supplier must have a specific **Country of sourcing** for the item. **Unit cost** for the item must be associated with a **Cost Zone** and a **Retail Price Zone Group**. A **Unit of measure** must be defined for the item along with a **Conversion factor**. Further, each item in RMS has a unique code. This code can be of different types and is generated automatically by RMS, except when the code type is manual in which case users will enter the code manually. Therefore, the **Item number code type** from code types available in RMS must be set for the item. To get a better understanding of these Item properties in RMS, you can refer to the Oracle Retail Merchandising System Data Model.

About the Create Regular Item Micro-Application

In this tutorial, you will create a Create Regular Item Micro-Application that will be used to create a regular item in RMS. Users will then be able to provide the information for the item in a user interface outside of RMS which will call the RMS Business Service to create the item.

Figure 1–1 Create Regular Item Micro-Application Screen

The screenshot displays the 'Create Regular Item Micro-Application' interface. It features several input fields for mandatory information, each with a search icon to its right:

- Item description
- * Department ID
- Class ID
- Subclass ID
- * Supplier ID
- Supplier site ID
- * Supplier country ID
- Unit cost

Below these fields is an 'Advanced' section, indicated by a downward arrow and the text 'Advanced'. It includes a checkbox for 'Edit advanced fields' which is currently unchecked. The advanced fields include:

- Item number type: ITEM
- Item Number (Manual)
- Cost zone group ID: 1000
- Retail price zone group ID: 1
- Standard unit of measure (UOM): EA
- UOM conversion factor: 1

At the bottom of the form are two buttons: 'Create Item' and 'Reset'.

The figure above illustrates the Create Regular Item Micro-Application. Users can enter values specific to the new item being created directly in the user interface. The form is divided in two parts. The upper half consists of values that the users must provide each time they create a new item. The lower half of the form, under the Advanced section, contains fields that have default values which the users may choose to edit (or use as is). Once the users enter all the relevant values, they can press the Create Item button at the bottom of the screen to create the item. When the button is pressed, all the values entered by the users are submitted to the server, and the RMS business service to create a new item is called. The result of the business service call is returned to the user as a message on screen.

Accessing the Sample JDeveloper Project

Although this tutorial includes all the information you need to create a Micro-Application, a JDeveloper project with sample code is also available on My Oracle Support with the patch ID 10149374 for reference purposes.

To access the sample JDeveloper project:

1. In a Web browser, open the following URL:
<https://support.oracle.com/>
The My Oracle Support Web page appears.
2. Select a language and sign on to the Web site by clicking **Sign In**.
Once signed in, the **My Oracle Support | Dashboard** screen appears.
3. Click the **Patches & Updates** tab.
4. On the **Patch & Updates** screen, under **Patch Search**, click **Patch ID or Number**.
5. In the **Patch ID or Number is** field, enter **10149374**.
6. Optionally, you can also choose a platform from the **Platform is** drop-down list.
7. Click **Search**. The **Patch Search Results** screen appears.
8. In the **Patch Search Results** screen, under **Patch ID**, click the relevant patch.
9. On the next screen, click **Download** (appears on the left side of the screen).

Note: On the Patch Search Results screen, you can also select the row that matches the patch description, and then click Download on the toolbar that appears under the selected row.

10. Unpack the ZIP file to your working directory.

Using the Sample JDeveloper Project

To use the sample JDeveloper project:

1. In JDeveloper, from the **File** menu, click **Open**.
2. Navigate to your working directory where you extracted the contents of the ZIP file.
3. Select the **MicroApps.jws** file, and click **Open**. The JDeveloper project opens in JDeveloper.

Before you can use the project, you must first set up the RMS database connection for the project. To set up the RMS database connection:

1. From the **Applications Navigator**, expand **Application Resources** navigator.
2. Expand **Connections**, and then expand **Database**.
3. Right-click on **RMS_DB**, and then click **Properties**. The **Edit Database Connection** window appears.
4. Enter relevant information in the **Edit Database Connection** window, and then click **OK**.

Creating the Create Regular Item Micro-Application

This chapter describes how you can create the Create Regular Item Micro-Application. To create the Create Regular Item Micro-Application, you must complete the following steps:

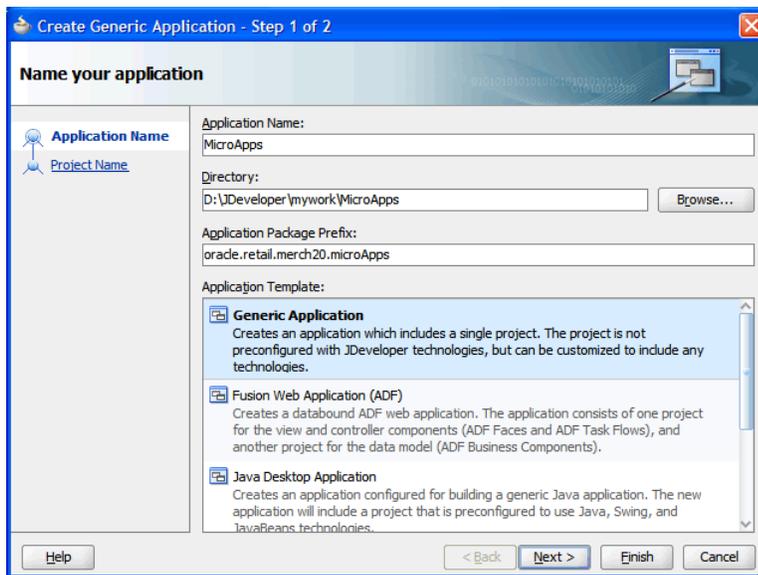
1. [Creating a new ADF application](#)
2. [Creating Model Layer Objects](#)
3. [Internationalization of Labels](#)
4. [Adding View Objects to the Application Module](#)
5. [Creating SQL Object Types in Java](#)
6. [Creating the User Interface](#)
7. [Calling X API in the Micro-Application](#)
8. [Creating a Business Service User Interface Button](#)
9. [PL/SQL Wrapper for Item Sequence Generator in RMS](#)
10. [Creating an ADF Task Flow](#)
11. [Packaging the Application](#)

Creating a new ADF application

You must first start by creating a ADF-based application and a project that is based on the ADF-based technologies. To create a new ADF-based application:

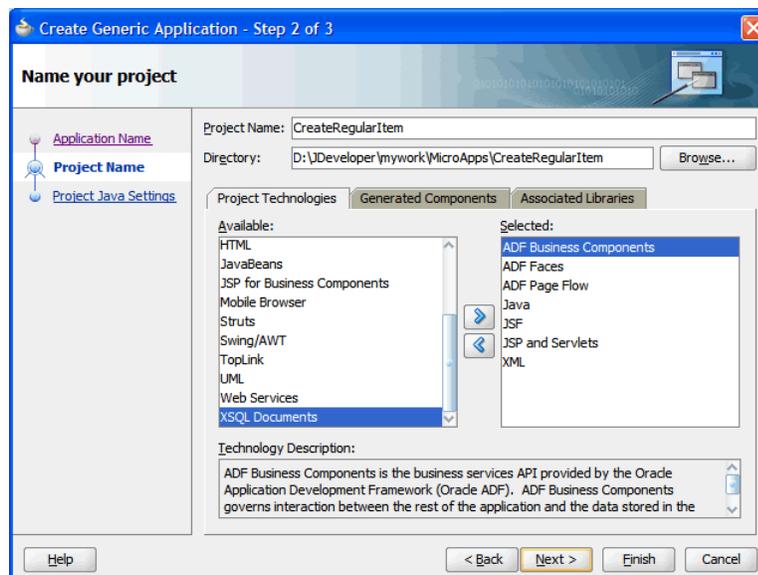
1. In JDeveloper, from the **File** menu, click **New**. The **New Gallery** wizard appears.
2. In the **New Gallery** wizard, under **General**, click **Applications**.
3. In the **Items** area, click **Generic Application**, and then click **OK**. The **Create Generic Application** wizard appears.

Figure 2–1 Create Generic Application Wizard – Name your application Screen



4. On the **Name your application** screen, enter **MicroApps** in the **Application Name** field.
5. In the **Directory** field, select a working directory for the application.
6. In the **Application Package Prefix** field, enter a relevant package prefix. For example, **oracle.retail.merch20.microApps**.
7. Ensure that **Generic Application** is selected in the **Application Template** area, and then click **Next**. The **Name your project** screen appears.

Figure 2–2 Create Generic Application Wizard – Name your project Screen

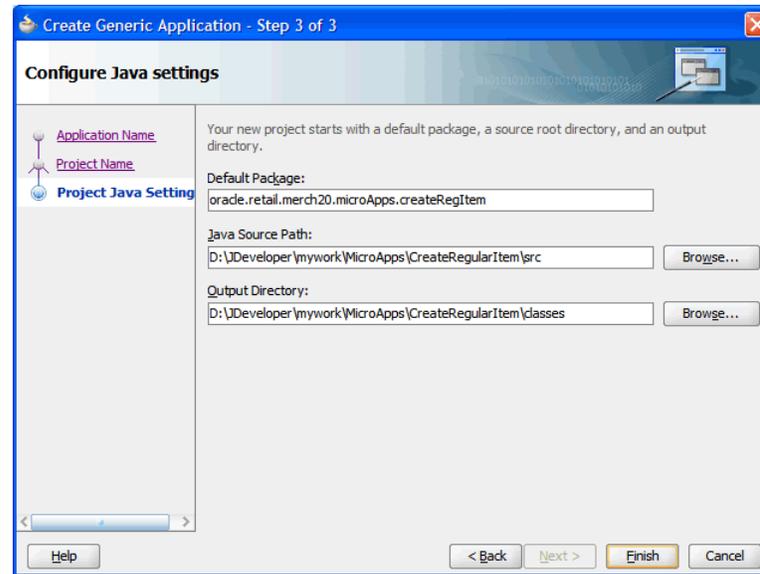


8. On the **Name your project** screen, enter **CreateRegularItem** in the **Project Name** field.

Note: By default, the **Directory** field displays the working directory you set in the previous screen along with the project sub folder.

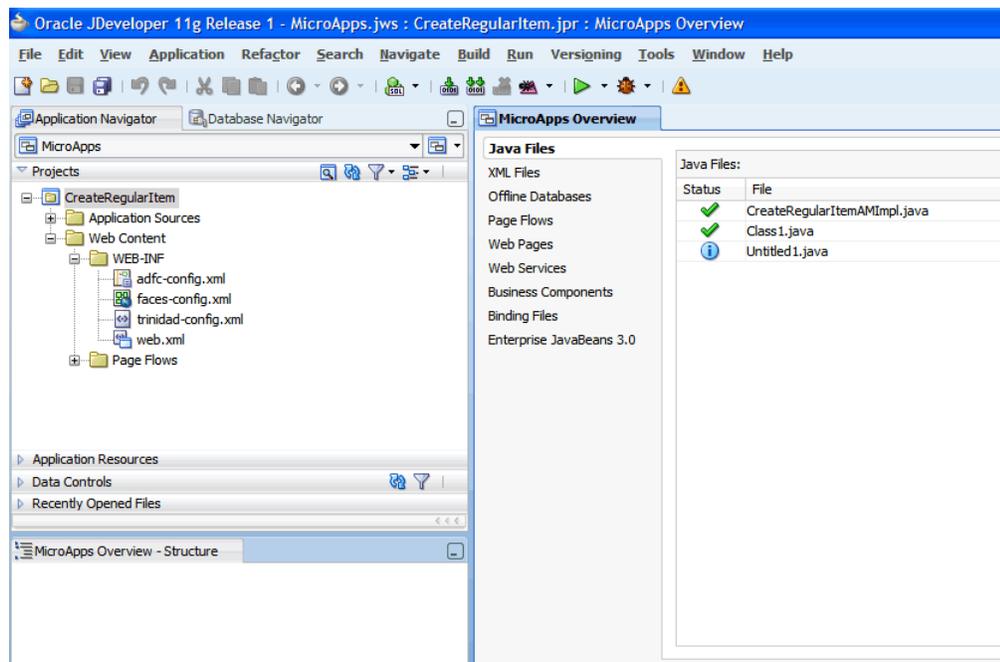
9. On the **Project Technologies** tab, select **ADF Business Components**, **ADF Faces**, **ADF Page Flow**, **HTML**, **Java**, **JSF**, **JSP and Servlets**, and **XML**, and move them to the **Selected** area.
10. Click **Next**. The **Configure Java settings** screen appears.

Figure 2–3 Create Generic Application Wizard – Configure Java settings Screen



11. In the **Default Package** field, append the application package prefix name with the project name. For example, **oracle.retail.merch20.microApps.createRegItem**.
12. Click **Finish**. The **Application Navigator** appears with the **MicroApps** application and the **CreateRegularItem** project.

Figure 2–4 MicroApps Overview Tab in the Editor Window



Note: An ADF Web-based application typically has the following projects:

- Model – intended to provide the Model layer with Business components.
- ViewController – intended to provide the View layer with ADF Faces.

This is a recommended practice to keep the Model layer separate from the View layer. Since the Micro-Applications are very small applications, this practice is not needed.

The Model and View layer are merged automatically in a single project when you selected all the required technologies. This way each project represents a single Micro-Application and all Micro-Applications reside in a single ADF application called MicroApps.

Creating Model Layer Objects

The Create Regular Item Micro-Application will contain several View objects (View objects of the ADF Business Component). These View objects will be used to read data from the relevant database tables in the RMS database schema and display the data in the user interface. Since these objects will be used only to read data, they will need to be set up as "read-only" view objects. The following View objects are used in this application:

View Object name	Related table in RMS schema	Description
DepartmentRVO	deps	Department for the item.

View Object name	Related table in RMS schema	Description
ClassRVO	class	Class for the item.
SubclassRVO	subclass	Subclass for the item.
SystemOptionsRVO	system_options	System options for the RMS installation.
SupplierRVO	sup	Supplier for the item.
SupplierSiteRVO	sup	Supplier site.
SupplierCountryRVO	country	Supplier's country of sourcing for the item.
ItemNumberTypeRVO	code_detail	Item number sequence type.
CostZoneGroupRVO	cost_zone_group	Cost zone group for item.
RetailPriceZoneGroupRVO	rpm_zone_group	Retail price zone group for item.
StandardUomRVO	uom_class	Standard unit of measure.

Note: For more information on the RMS database tables listed above, refer to the Oracle Retail Merchandising System Data Model.

The following steps provide you an example of how you can declare a read-only view object in JDeveloper using the Department and Class view object:

- [Creating the Department View Object](#)
- [Setting Up the DepartmentRVO View Object for LOV](#)
- [Creating the Class View Object](#)

Creating the Department View Object

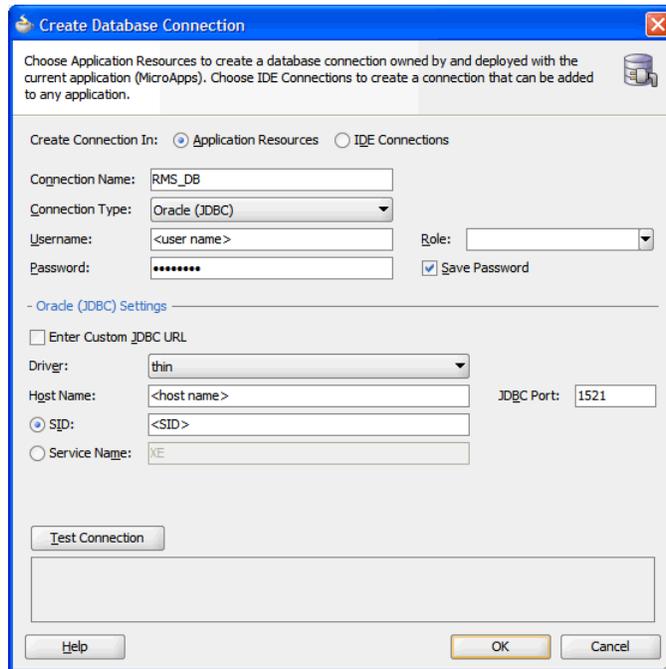
In RMS, all items are categorized in a Department, Class, and then Subclass hierarchy. Consider a scenario where a user needs to search through all departments in the RMS database schema, and then select an item. This can be achieved by defining a View object for Department in the following manner:

1. In the **Application Navigator**, right-click on the **CreateRegularItem** project, and then click **New**. The **New Gallery** wizard appears.
2. In the **Current Project Technologies** tab, under **Business Tier**, click **ADF Business Components**.
3. From the **Items** area, click **View Object**, and then click **OK**. The **Initialize Business Components Project** window appears.

The **Initialize Business Components Project** window enables you to set up a database connection to the RMS database schema.

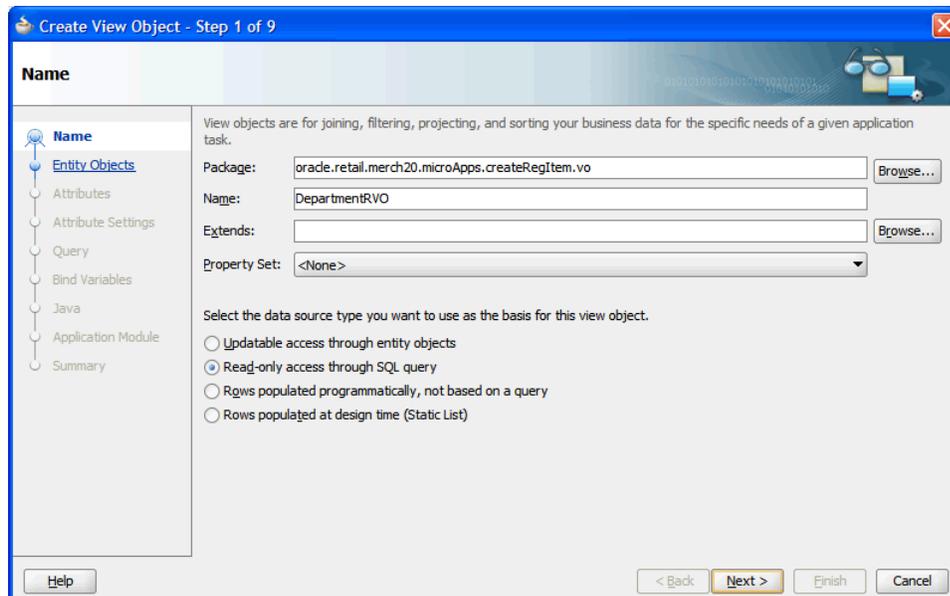
4. Click the **Plus (+)** button. The **Create Database Connection** window appears.

Figure 2–5 Create Database Connection Window



5. In the **Create Database Connection** window, enter a database connection name (for example, RMS_DB), user name, password, host name, JDBC port, and SID for the RMS database schema.
 Click **Test Connection** to ensure that a database connection is established.
6. Click **OK**. The **Create View Object** wizard appears.

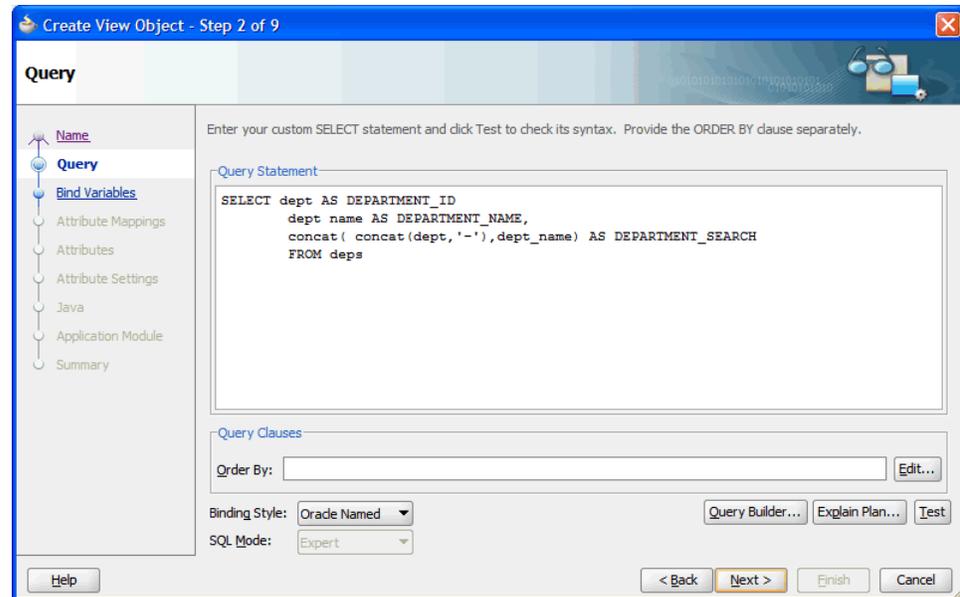
Figure 2–6 Create View Object Wizard – Name Screen



7. On the **Create View Object** wizard, enter the following information:

- Enter or select **oracle.retail.merch20.microApps.createRegItem.vo** in the **Package** field.
 - Enter **DepartmentRVO** in the **Name** field. **DepartmentRVO** is the View Object name where RVO indicates that this is a read-only view object.
 - Under **Select the data source type you want to use as the basis for this view object**, select the **Read-only access through SQL query** option.
8. Click **Next**. The **Query** screen appears.

Figure 2–7 Create View Object Wizard – Query Screen



9. On the **Query** screen, under **Query Statement**, enter the following statement:

```
SELECT dept AS DEPARTMENT_ID
       dept name AS DEPARTMENT_NAME,
       concat( concat(dept, '-'), dept_name) AS DEPARTMENT_SEARCH
FROM depts
```

10. Click **Next**. The **Bind Variables** screen appears.
11. On the **Bind Variables** screen, click **Next**. The **Attribute Mappings** screen appears.
12. On the **Attributes** screen, click **Finish**.

The **DepartmentRVO.xml** configuration tab appears in the **Editor** window.

Setting Up the DepartmentRVO View Object for LOV

Once the **DepartmentRVO** view object is created, you can proceed ahead to configure the **DepartmentId** attribute from the **DepartmentRVO** view object as a List of Values (LOV) component in the user interface. Once you define the attribute as an LOV, you can then use the attribute on a JSF page as an "ADF LOV Input" component (for more information, see the section **Creating User Interface**). Once implemented, such components are rendered as an input text field with a Search icon. Users can click the Search icon to view a list of attribute values and select a relevant value.

This section describes how you can set up a DepartmentId attribute in the DepartmentRVO View Object as an LOV. It includes the following tasks:

- [Setting Up the View Criteria](#)
- [Setting Up the DepartmentID Attribute as an LOV](#)

Setting Up the View Criteria

Before you set up the DepartmentID attribute as an LOV, you must first define the View criteria for the view object. View criteria is used to filter out relevant rows from the result set of the View object query.

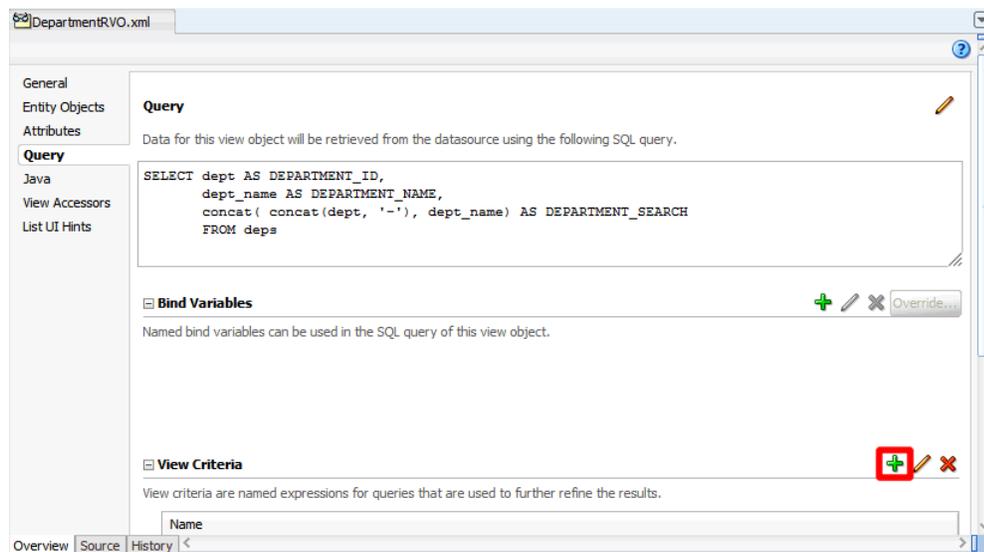
To set up the View criteria:

1. Open the **DepartmentRVO** view object configuration tab:
 - a. In the **Application Navigator**, under **CreateRegularItem**, expand **Application Sources**.
 - b. Under **Application Sources**, expand **oracle.retail.merch20**, **microApps**, **createRegItem**, and then **DepartmentRVO**.
 - c. Under **DepartmentRVO**, double-click **DepartmentRVO.xml**.

OR

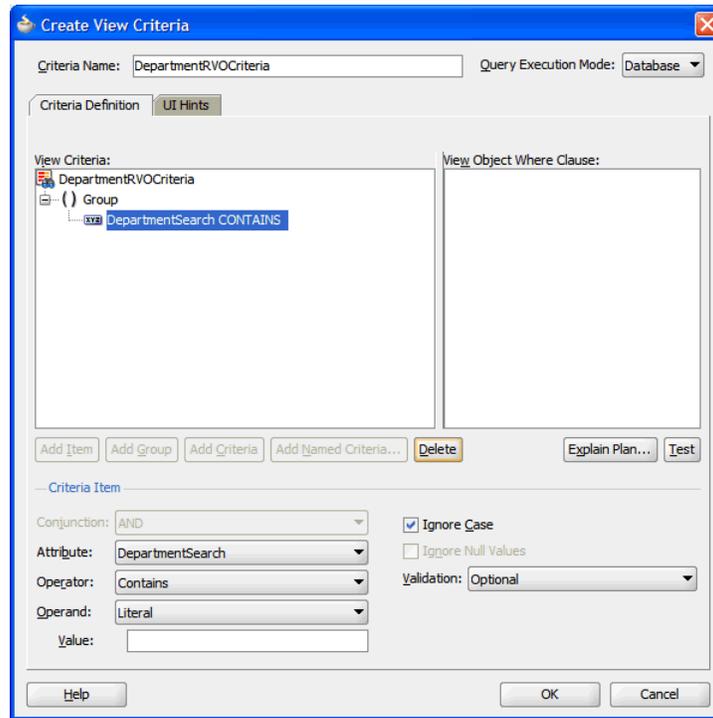
In case it was already opened before, in the **Editor** window, click the **DepartmentRVO.xml** configuration tab.
2. On the **DepartmentRVO.xml** tab, click **Query**. The **Query** page appears.

Figure 2–8 DepartmentRVO.xml Tab – Query Page



3. On the **Query** page, under **View Criteria**, click the **Plus (+)** button. The **Create View Criteria** window appears.

Figure 2–9 Create View Criteria Window



4. On the **Create View Criteria** window, in the **Criteria Definition** tab, select the default criteria (**DepartmentRVOCriteria**) in the **View Criteria** section, and then click **Add Item**.
5. In the **Criteria Item** section, enter the following information:
 - In the **Attribute** field, select **DepartmentSearch**.
 - In the **Operator** field, select **Contains**.
 - In the **Operand** field, select **Literal**.
 - Select the **Ignore Case** check box.
6. Click **OK**.

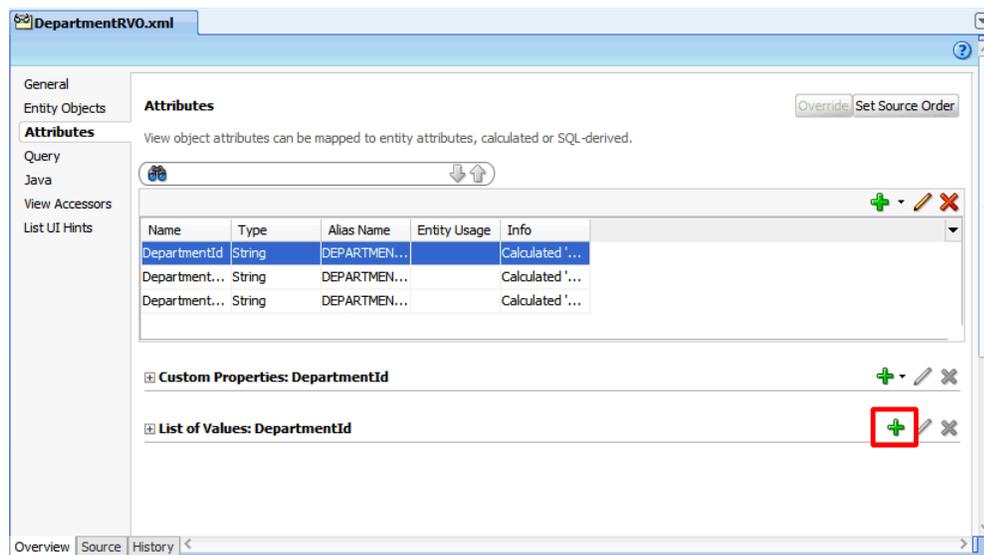
The **DepartmentRVOCriteria** view criteria is added to the **DepartmentRVO** view object.

Setting Up the DepartmentID Attribute as an LOV

To set up the DepartmentID Attribute as an LOV:

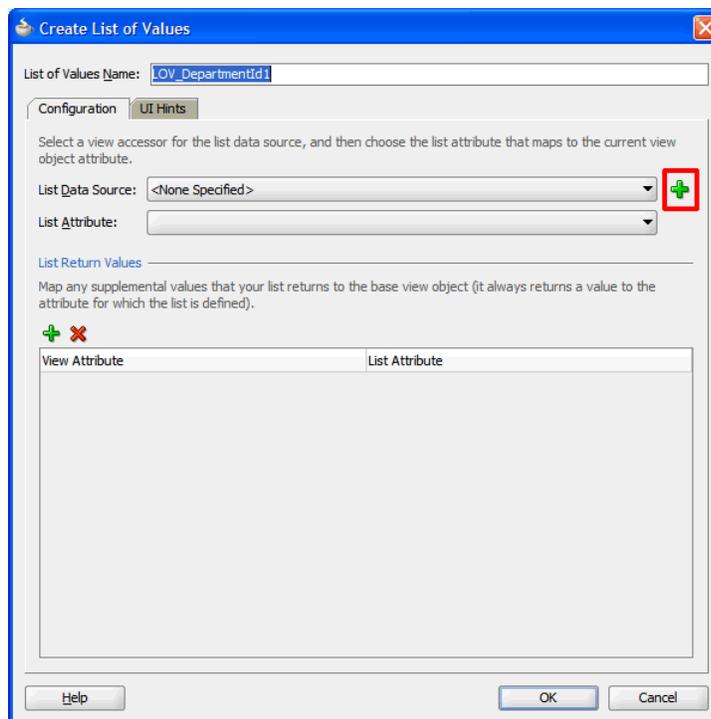
1. On the **DepartmentRVO.xml** tab, click **Attributes**. The **Attributes** page appears.

Figure 2–10 DepartmentRVO.xml Configuration Tab – Attributes Page

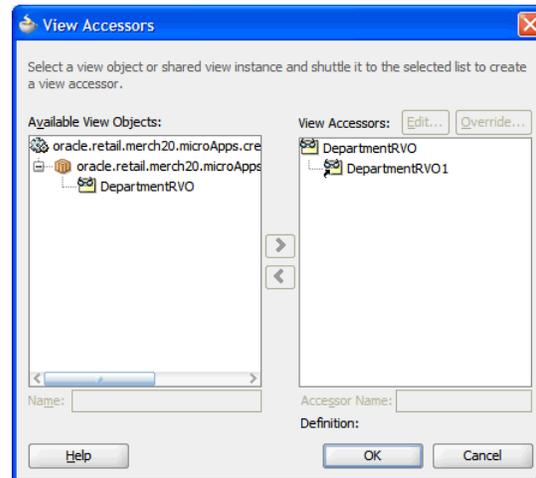


2. On the **Attributes** page, select the **DepartmentId** attribute, and then click the **Plus (+)** button in the **List of Values: DepartmentId** section. The **Create List of Values** window appears.

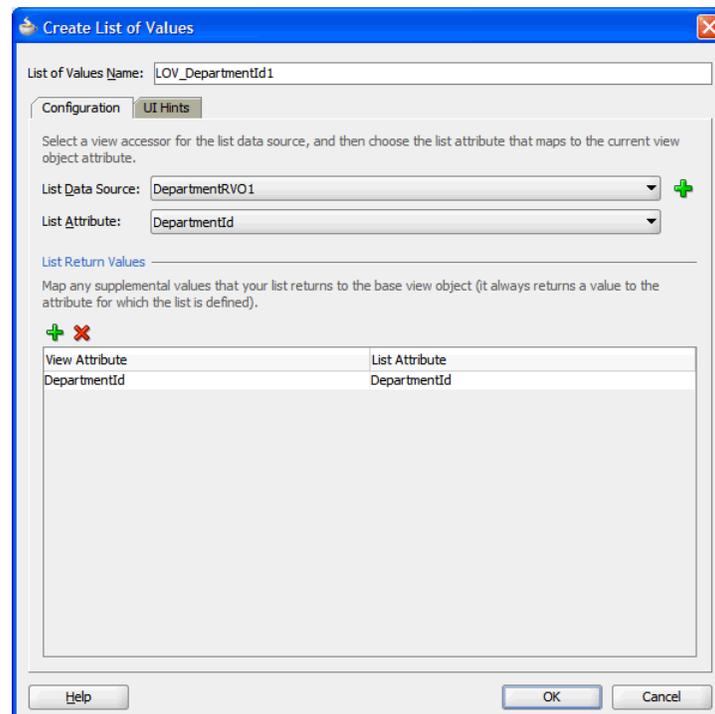
Figure 2–11 Create List of Values Window



3. In the **Create List of Values** window, under the **Configuration** tab, click the **Plus (+)** button next to the **List Data Sources** field. The **View Accessors** window appears.

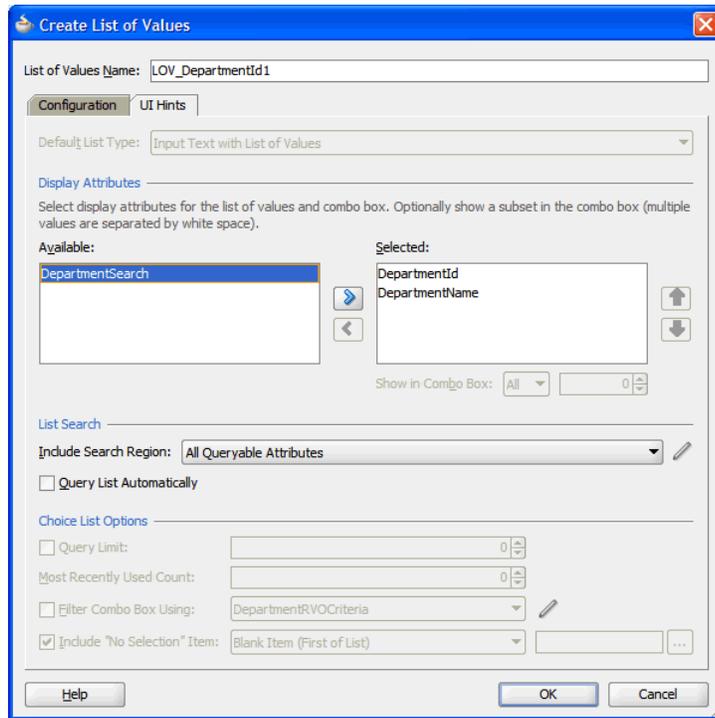
Figure 2–12 View Accessors Window

4. In the **View Accessors** window, select **DepartmentRVO** from the **Available View Objects** section, and move it to the **View Accessors** section under **DepartmentRVO**.
5. Click **OK**. The **Create List of Values** window appears.

Figure 2–13 Create List of Values Window

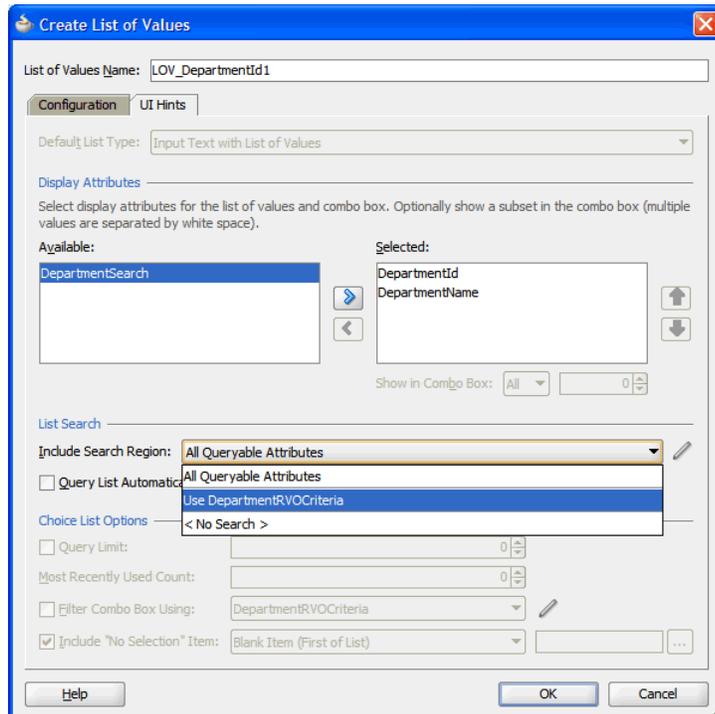
6. In the **Create List of Values** window, select **DepartmentId** in the **List Attribute** field.
7. On the **UI Hints** tab, move **DepartmentId** and **DepartmentName** from the **Available** list to the **Selected** list.

Figure 2–14 Create List of Values Window



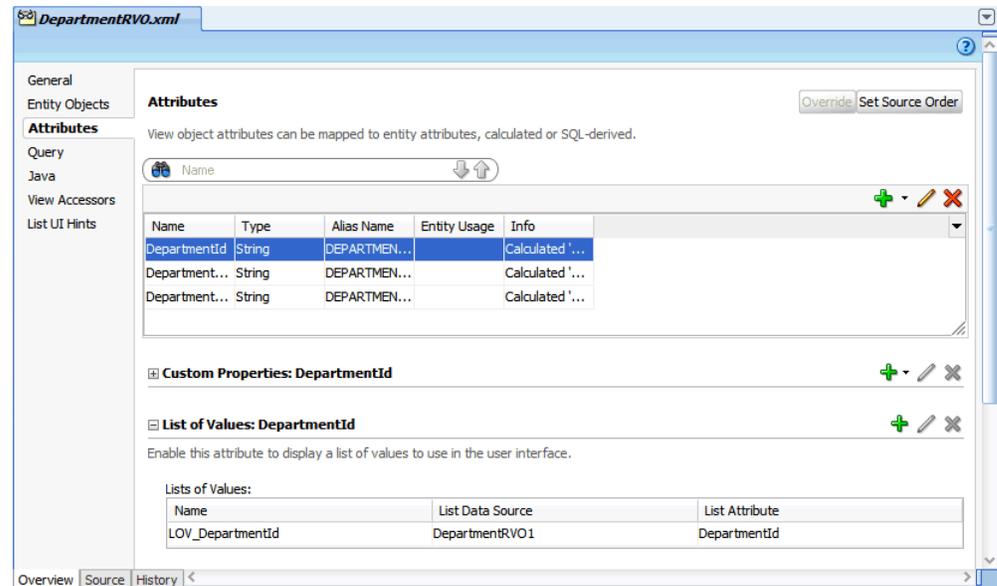
8. In the **List Search** section, select **Use DepartmentRVOCriteria** in the **Include Search Region** field.

Figure 2–15 Create List of Values Window



9. Select the **Query List Automatically** check box.
10. Click **OK**.

Figure 2–16 DepartmentRVO.xml Configuration Tab – Attributes Page



Creating the Class View Object

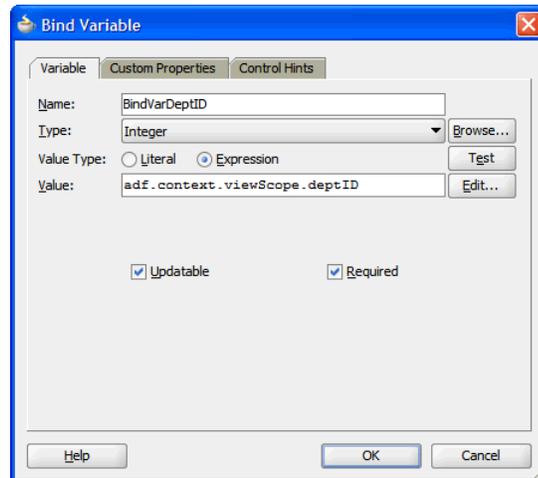
Consider a scenario where a user needs to also select a Class from a list of values and only the classes relevant to the selected Department must be included in the list. This can be achieved by defining a view object for the class in a similar manner as the Department view object (see [Creating the Department View Object](#)), but with the following differences:

- Create a read-only view object **ClassRVO** in the **oracle.retail.merch20.microApps.createRegItem.vo** package.
- Set up the following query for the view object:

```
SELECT class AS CLASS_ID,
       class_name AS CLASS_NAME,
       concat( concat(class, '-'), class_name) AS CLASS_SEARCH
FROM class
WHERE dept =:BindVarDeptID
UNION
SELECT null as CLASS_ID,
       null as CLASS_NAME,
       null as CLASS_SEARCH
FROM class
```

Note: The BindVarDeptID bind variable provides the department ID at run time.

- On the **Query** page, click the **Plus (+)** button in the **Bind Variable** section. The **Bind Variable** window appears.

Figure 2–17 Bind Variable Window

- a. In the **Bind Variable** window, under the **Variable** tab, enter the bind variable name as the same name used in the **query (BindVarDeptID)**.
 - b. Select **Integer** in the **Type** field.
 - c. Select the **Expression** option in the **Value Type** field.
 - d. Enter **adf.context.viewScope.deptID** in the **Value** field.
 - e. Click **OK**.
- In the **ClassRVO** view object, set up the **classId** attribute for LOV similar to the **departmentId** attribute in the **DepartmentRVO** view object.

The bind variable used in this view object supplies values at run time. This value comes from the **adf.context.viewScope.deptID** expression. This expression returns a **deptID** parameter set in a view scope object. This parameter will be set in the user interface managed bean when the user selects the department ID. For more information, see [Creating the User Interface](#).

Note: The view objects discussed above are for illustration purposes only. You can now create the rest of the view objects referring to these examples. Guidelines and hints to create all other view objects used in this application are provided in the appendix below. For more information, see [View Objects](#).

Internationalization of Labels

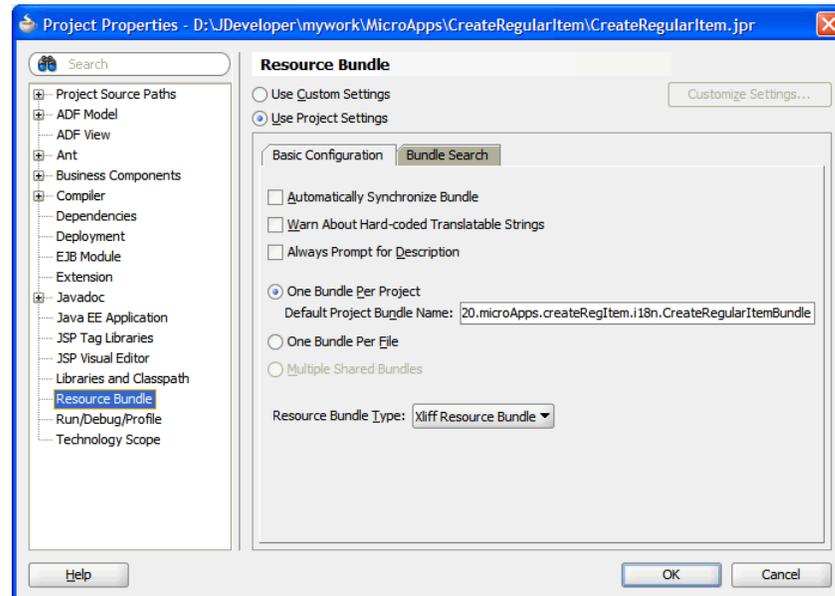
The **departmentId** attribute in the **DepartmentRVO** view object and **classId** attribute in the **ClassRVO** view object will be used in the user interface as described in the following sections. Before using them in the user interface, you must first set up labels for these in the view object itself. This way these labels are used in the user interface when ever the view object is used.

This section describes how you can set up a label for the **DepartmentRVO** view object and also internationalize it using the declarative way available in JDeveloper. You can follow the same instructions to set up labels for the **ClassRVO** view object and all other view objects in the application.

To set up a label for the DepartmentRVO view object and internationalize it:

1. In the **Application Navigator**, right-click on the **CreateRegularItem** project, and click **Project Properties**.
2. In the **Project Properties** window, select **Resource Bundle** in the left navigation pane.

Figure 2–18 Project Properties Window

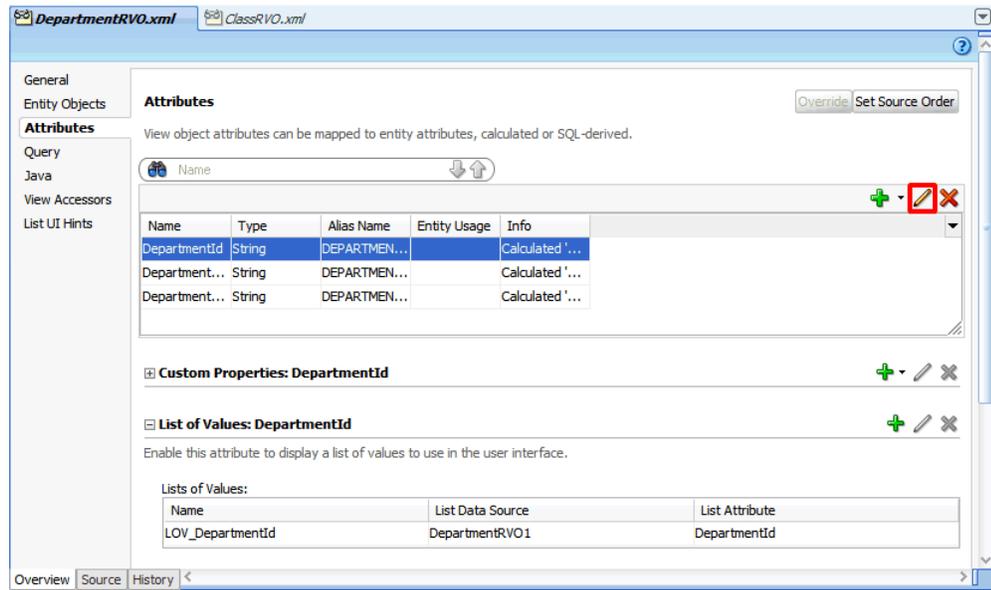


3. In the **Basic Configuration** tab, enter the following information:
 - Select the **One Bundle Per Project** option.
 - Enter **oracle.retail.merch20.microApps.createregItem.i18n.CreateRegularItemBundle** in the **Default Project Bundle Name** field.

Note: The package name includes **i18n** to indicate the location where the bundle is created.

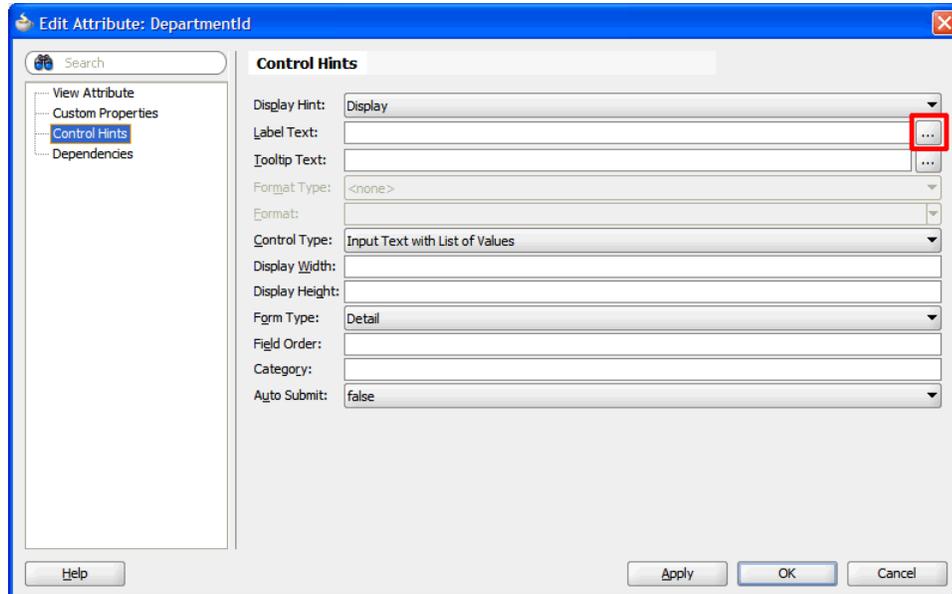
- Select **Xliff Resource Bundle** in the **Resource Bundle Type** field.
4. Click **OK**.
 5. Open the **DepartmentRVO** view object (**DepartmentRVO.xml** tab).
 6. On the **DepartmentRVO.xml** tab, click **Attributes**.

Figure 2–19 DepartmentRVO.xml Configuration Tab – Attributes Page

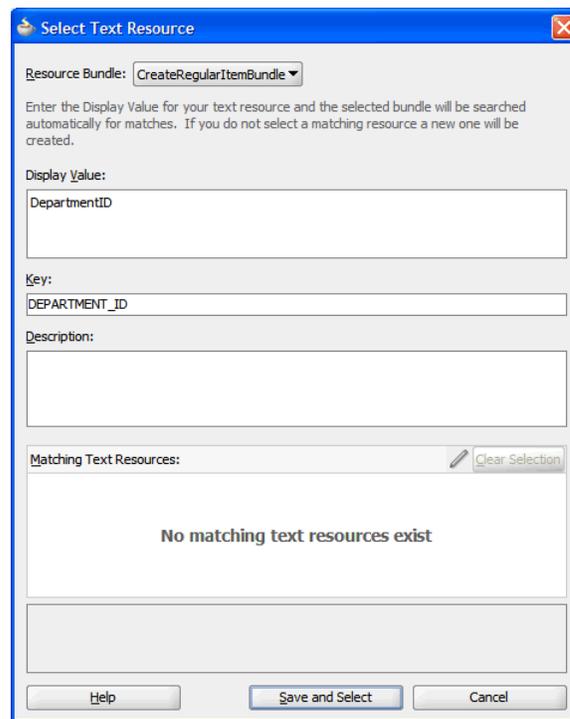


7. On the **Attributes** page, select the **DepartmentId** attribute, and then click the **Edit (Pencil)** button. The **Edit Attribute** window appears.

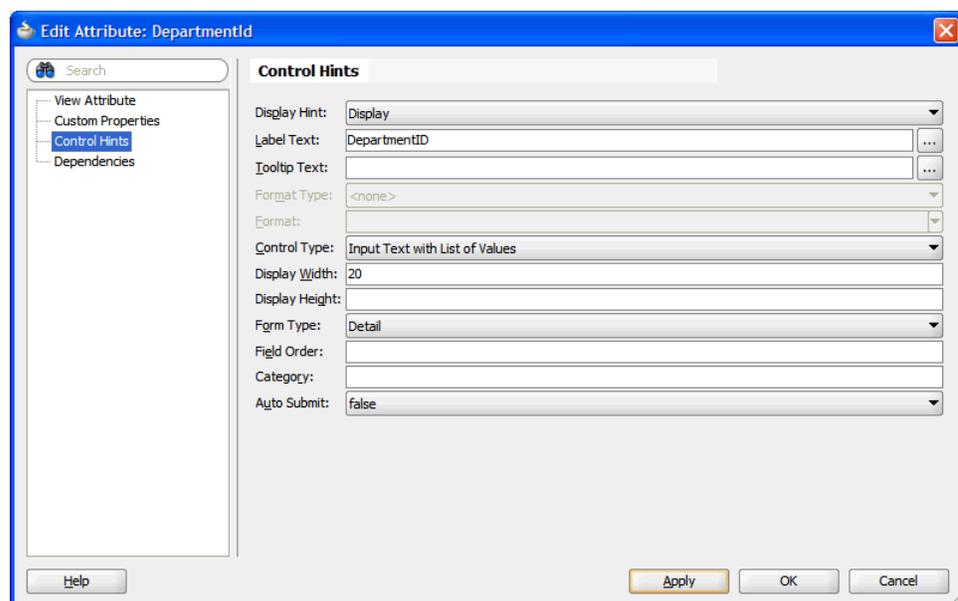
Figure 2–20 Edit Attribute: DepartmentId Window



8. In the **Edit Attribute** window, click **Control Hints** in the left navigation pane.
9. Click the **Browse** button next to the **Label Text** field. The **Select Text Resource** window appears.

Figure 2–21 Select Text Resource Window

10. In the **Select Text Resource** window, enter a relevant label name in the **Display Value** field. A corresponding entry will be added in the resource bundle automatically.
11. Click **Select**. The **Edit Attribute** window appears. Notice that the **Label Text** field now displays the label you set up.

Figure 2–22 Edit Attribute: DepartmentId Window

12. Enter a relevant width (for example, 20) in the **Display Width** field. This will be used for the width of the corresponding user interface component.
13. Click **Apply**, and then click **OK**.

The `oracle.retail.merch20.microApps.createRegItem.i18n` package now includes a `CreateRegularItem.xlf` resource bundle file.

To view an illustration of this resource bundle file, refer to the sample code available along with this tutorial. For more information, see [Accessing the Sample JDeveloper Project](#).

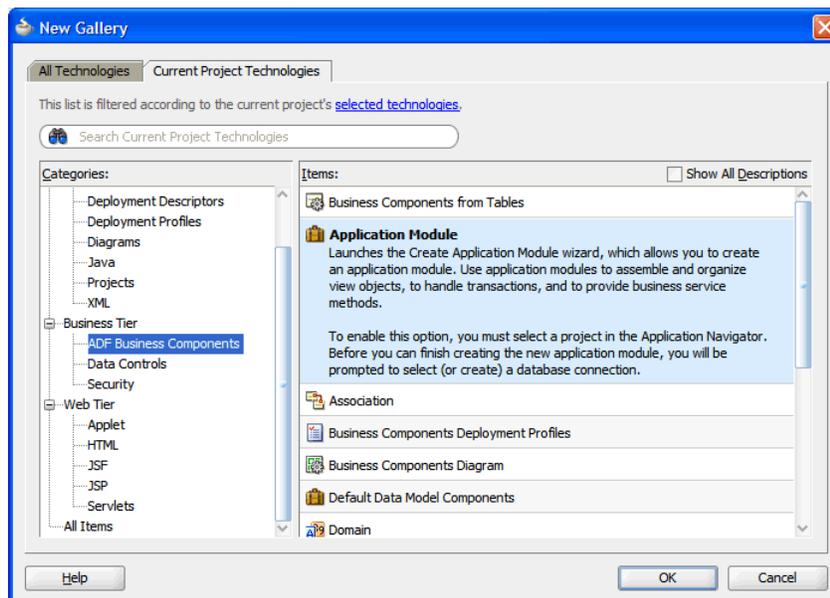
Adding View Objects to the Application Module

The application module represents a collection of all view objects and services that are used together to complete a unit of work. This section describes how you can create an application module and include all the view objects created for the project.

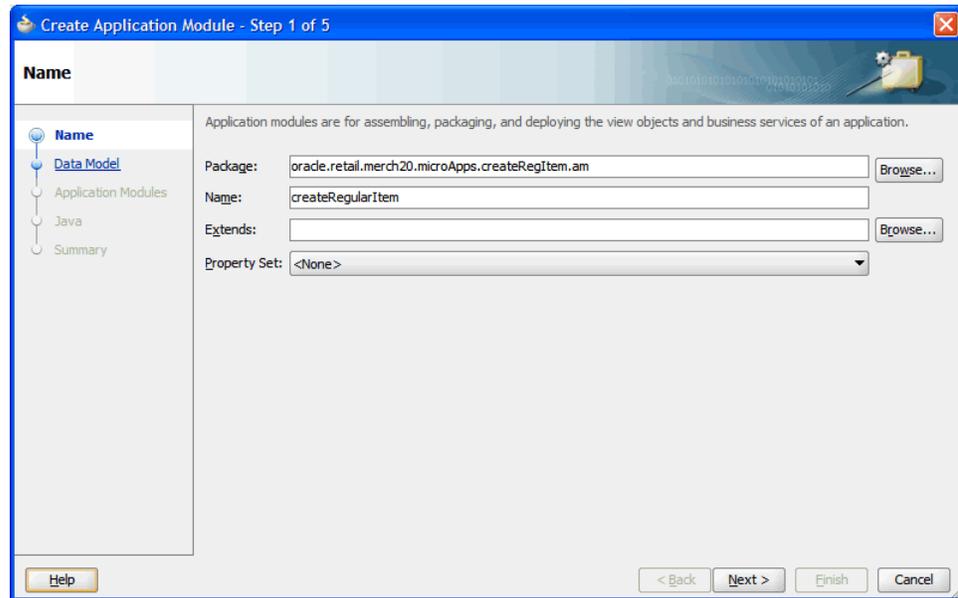
To create an application module and add view objects to the application module:

1. In the **Application Navigator**, expand `CreateRegularItem`.
2. Under `CreateRegularItem`, right-click on the **Application Sources** folder, and click **New**. The **New Gallery** wizard appears.

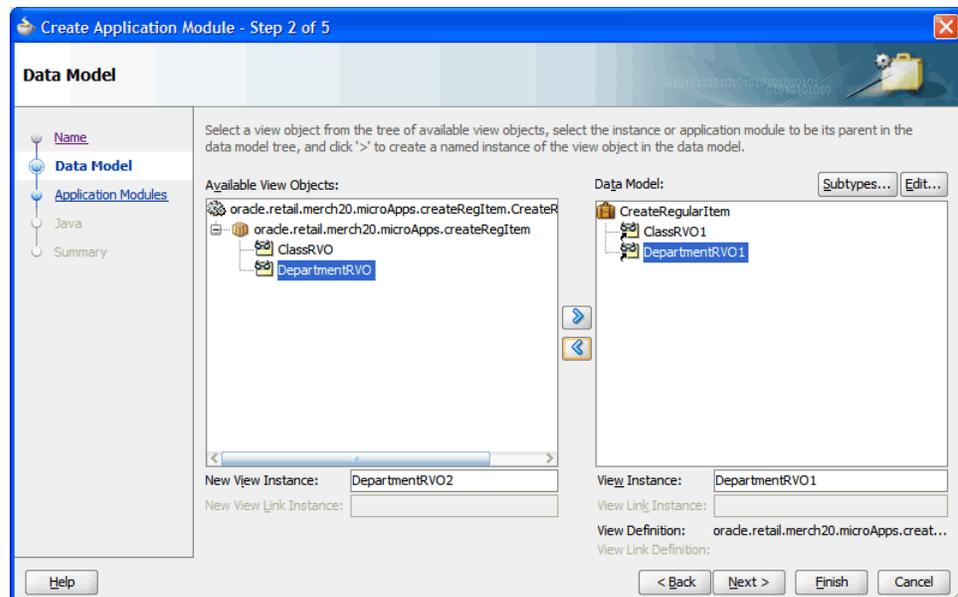
Figure 2–23 *New Gallery Wizard*



3. In the **New Gallery** wizard, click **ADF Business Components** under **Business Tier** in the left navigation pane.
4. Under **Items** area, click **Application Module**, and then click **OK**. The **Create Application Module** wizard appears.

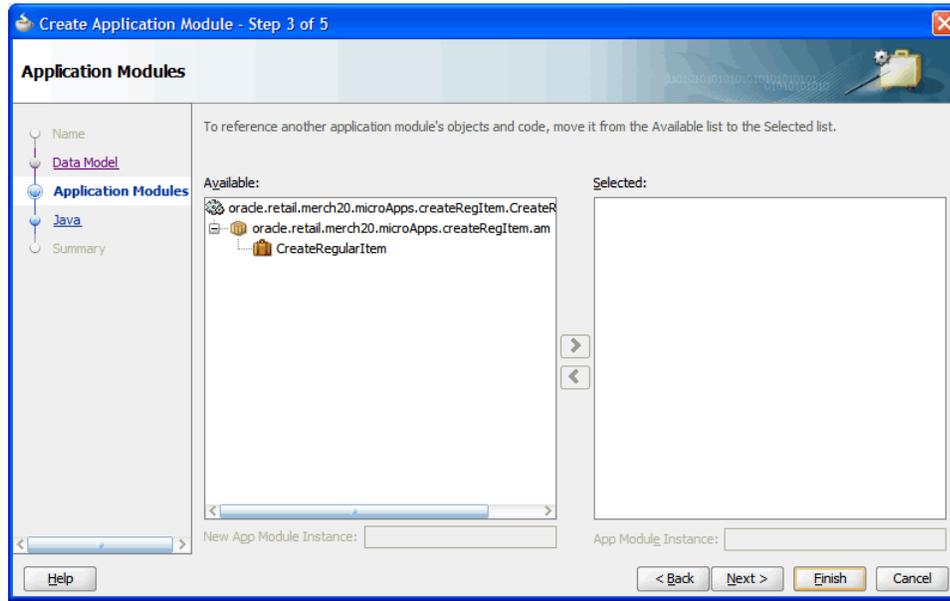
Figure 2–24 Create Application Module Wizard – Name Screen

5. On the **Name** screen, enter the following information:
 - Enter **oracle.retail.merch20.microApps.createRegItem.am** in the **Package** field.
 - Enter **CreateRegularItem** in the **Name** field.
 - Leave other fields As Is.
6. Click **Next**. The **Data Model** screen appears.

Figure 2–25 Create Application Module Wizard – Data Model Screen

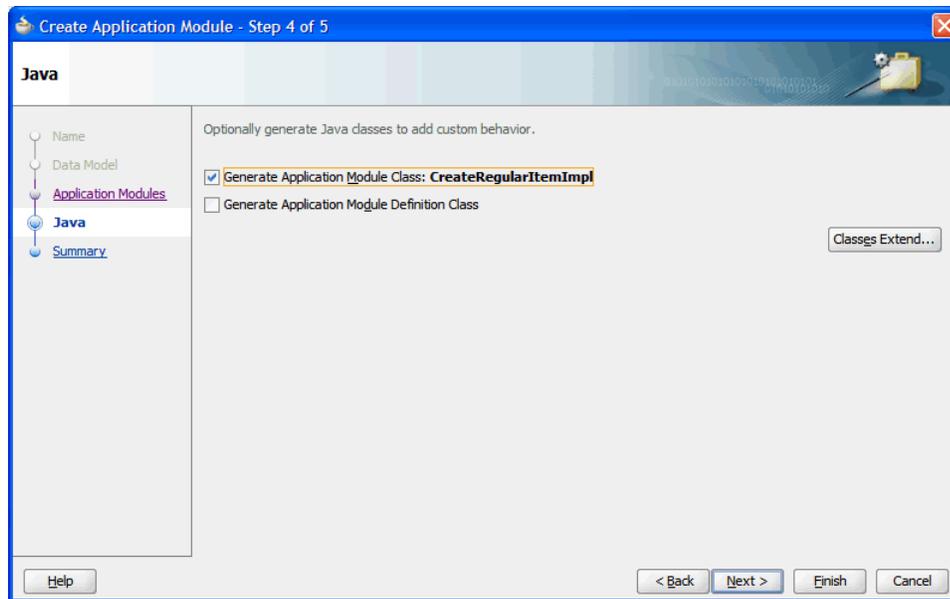
7. On the **Data Model** screen, select all the view objects listed in the **Available View Objects** section and move them to the **Data Model** section. If by now you have created all the required view objects, they will be available in the **Available View Objects** list. You can complete this step after creating the application module.
8. Click **Next**. The **Application Modules** screen appears.

Figure 2–26 Create Application Module Wizard – Application Modules Screen



9. Leave this screen As Is and click **Next**. The **Java** screen appears.

Figure 2–27 Create Application Module Wizard – Java Screen



10. On the **Java** screen, select the **Generate Application Module Class** check box.

11. Click **Finish**.

Note: We chose to generate the application module class because this is the place where we will call the PL/SQL package for the business process. An application module implementation class has access to the database connection object which is used to call the PL/SQL procedure. For more information, see [Calling X API in the Micro-Application](#).

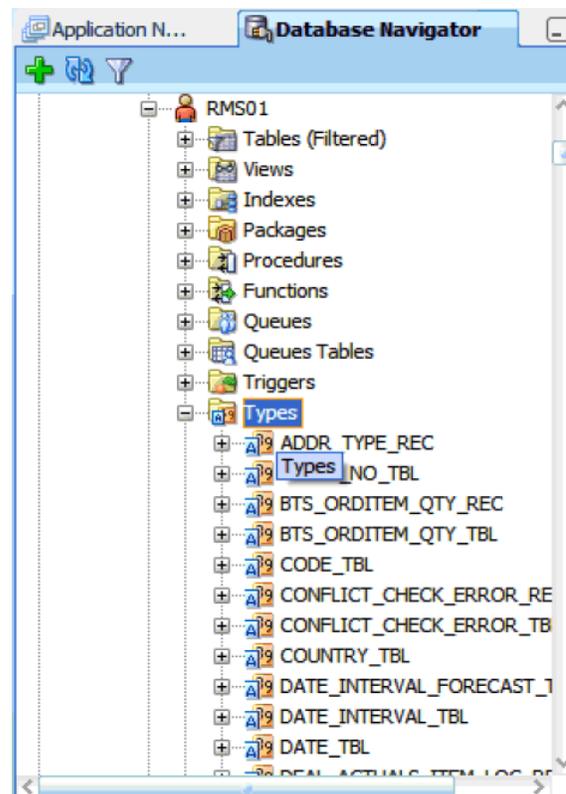
Creating SQL Object Types in Java

X APIs are PL/SQL packages that are used to initiate and complete a business process in RMS by external applications. These PL/SQL packages take in SQL objects, called as RIB objects, as parameters. These SQL objects are defined as types in RMS database schema. Before calling the X API in Java, you must first have the corresponding Java objects for these SQL objects. This section describes how you can create such a SQL object type for the Create Regular Item Micro-Application.

To create a SQL object type:

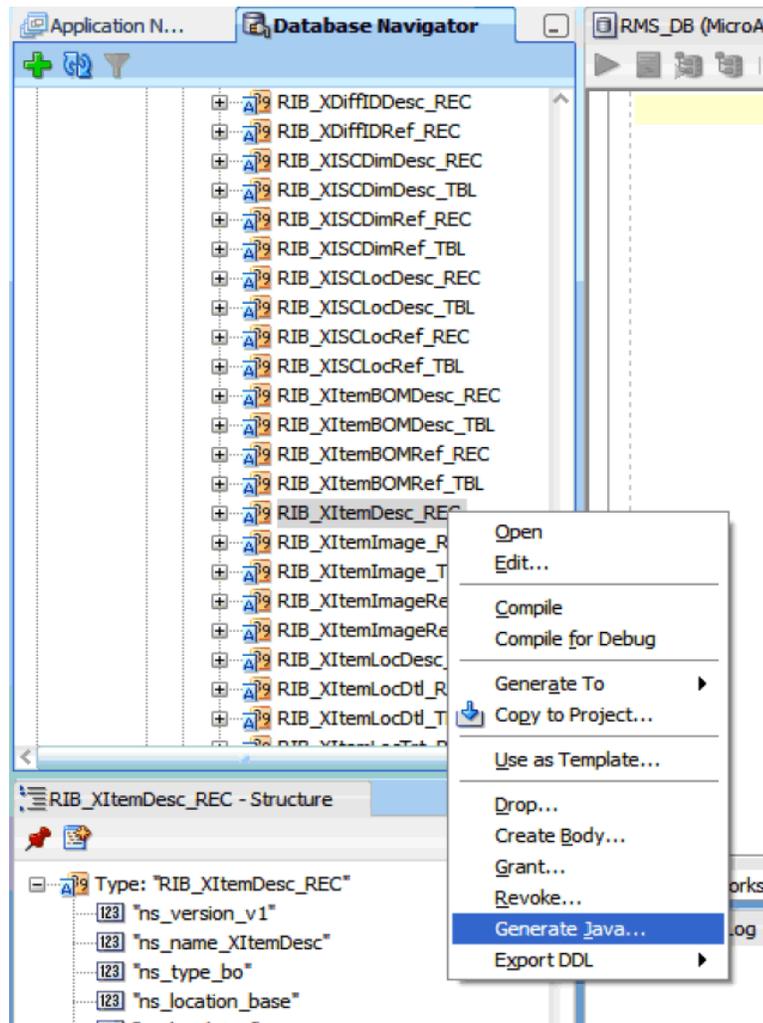
1. In JDeveloper, open the **Database Navigator** (From the **View** menu, point to **Database**, and then click **Database Navigator**).
2. Under **MicroApps**, open/expand the RMS database connection (**RMS_DB**).
3. Under **RMS_DB**, expand **Types**.

Figure 2–28 Database Navigator Pane



4. Under **Types**, search for **RIB_XItem_Desc_REC**. This is the RIB object that is required to be passed as the parameter to the X API for creating an item. It is composed of several other SQL types.
5. Right-click on this type, and click **Generate Java**.

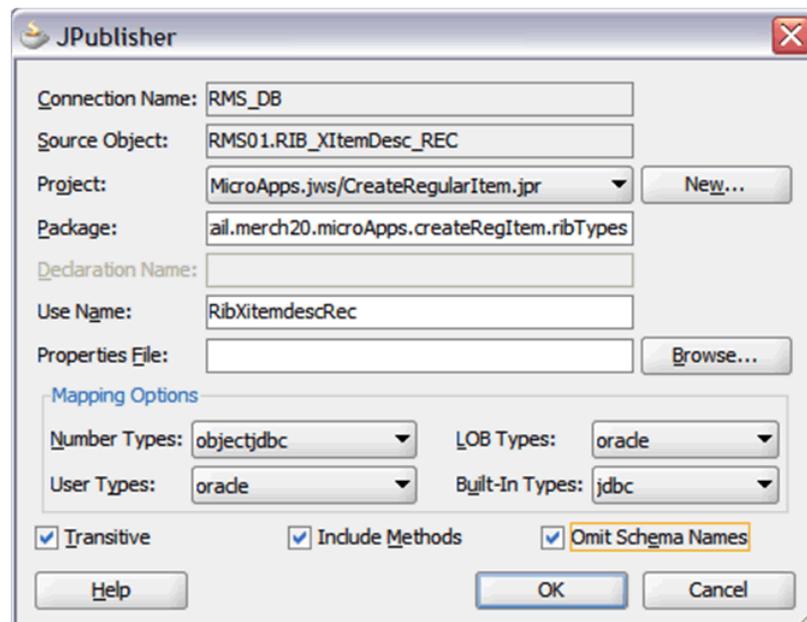
Figure 2–29 Generate Java Option in Database Navigator



The **JPublisher** wizard appears.

6. In the **JPublisher** wizard, ensure that the selected project is the current project.

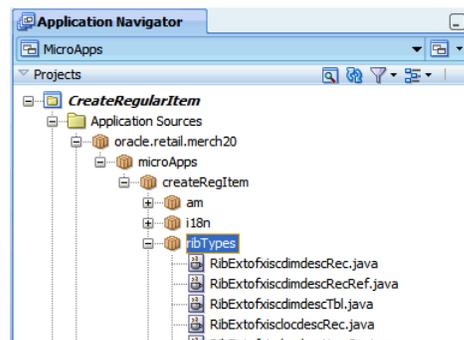
Figure 2–30 JPublisher Wizard



Note: In the JPublisher wizard, select the **Omit Schema Names** option if you do not want the generated class files to contain the schema name. This option will work only when you are connected to the database as the user who owns the RIB schema.

7. Enter `oracle.retail.merch20.microApps.createRegItem.ribtypes` in the **Package** field.
8. Click **OK**. This may take a few minutes to complete. Once complete, you will be able to see all the related objects formed in the package you specified.

Figure 2–31 Application Navigator with SQL Objects



Note: It requires a functional knowledge of RMS to be able to identify the RIB Object used in the procedure call. For more information, refer to the Oracle Retail Merchandising System Operations Guide, Release 13.1 (Volume 2 – Message Publication and Subscription Designs). Once the top level object is identified, all objects referenced by it would be automatically generated as Java.

Creating the User Interface

This section describes how you can create the user interface using ADF Faces. The user interface will include a Form where the user will enter all the required information to create an item. This user interface will also include code that will capture the values entered by the user. The user interface will be created as a JSF page fragment. This page fragment will then be used to create the ADF Task Flow. To create the user interface, you must complete the following tasks:

1. [Creating a Page Fragment](#)
2. [Constructing the User Interface](#)
3. [Setting Up a Managed Bean](#)

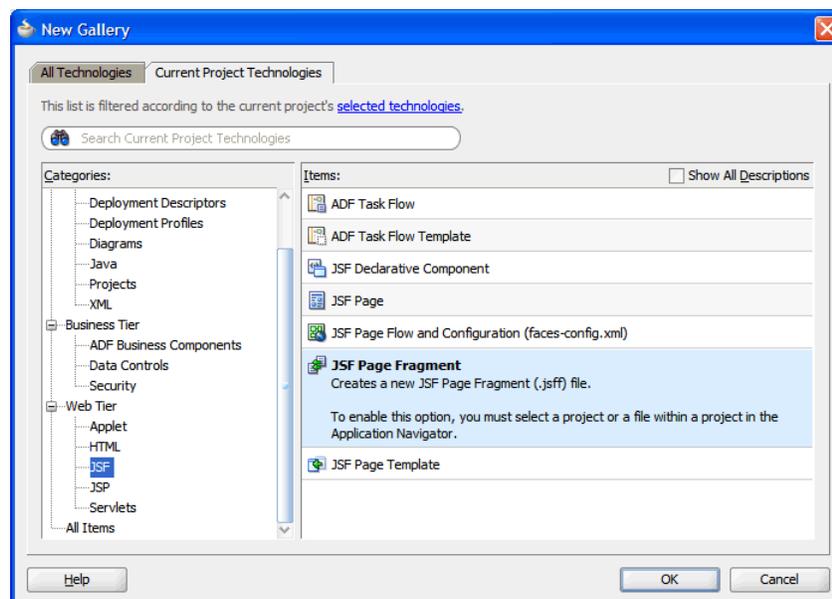
Note: This user interface form will contain several user interface components. For illustration purposes, this section covers the usage of the Department LOV and Class LOV input fields. To view an illustration of this page fragment along with the corresponding backing bean and page definition file, refer to the sample code available along with this tutorial. For more information, see [Accessing the Sample JDeveloper Project](#).

Creating a Page Fragment

To create a page fragment:

1. In the **Application Navigator**, expand the **CreateRegularItem** project.
2. Right-click on the **Web Content** folder, and click **New**. The **New Gallery** wizard appears.

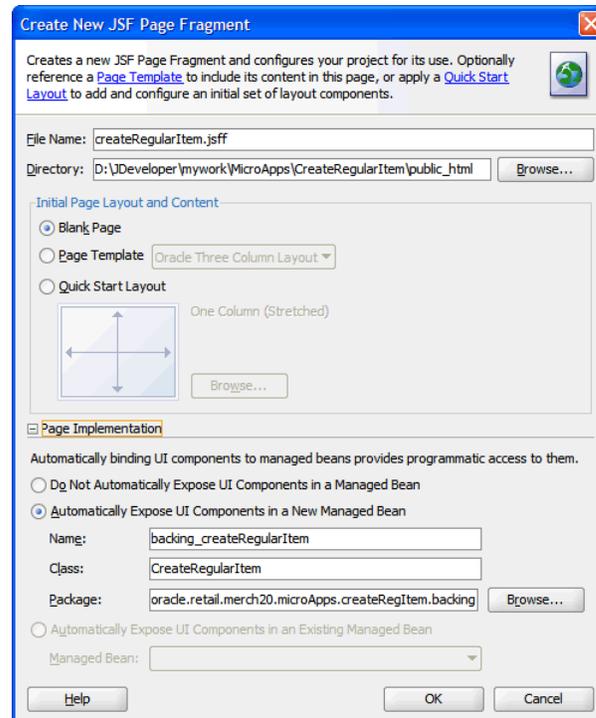
Figure 2–32 *New Gallery Wizard*



3. In the **New Gallery** wizard, click **JSF** under **Web Tier** in the left navigation pane.

- Under **Items** area, click **JSF Page Fragment**, and then click **OK**. The **Create New JSF Page Fragment** window appears.

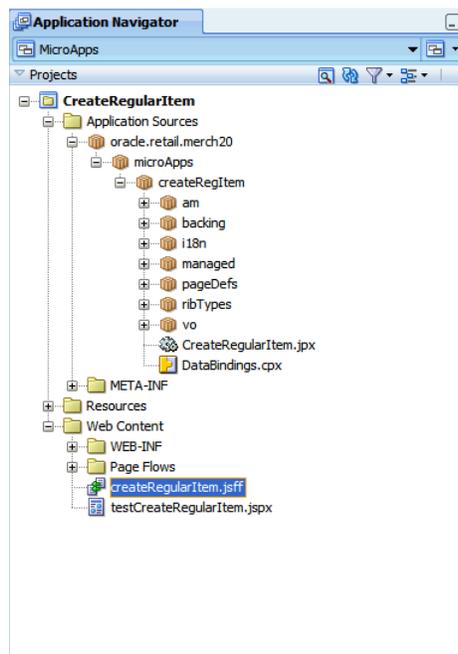
Figure 2–33 Create New JSF Page Fragment Window



- In the **Create New JSF Page Fragment** window, enter `createRegularItem.jsff` in the **File Name** field.
- Expand the **Page Implementation** section, and select the **Automatically Expose UI Components in a New Managed Bean** check box.
- Click **OK**.

A new page fragment is created in the **Web Content** directory. The corresponding backing bean is created in the `oracle.retail.merch20.microApps.createRegItem.backing` package under the **Application Sources** for the project.

Figure 2–34 Application Navigator Pane



Note: Backing bean is useful to define any custom user interface functionality that you may require. It is a good practice to have one backing bean per page.

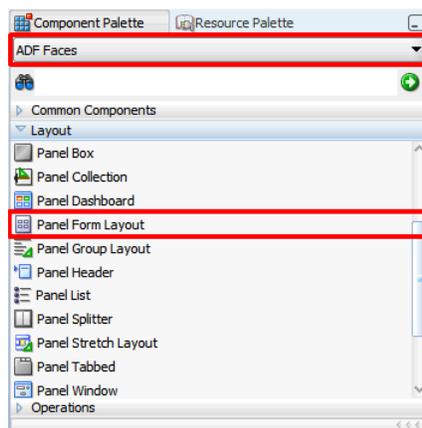
Constructing the User Interface

Once the page fragment is created, you can now open the page to construct the user interface.

To construct the user interface:

1. Under the **Component Palette**, from the **ADF Faces**, drag and drop the **Panel Form Layout** to the **Design** view of the **Editor** window for the **createRegularItem.jsff** file.

Figure 2–35 Component Palette Pane

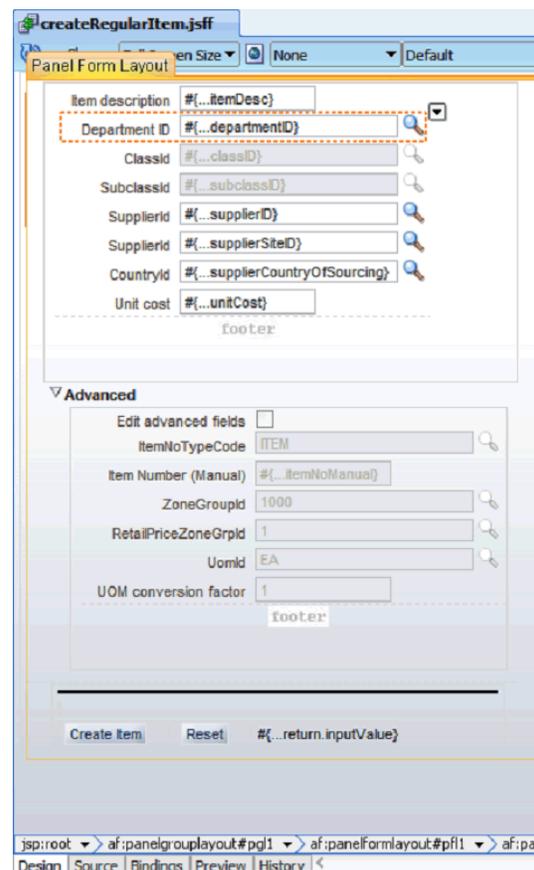


2. From the **Application Navigator**, expand **Data Controls** navigator.
3. In the **Data Controls** navigator, expand **CreateRegularItemDataControl**.
4. Under **CreateRegularItemDataControl**, expand the **DepartmentRVO1** view object instance.
5. Drag and drop the **DepartmentID** attribute to the **Panel Form Layout** you added in the step 1. A menu appears as soon as you drop the attribute to the page.
6. On the menu, point to **List of Values**, and then click **ADF LOV Input**. An input box will appear on the page with a **Search** icon on the right.
7. Right-click anywhere on the page and click **Go to Page Definition**.
In the **Binding and Executables** tab of the page definition, observe that the **DepartmentID** binding is created along with a **DepartmentRVO1Iterator** executable.
8. In a similar manner, drag and drop the **ClassId** attribute from the **ClassRVO1** view object to the **Panel Form Layout**.

Repeat this process for all the input components needed in the user interface. To view or copy an illustration of this page along with the corresponding backing bean and page definition file, refer to the sample code available along with this tutorial. For more information, see [Accessing the Sample JDeveloper Project](#).

If you choose to copy the code, your page will look like the figure below.

Figure 2–36 createRegularItem.jsff Page in the Editor Window



Setting Up a Managed Bean

Before proceeding further with the page, you must define a managed bean in the application. This managed bean will be used to set and store the values that the user will enter in the user interface components on the page. For example, as the user selects the department ID for the item using the ADF LOV input component create above, the value for the department ID will be captured in this managed bean.

To set up a managed bean:

1. Create a new Java class:
 - a. From the **Application Navigator**, right-click on the **CreateRegularItem** project, and click **New**. The **New Gallery** wizard appears.
 - b. In the **New Gallery** wizard, click **General** in the left navigation pane.
 - c. Under the **Items** area, click **Java Class**, and then click **OK**. The **Create Java Class** window appears.
 - d. In the **Create Java Class** window, enter the following information:
 - Enter **oracle.retail.merch20.microApps.createRegItem.managed** in the **Package** field.
 - Enter **Item** in the **Name** field.
 - e. Click **OK**.
2. Define a field as **private String departmentID** in the class.
3. Generate the getter and setter method for the **departmentID** field.

This field will be used to capture and store the value of the department ID as entered in the user interface. Similarly, you must define fields corresponding to all input components on the page whose value you want to capture. The following code snippet illustrates how you can capture the department ID value when entered by the user. To view the full source code for this class, refer to the sample code available along with this tutorial. For more information, see [Accessing the Sample JDeveloper Project](#).

```
//String field to store department value.
private String departmentID;
// ... other fields

/*Setter method for departmentID.
   This method will get called when user enters some value in the ADF LOV input
   component for Department ID.
   The entered value will be available as the method argument "departmentID".
   This method sets the value of the department ID in the View scope object
   to be used by the bind variable used in the "class" view object.
   Then this method gets hold of the "class" view object iterator from the page
   definition,
   and execute the query on the class view object iterator.
*/

public void setDepartmentID(String departmentID) {

    //retriving the value entered for departmentID
    this.departmentID = departmentID;
    //setting the value in a view scope object
    ADFContext.getCurrent().getViewScope().put("deptID",new
    Integer(this.departmentID) );
    //getting the Class view object iterator
```

```

        DCIteratorBinding dcib = findIterator("ClassRV01Iterator");
        //executing the query associated with the view object
        dcib.executeQuery();

    }

    /*Getter method for departmentID.
    */

    public String getDepartmentID() {
        return departmentID;
    }

    /*This is a utility method that returns the iterator instance from the binding
    context
    Whose name is provided as the parameter.
    */

    private DCIteratorBinding findIterator (String iteratorNameInPageBinding){

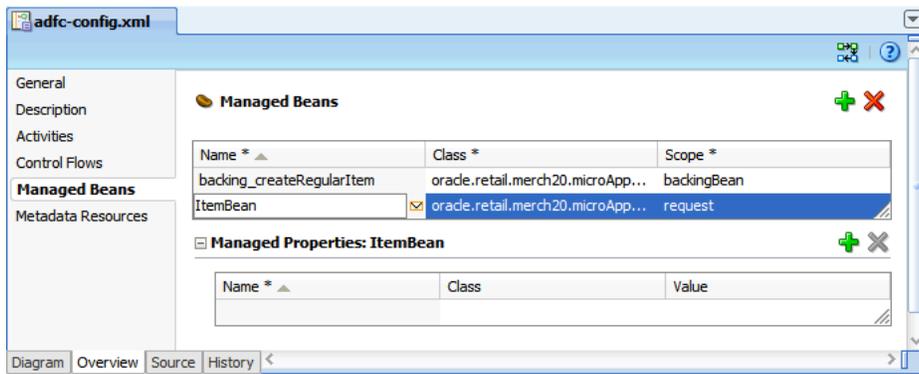
        //getting binding container object
        FacesContext facesContext = FacesContext.getCurrentInstance();
        Application app = facesContext.getApplication();
        ExpressionFactory elFactory = app.getExpressionFactory();
        ELContext elContext = facesContext.getELContext();
        ValueExpression valueExp =
        elFactory.createValueExpression(elContext, "#{bindings}",
        Object.class);
        DCBindingContainer dcBindingContainer =
        (DCBindingContainer)valueExp.getValue(elContext);

        //getting iterator binding from the binding container object
        DCIteratorBinding iter =
        dcBindingContainer.findIteratorBinding(iteratorNameInPageBinding);
        return iter;
    }
}

```

4. In the **Application Navigator**, under **Web Content**, expand the **WEB-INF** folder.
5. Open the **adfc-config.xml** file.
6. In the **Overview** tab of the **adfc-config.xml** file, click **Managed Beans**.
7. On the **Managed Beans** page, define a managed bean by clicking the **Plus (+)** icon and enter the following information:
 - Under the **Name** column, enter **ItemBean**.
 - Under the **Class** column, enter **oracle.retail.merch20.microApps.createRegItem.managed.Item**.
 - Under the **Scope** column, select **Request**.

Figure 2–37 *adfc-config.xml Configuration Tab – Managed Beans Page*

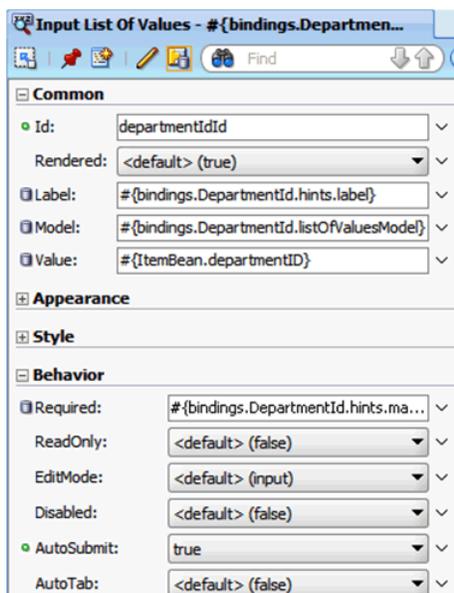


8. Save the file.

Note: Now that the Java Class file is declared as a managed bean, all the fields defined in our bean can be retrieved using EL expressions during run time.

9. Open the page fragment (**createRegularItem.jsff**) you created above.
10. In the **Property Inspector**, clear the **Value** property for the **DepartmentID** ADF LOV Input Component.
11. Bind this **Value** property to the **departmentID** field defined above in the **ItemBean** managed bean using the EL expression **#{ItemBean.departmentID}**.
12. In the **Property Inspector** for the component above, set the **AutoSubmit** property value to **true**.

Figure 2–38 *Property Inspector Pane*



Note: Setting the AutoSubmit property to true ensures that the value entered for the department ID will be available to the server in the same request cycle without explicitly submitting the page (partial page submit).

Once you complete these steps, there will be two "ADF LOV Input Text" components in the Panel Form Layout (Department and Class). If the user selects a value for the department ID using the "DepartmentID ADF LOV Input Component", the setter method "setDepartmentID" will be called, which will execute the custom code show above. This code will set the "deptID" property in the view scope, from where it will be provided to the Bind Variable used in the ClassRVO view object. Executing the query of the ClassRVO view object, you will be able to display the list of all classes in the selected department.

You will need to construct input UI elements for the other view objects using the same logic as above, if the bind variable dependency exists between two view objects. As the user enters the value, it will be retrieved in the ItemBean managed bean and stored for calling the business process. Once all input elements are created in the user interface, the next step is to provide a button to initiate the item creation process.

Calling X API in the Micro-Application

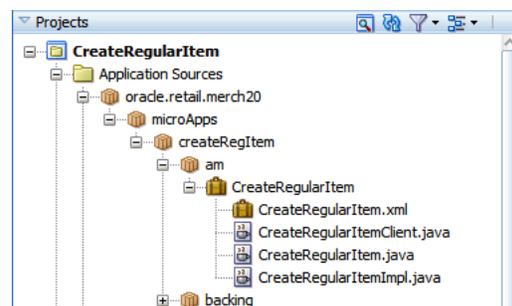
In the sections above, you have created the base for the application user interface. You can now form a user interface where the users can see the data from the RMS database schema and enter data by selecting from lists. You must now provide the business service call for creating the regular item in RMS. This section describes how you can set up this service call.

The real business functionality is provided by a PL/SQL procedure in the RMS database schema (X API called xitemcre). You must call this procedure from Java. Since you have already created an application module implementation class, this is the place where you will call the procedure.

To set up the service call in the Micro-Application:

1. In the **Application Navigator**, expand the **Application Sources** folder under the **CreateRegularItem** project.
2. Under **Application Sources**, expand **oracle.retail.merch20**, **microApps**, **createRegItem**, and then **am**.
3. Under **am**, expand **CreateRegularItem**.

Figure 2–39 *CreateRegularItemImpl Java File in the Application Navigator*



4. Open the **CreateRegularItemImpl** Java Class and add the following method in this class:

```

/**
 * This is the custom method which will be exposed to the client through
 DataControl.
 * This method calls the business service in RMS to create an item in RMS
 schema.
 * The business service used is a PL/SQL procedure 'RMSSUB_XITEM.CONSUME' in RMS
 schema.
 * @param inputValuesForItemCre Map -Contains all parameters required for the
 business service.
 * @return String -Status of the business service call.
 */
public String createRegularItem(Map inputValuesForItemCre){

    CallableStatement callableStmt = null;
    String rmsUser = getDBTransaction().getConnectionMetadata().getUserName();
    String callableStatement = "begin "+rmsUser+".RMSSUB_XITEM.CONSUME(?,?,?,?);
end;";
    try{
        callableStmt =
getDBTransaction().createCallableStatement(callableStatement,0);
        callableStmt.registerOutParameter(1, Types.VARCHAR);
        callableStmt.registerOutParameter(2, Types.VARCHAR);
        callableStmt.setString(1, "");
        callableStmt.setString(2, "");
        callableStmt.setObject(3,
getSqlObjForRegularItemCreation(inputValuesForItemCre));
        callableStmt.setString(4, "xitemcre");
        callableStmt.executeUpdate();
        String status = "";
        if(callableStmt.getString(1).equals("S")){
            status = "<big>SUCCESS</big>";
        }else{
            status = "<big>ERROR</big> : "+callableStmt.getString(2);
        }
        getDBTransaction().commit();
        return status;
    }catch(SQLException e){
        throw new JboException(e);
    }finally{
        try{
            callableStmt.close();
        }catch(SQLException e){
            throw new RuntimeException(e);
        }
    }
}

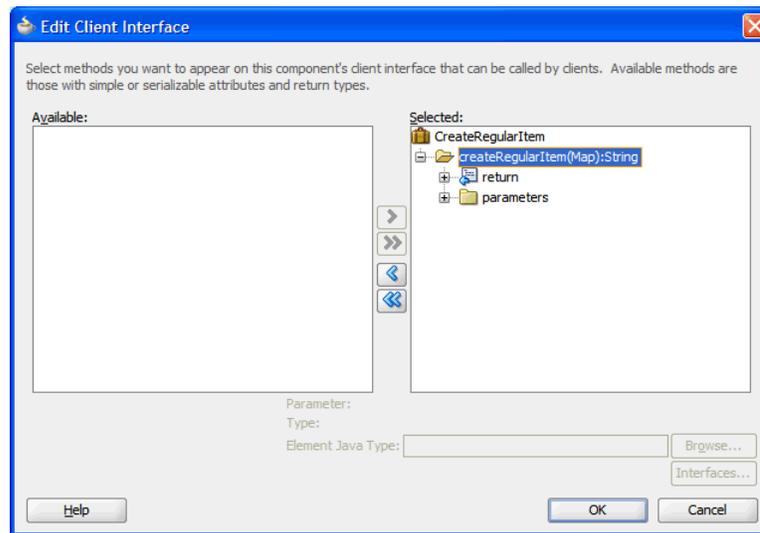
/**This method returns the RIB Object required for "xitemcre" X API.
 The RIB object has to be populated with the user input and passed to
 PL/SQL (X API) procedure.
 */
private Object getSqlObjForRegularItemCreation( Map inputValuesForItemCre){
    //Code for this method is not included here for clarity.
    //The code has intentional syntax error.
}

```

Note: To view an illustration of the `CreateRegularItemImpl` Java Class, refer to the sample code available along with this tutorial. For more information, see [Accessing the Sample JDeveloper Project](#).

5. Double-click the **CreateRegularItem** application module to open it in the **Editor** window.
6. On the **CreateRegularItem.xml** tab, click **Java**.
7. On the **Java** page, click the **Edit (Pencil)** button next to the **Client Interface** section. The **Edit Client Interface** window appears.

Figure 2–40 *Edit Client Interface Window*



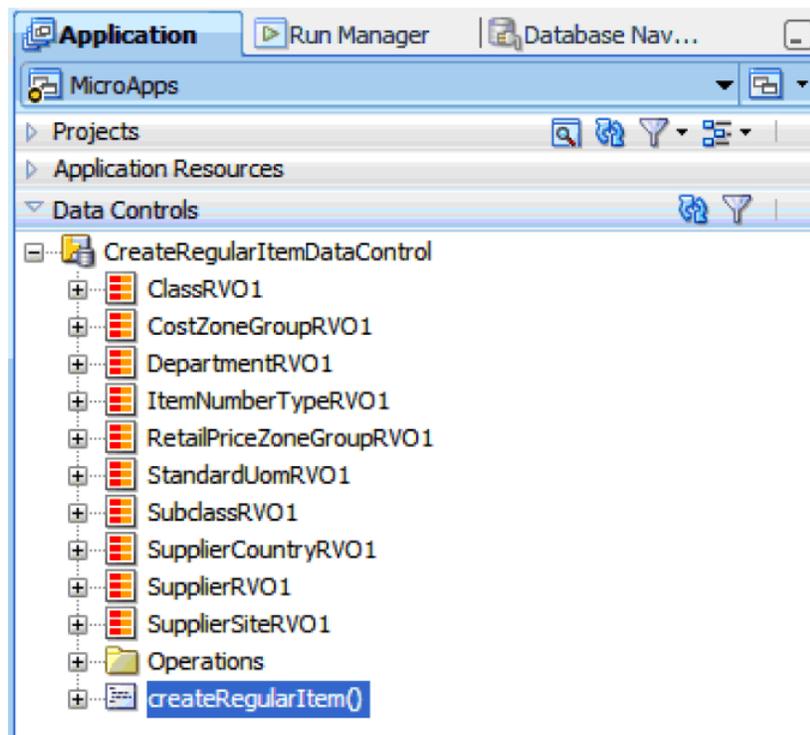
In the **Edit Client Interface** window, the `createRegularItem` method appears in the **Available** list.

8. Move the `createRegularItem` method under the **Selected** list, and then click **OK**.

Note: Once you complete this step, this method will be available in the application module data control. You can drag it to the user interface as a command button.

9. In the **Data Control Navigator**, expand **CreateRegularItemDataControl**. You will be able to see the method added above.

Figure 2-41 Data Controls Navigator Pane



10. You can drag and drop this method control as a command button on the JSF page fragment `createRegularItem.jsff` created above. This is illustrated in detailed in the section below.

Creating a Business Service User Interface Button

In the section above, the `createRegularItem` method of `CreateRegularItemImpl` class was exposed in the data control. This method takes in `java.util.Map` as a parameter and supplies as key value pairs, all the parameters required to create the item. This key value pair contract is defined in the class itself as `String` constants. For example, `DEPARTMENT_ID` key should be present in the map with the value equal to the department id for the item to be created. Whenever this method is called, the `Map` parameter passed should contain all the keys defined in the contract with proper values. The keys required in the `Map` are listed below.

To create a business service user interface button, you can simply drag and drop this method from the data control to the JSF page as a button.

Before you create this button, you must complete the following steps:

1. From the `oracle.retail.merch20.microApps.createRegItem.managed` package in the **Application Navigator**, open the **Item** Java Class file. This class file must have the following fields, each corresponding to an input field in the user interface:

```
private String itemDesc;
private String departmentID;
private String classID;
private String subclassID;
private String supplierID;
private String supplierSiteID;
private String supplierCountryOfSourcing;
```

```

private String unitCost;
private String itemNoTypeCode;
private String itemNoManual;
private String costZoneID;
private String retailPriceZoneGrpID;
private String unitOfMeasure;
private String unitOfMeasureConversionFactor;

```

2. Add a new field **private Map formInputValues**. This field will be used to pass the **Map** parameter to the business service method you create in the previous section.
3. Add the following getter method for the **formInputValues**:

```

public Map getFormInputValues() {
    //putting all UI values in the form
    formInputValues.put("ITEM_DESC", itemDesc);
    formInputValues.put("DEPARTMENT_ID", departmentID);
    formInputValues.put("CLASS_ID", classID);
    formInputValues.put("SUBCLASS_ID", subclassID);
    formInputValues.put("SUPPLIER_ID", supplierID);
    formInputValues.put("SUPPLIER_SITE_ID", supplierSiteID);
    formInputValues.put("SUPPLIER_COUNTRY_OF_SOURCING",
supplierCountryOfSourcing);
    formInputValues.put("UNIT_COST", unitCost);
    formInputValues.put("ITEM_NO_TYPE_CODE", itemNoTypeCode);
    formInputValues.put("ITEM_NO_MANUAL", itemNoManual);
    formInputValues.put("COST_ZONE_ID", costZoneID);
    formInputValues.put("RETAIL_PRICE_ZONE_GRP_ID", retailPriceZoneGrpID);
    formInputValues.put("UNIT_OF_MEASURE", unitOfMeasure);
    formInputValues.put("UOM_CONVERSION_FACTOR",
unitOfMeasureConversionFactor);
    return formInputValues;
}

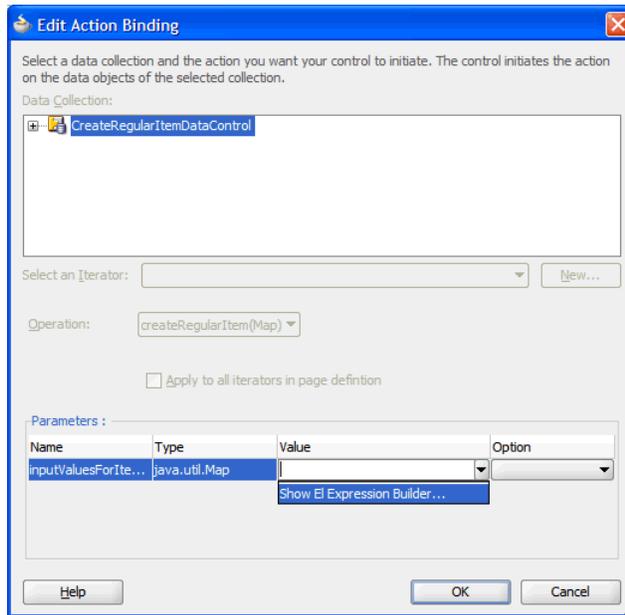
```

4. Open the **createRegularItem.jsff** page.

Note: If you have used the code listed for this page from the sample code, the business service button will already be present.

5. In the **Data Control Navigator**, expand **CreateRegularItemDataControl**, and then drag and drop the exposed method **CreateRegularItem(Map)** to the page as an **ADF Button**. The **Edit Action Binding** window appears.

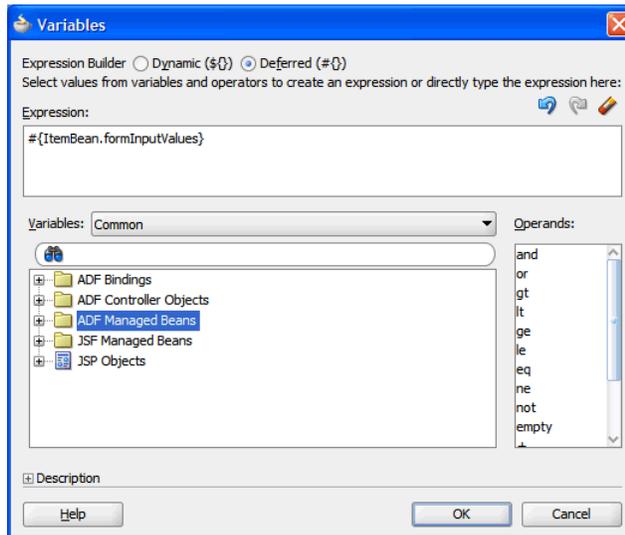
Figure 2–42 Edit Action Binding Window



As this method requires an input parameter, you will need to provide an EL expression to retrieve the parameter.

- In the **Edit Action Binding** window, under **Parameters**, select **Show EL Expression Builder** under the **Value** column. The **Variables** window appears.

Figure 2–43 Variables Window



- In the **Variables** window, enter `#{ItemBean.formInputValues}` in the **Expression** field.
- Click **OK** to go back to the **Edit Action Binding** window.
- Click **OK** on the **Edit Action Binding** window.

Observe that a button is created in the page. Whenever the user will click this button, the business service method in the `CreateRegularItemImpl` class will get called. The `ItemBean` managed bean's `formInputValues` field will be passed as the method parameter. This `Map` object will have all the required values for item creation as key value pairs following the agreed contract for keys.

PL/SQL Wrapper for Item Sequence Generator in RMS

As illustrated above, each item in RMS has a unique code. These codes can be of different types. These codes can be generated automatically by a PL/SQL procedure call in RMS. This procedure takes in the code type as argument and returns the unique code of that type. This procedure has the following form in the RMS database schema:

```
FUNCTION GET_NEXT(O_error_message IN OUT VARCHAR2,
  IO_item_no IN OUT ITEM_MASTER.ITEM%TYPE,
  I_item_type IN ITEM_MASTER.ITEM_NUMBER_TYPE%TYPE)
return BOOLEAN;
```

This procedure cannot be called directly from Java code due to the lack of support in the Oracle JDBC implementation for the `BOOLEAN` type returned from the procedure. As a work around, you can write a PL/SQL wrapper function that will take in the "item code type" as argument, calls the above PL/SQL procedure, retrieve the item code generated, and return the item code to the caller. This eliminates the `BOOLEAN` used in the original procedure. This wrapper procedure definition is as follows:

```
create or replace
PROCEDURE NEXT_ITEM_NUMBER (O_item_number IN OUT ITEM_MASTER.ITEM%TYPE,
I_item_type IN VARCHAR2,
O_error_message IN OUT VARCHAR2) IS

L_status BOOLEAN;

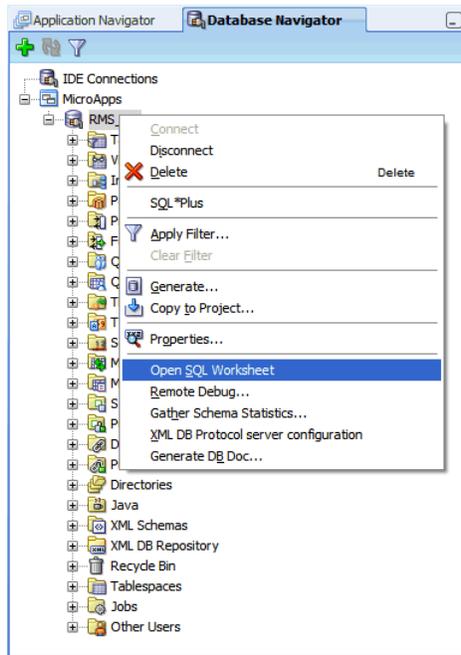
BEGIN
L_status := item_number_type_sql.get_next(O_error_message,
O_item_number,
I_item_type);

EXCEPTION
WHEN OTHERS THEN
O_error_message := SQLERRM || ' from NEXT_ITEM_NO proc.';
END NEXT_ITEM_NUMBER;
/
```

This procedure must be present in the RMS database schema for the application to run. The following steps illustrates this scenario:

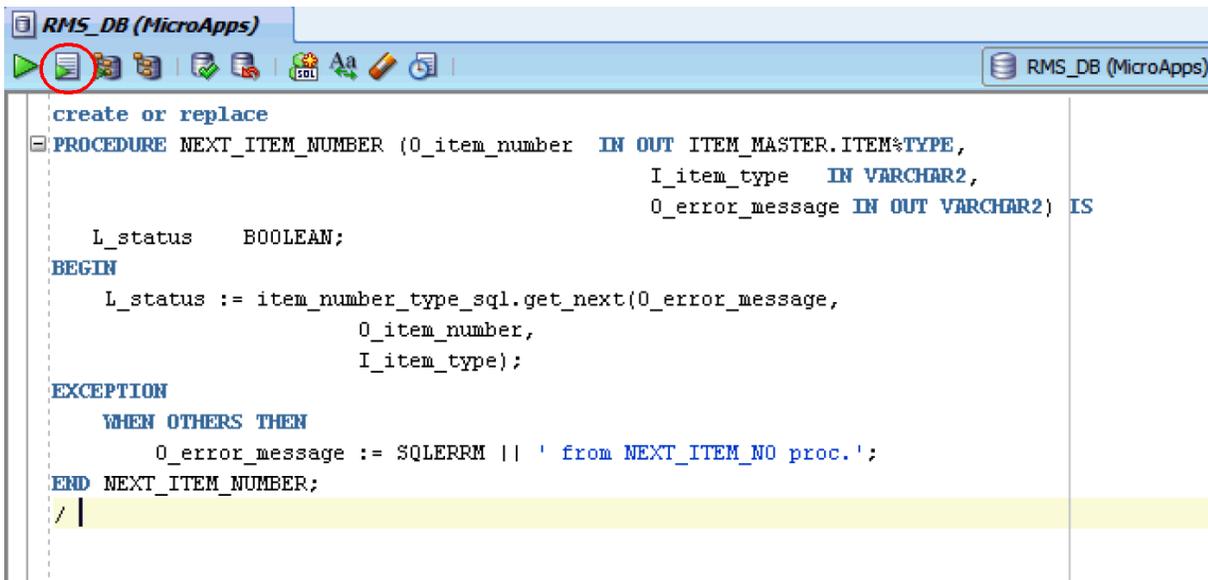
1. In JDeveloper, open the **Database Navigator** (From the **View** menu, point to **Database**, and then click **Database Navigator**).
2. Under **MicroApps**, open/expand the RMS database connection (**RMS_DB**).
3. Right-click on **RMS_DB** and click **Open SQL Worksheet**.

Figure 2–44 Open SQL Worksheet Option in the Database Navigator



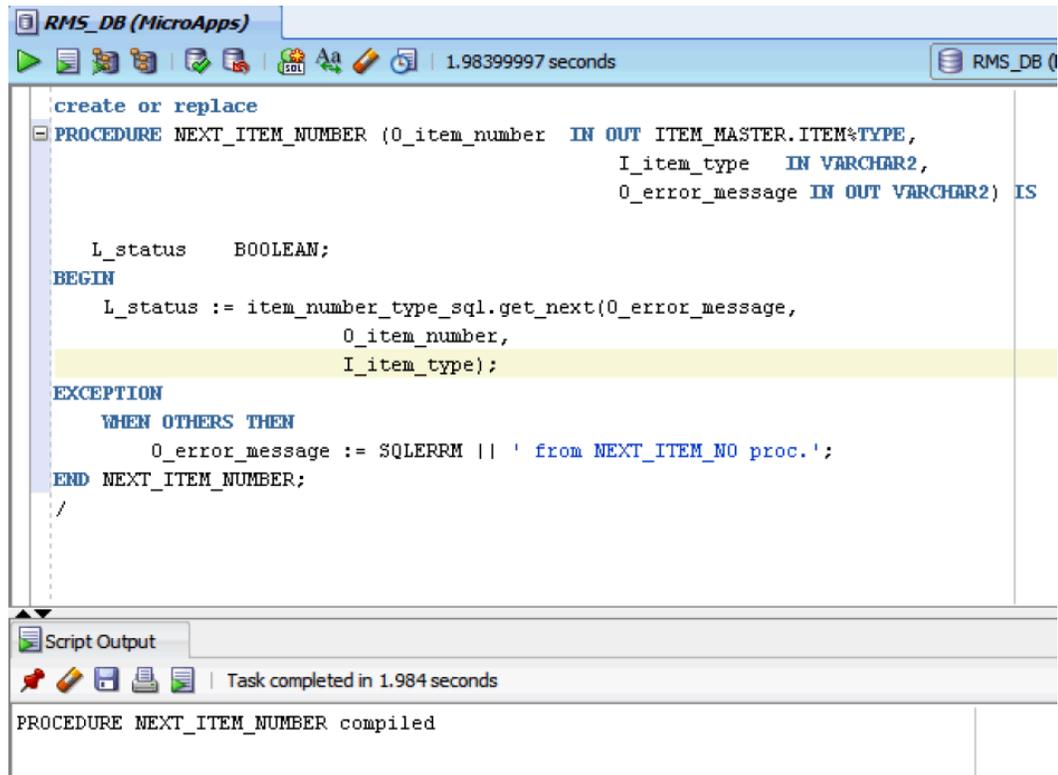
4. Copy the wrapper procedure definition provided above and paste it in the SQL Worksheet.

Figure 2–45 SQL Worksheet with the Run Script Option



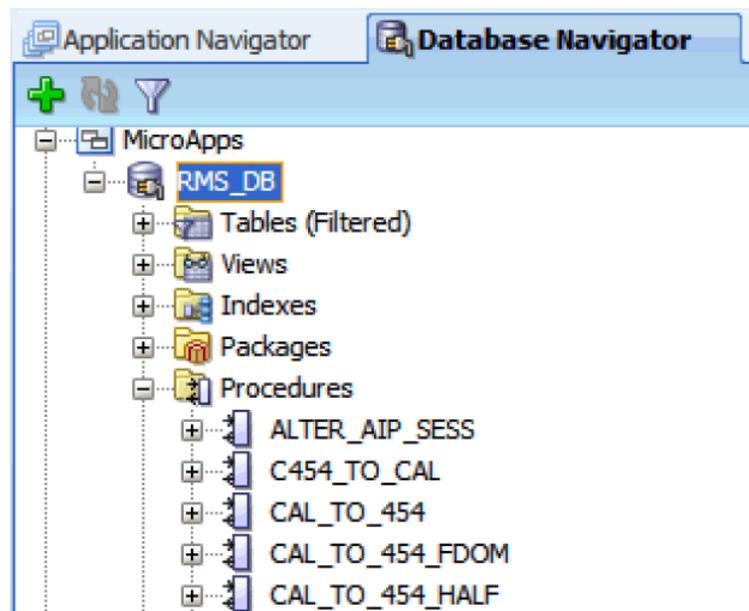
5. Click **Run Script** (or press the F5 key).
Ensure that the procedure is successfully compiled.

Figure 2–46 SQL Worksheet with the Script Output

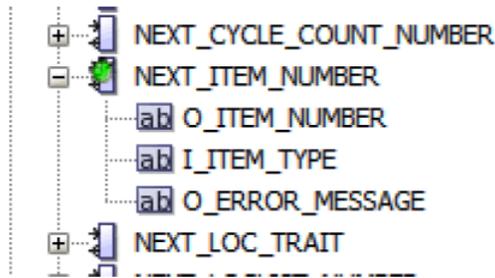


- Expand the procedure section in the Database Navigator under RMS_DB.

Figure 2–47 Database Navigator Pane with Listed Procedures



Observe that the compiled procedure is present in the schema now.

Figure 2–48 Compiled Procedures in the Database Navigator

This procedure will be called from the CreateRegularItemImpl class created above. The Java method that calls this procedure is listed below. Refer to the sample code available along with this tutorial for full code listing for this class. For more information, see [Accessing the Sample JDeveloper Project](#).

```

/**
 * This method generates sequence number for a new item in RMS.
 * It calls the PL/SQL procedure 'NEXT_ITEM_NUMBER' in RMS to generate the
 * sequence.
 * NOTE: 'NEXT_ITEM_NUMBER' is a custom PL/SQL procedure not present in RMS by
 * default.
 * It acts as a wrapper function for calling 'NEXT_ITEM_NUMBER_SQL.GET_NEXT'
 * procedure in RMS.
 * @return String Sequence number generated for Item in RMS.
 */
private String getNextItemNumber(String itemNumberType){
    CallableStatement callableStmt = null;
    String rmsUser = getDBTransaction().getConnectionMetadata().getUserName();
    String callableStatement = "begin "+rmsUser+".NEXT_ITEM_NUMBER(?,?,?); end;";
    try{
        callableStmt = getDBTransaction().createCallableStatement(callableStatement,0);
        callableStmt.registerOutParameter(1, Types.VARCHAR);
        callableStmt.registerOutParameter(3, Types.VARCHAR);
        callableStmt.setString(1, "");
        callableStmt.setString(2, itemNumberType); //passing item number type
        callableStmt.setString(3, "");
        callableStmt.executeUpdate();
        String itemSeq = callableStmt.getString(1);
        return itemSeq;
    }catch(SQLException e){
        throw new JboException(e);
    } finally{
        try{
            callableStmt.close();
        }catch(SQLException e){
            throw new RuntimeException(e);
        }
    }
}

```

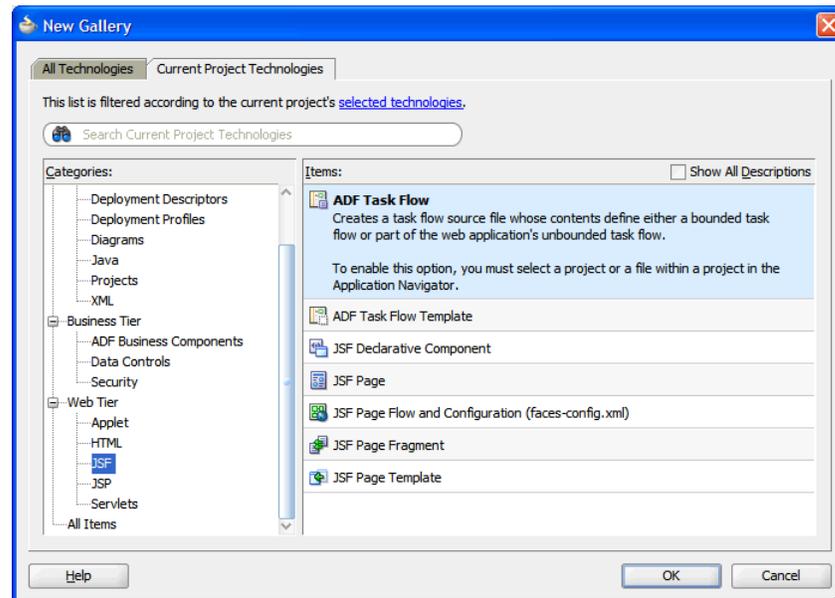
Creating an ADF Task Flow

The user interface created above is a JSF page fragment. It cannot run independently unless it is included as a region in a JSF page. This section describes how you can use this page fragment to define a task flow. Once the task flow is defined, it can be included as a region in a JSF page.

To create an ADF task flow:

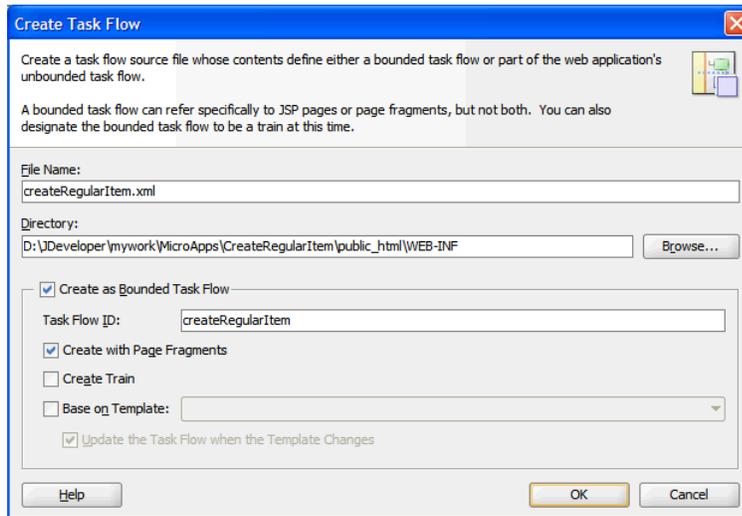
1. In the **Application Navigator**, under **Web Content**, right-click the **WEB-INF** folder, and click **New**. The **New Gallery** wizard appears.
2. In the **New Gallery** wizard, select **JSF** under **Web Tier** in the left navigation pane.

Figure 2–49 *New Gallery Wizard*



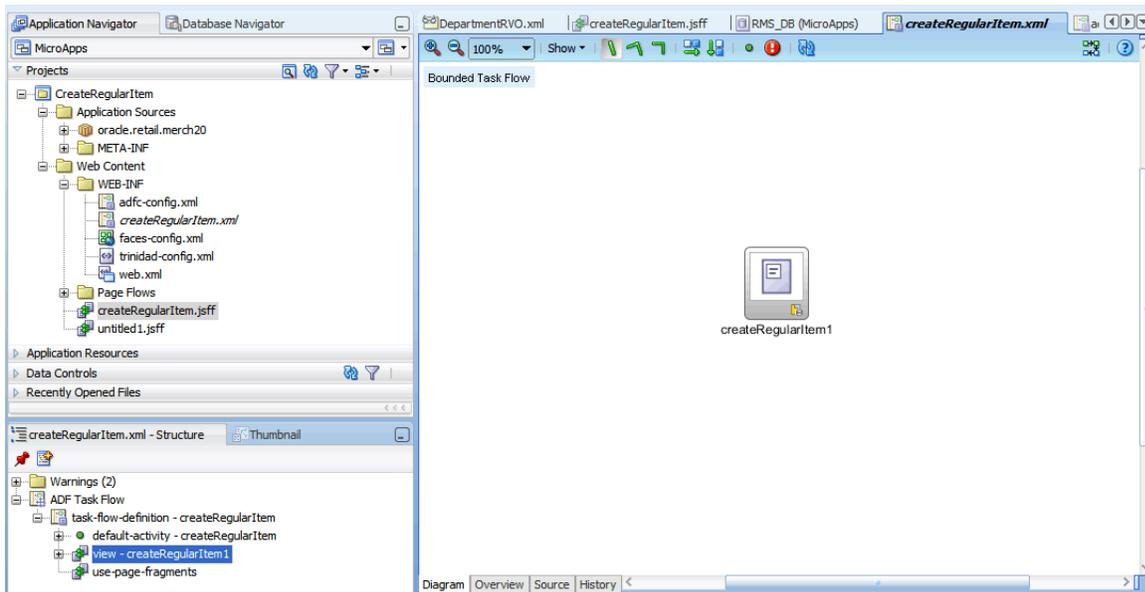
3. Under the **Items** area, click **ADF Task Flow**, and click **OK**. The **Create Task Flow** window appears.

Figure 2–50 Create Task Flow Window



4. In the **Create Task Flow** window, enter **createRegularItem.xml** in the **File Name** field.
5. Click **OK**. A task flow definition file is created in the folder.
6. Open the task flow definition file.
7. Drag and drop the page fragment you create above in to the task flow diagram.

Figure 2–51 Task Flow Definition File with a Page Fragment



8. Ensure that the default activity is set to **createRegularItem1**.
9. Save all files.
10. Now, create a new JSF page.
11. Drag and drop the task flow definition file on to the JSF file as a region.

12. Run the JSF page.

The project will deploy in the built-in WebLogic server and the page will appear in a Web browser.

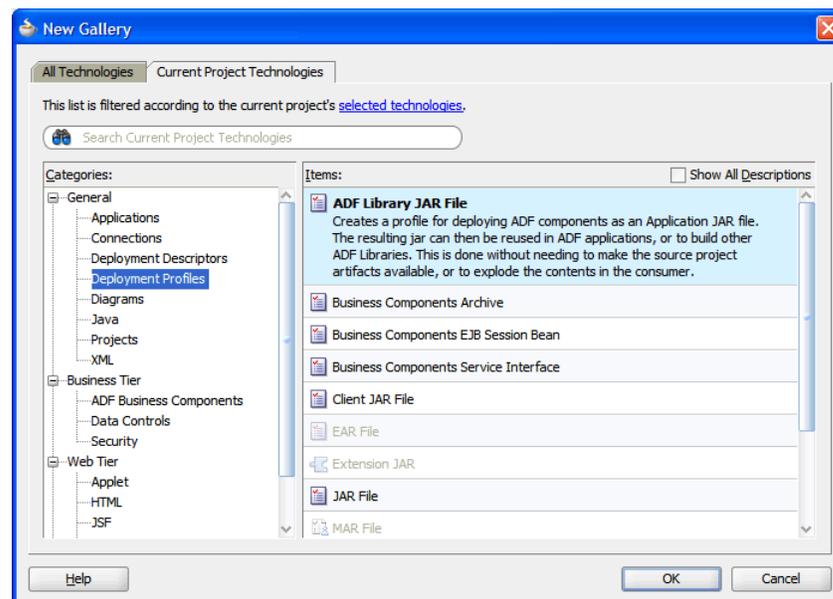
Packaging the Application

Now that you have created the application, you must package it. This application includes an ADF task flow. This task flow can be included in any ADF application as a region. This section describes how you can package the application in an ADF JAR library. This task flow will then be available in the ADF JAR library for future use.

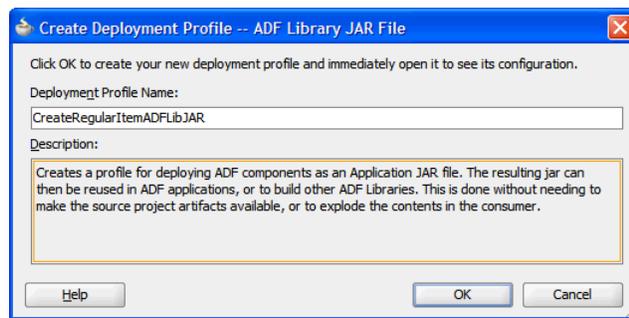
To package the application:

1. In the **Application Navigator**, right-click on the **CreateRegularItem** project.
2. In the right-click menu, point to **Deploy**, and then click **New Deployment Profile**. The **New Gallery** wizard appears.

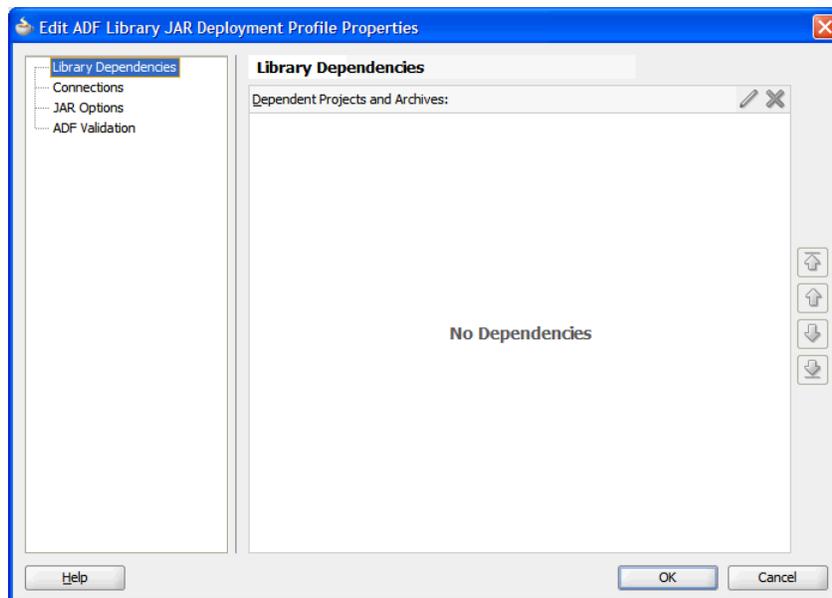
Figure 2–52 *New Gallery Wizard*



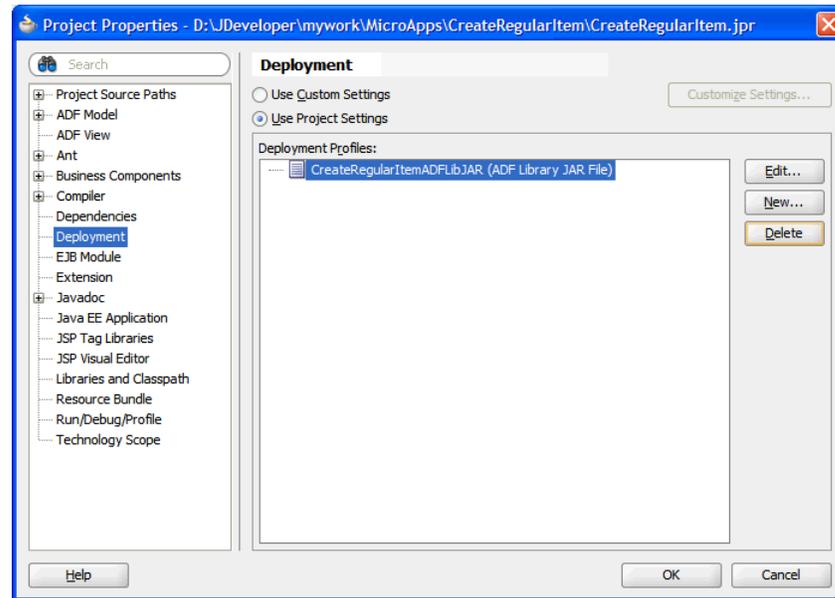
3. In the **New Gallery** wizard, select **Deployment Profiles** under **General** in the left navigation pane.
4. Under the **Items** area, click **ADF Library JAR File**, and then click **OK**. The **Create Deployment Profile** window appears.

Figure 2–53 Create Deployment Profile – ADF Library JAR File Window

5. In the **Create Deployment Profile** window, enter **CreateRegularItemADFLibJAR** in the **Deployment Profile Name** field.
6. Click **OK**. The **Edit ADF Library JAR Deployment Profile Properties** window appears.

Figure 2–54 Edit ADF Library JAR Deployment Profile Properties Window

7. Click **OK**. The **Project Properties** window appears. You can now view the deployment profile properties and edit it at any point.

Figure 2–55 Project Properties Window

8. Click **OK** and save the project.
9. Right-click on the project again, point to **Deploy**, and then click **CreateRegularItemADFLibJar**.
10. In the **Deploy CreateRegularItemADFLibJar** window, click **Finish** to deploy the application.
11. Once the deployment finishes, go to the project directory in the file system. You will find an ADF library JAR file inside the **Deploy** folder.

A

View Objects

There are several read only view objects used in this application. All these view objects are based on RMS database schema. For illustration purposes, Department and Class view objects are already explained in details in the sections above. For the rest of the view objects, all required information is provided in this appendix.

This appendix provides information on the following view objects:

- [Subclass View Object](#)
- [SystemOptions View Object](#)
- [Supplier View Object](#)
- [Supplier Site View Object](#)
- [Supplier Country View Object](#)
- [Item Number Type View Object](#)
- [Cost Zone Group View Object](#)
- [Retail Price Zone Group View Object](#)
- [Standard Unit of Measure View Object](#)

Subclass View Object

Name: SubclassRVO

Category: Read only view object

Related table in RMS: SUBCLASS

Query used:

```
select subclass as SUBCLASS_ID,
       sub_name as SUBCLASS_NAME,
       concat(concat(subclass, '-'), sub_name) as SUBCLASS_SEARCH
  from subclass
 where dept=:BindVarDeptID and class=:BindVarClassID
 Union
select null as SUBCLASS_ID,
       null as SUBCLASS_NAME,
       null as SUBCLASS_SEARCH
  from subclass
```

Bind variables used:

- BindVarDeptID: This bind variable provides the department ID selected at runtime to the Subclass view object. Process for defining the bind variable is similar to that explained in the ClassRVO creation steps.
- BindVarClassID: This bind variable provides the class ID selected at runtime to the Subclass view object. Process for defining the bind variable is similar to that explained in the ClassRVO creation steps.

Attributes to be used in UI:

- SubclassId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained in DepartmentRVO creation for DepartmentId attribute.

SystemOptions View Object

Name: SystemOptionsRVO

Category: Read only view object

Related table in RMS: SYSTEM_OPTIONS

Query used:

```
select SystemOptions.SUPPLIER_SITES_IND as SUPPLIER_ID,  
       from SYSTEM_OPTIONS SystemOptions
```

Bind variables used:

- None

Attributes to be used in UI:

- None

Note: SUPPLIER_SITES_IND attribute is intended to be exposed in the page definition file createRegularItemPageDef as an attribute values binding. Once exposed in the page definition, it is used to determine if the RMS installation uses Suppliers or SupplierSite configuration for an item.

See the usage in the createRegularItem.jsff page fragment file present in the zipped code for Supplier and SupplierSite input LOV component. See the rendered property value for these components.

Supplier View Object

Name: SupplierRVO

Category: Read only view object

Related table in RMS: SUPS

Query used:

```
select supplier as SUPPLIER_ID,  
       sup_name as SUPPLIER_NAME,  
       concat(concat(supplier, '-'), sup_name) as SUPPLIER_SEARCH  
from sups  
where supplier_parent is null
```

Bind variables used:

- None

Attributes to be used in UI:

- SupplierId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained in DepartmentRVO creation for DepartmentId attribute.

Supplier Site View Object

Name: SupplierSiteRVO**Category:** Read only view object**Related table in RMS:** SUPS**Query used:**

```
select supplier as SUPPLIER_ID,
       sup_name as SUPPLIER_NAME,
       concat(concat(supplier, '-'), sup_name) as SUPPLIER_SEARCH
from sups
where supplier_parent is not null
```

Bind variables used:

- None

Attributes to be used in UI:

- SupplierId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained in DepartmentRVO creation for DepartmentId attribute.

Supplier Country View Object

Name: SupplierCountryRVO**Category:** Read only view object**Related table in RMS:** COUNTRY**Query used:**

```
select country_id as COUNTRY_ID,
       country_desc as COUNTRY_DESC,
       concat(concat(country_id, '-'), country_desc) as COUNTRY_SEARCH
from country
```

Bind variables used:

- None.

Attributes to be used in UI:

- CountryId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained for DepartmentRVO creation for DepartmentId attribute.

Item Number Type View Object

Name: ItemNumberTypeRVO

Category: Read only view object

Related table in RMS: CODE_DETAIL

Query used:

```
select code as ITEM_NO_TYPE_CODE,
       code_seq as ITEM_NO_CODE_SEQ,
       code_desc as ITEM_NO_CODE_DESC
from code_detail where code_type='UPCT'
order by code_seq
```

Bind variables used:

- None.

Attributes to be used in UI:

- ItemNoTypeCode: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained for DepartmentRVO creation for DepartmentId attribute.

Cost Zone Group View Object

Name: CostZoneGroupRVO

Category: Read only view object

Related table in RMS: COST_ZONE_GROUP

Query used:

```
select zone_group_id as ZONE_GROUP_ID,
       description as ZONE_DESCRIPTION,
       concat(concat(zone_group_id, '-'),description) as ZONE_SEARCH
from cost_zone_group
```

Bind variables used:

- None.

Attributes to be used in UI:

- ZoneGroupId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained for DepartmentRVO creation for DepartmentId attribute.

Retail Price Zone Group View Object

Name: RetailPriceZoneGroupRVO

Category: Read only view object

Related table in RMS: RPM_ZONE_GROUP

Query used:

```
select zone_group_id as RETAIL_PRICE_ZONE_GRP_ID,
       name as RETAIL_PRICE_ZONE_NAME,
       concat(concat(zone_group_id, '-'),name) as RETAIL_PRICE_ZONE_SEARCH
from rpm_zone_group
```

Bind variables used:

- None.

Attributes to be used in UI:

- RetailPriceZoneGrpId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained for DepartmentRVO creation for DepartmentId attribute.

Standard Unit of Measure View Object

Name: StandardUomRVO

Category: Read only view object

Related table in RMS: UOM_CLASS

Query used:

```
select uom as UOM_ID,  
       uom_desc as UOM_DESC,  
       concat(concat(uom, '-'), uom_desc) as UOM_SEARCH  
from uom_class
```

Bind variables used:

- None.

Attributes to be used in UI:

- UomId: It is intended to be used as ADF LOV input component in UI. This requires the attribute to be configured for LOV as explained for DepartmentRVO creation for DepartmentId attribute.



Oracle Corporation
World Headquarters
500 Oracle Parkway
Redwood Shores, CA 94065
U.S.A.

Worldwide Inquiries:
Phone: +1.650.506.7000
Fax: +1.650.506.7200
oracle.com

Copyright © 2008, Oracle. All rights reserved.
This document is provided for information purposes only and the contents hereof are subject to change without notice.
This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission. Oracle, JD Edwards, PeopleSoft, and Siebel are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.