Building J2EE Applications with Oracle JHeadstart for ADF

An End-to-End Tutorial on How to Be Effective Immediately with J2EE Application Development

Author: Steve Muench, Oracle ADF Development Team
Contributions from Steven Davelaar & Sandra Muller, JHeadstart Team

Date: April 29, 2005

Abstract

There is good chance you've heard that developing database-centric business applications for the J2EE platform is a complex endeavor. Luckily, there is hope. Over the past year, analysts like Gartner Group coined the term "J2EZ" for a class of tools and frameworks that make J2EE application development easy. These so-called "J2EZ" tools target developers for whom time to market, low implementation costs, and productivity are driving factors, by providing a visual, declarative, and highly productive environment. This document provides step-by-step instructions to evaluate what you can achieve today using production versions of Oracle's offerings in this "J2EZ" space by combining Oracle's JDeveloper 10g IDE, the Oracle Application Development Framework (ADF), and the Oracle JHeadstart 10g Application Generator for Oracle ADF.

By following this tutorial, you'll experience first-hand how Oracle JHeadstart turbo-charges your developer productivity for Oracle ADF-based web applications. You will build an attractive, consistent, interactive, and skinnable web application with browse, search, insert, update, and delete functionality against six related database tables from the Oracle HR sample schema. Your application will feature single- and multi-row editing, page-by-page scrolling, master/detail handling, dropdown lists, a pop-up LOV, a shuttle picker, and a tree control. Since no Java coding is required to implement the tutorial, even developers with minimal Java skills can follow along. This is possible because Oracle ADF-powered J2EE applications only require custom code to add application-specific business logic or to augment default framework behavior.

NOTE: While we've tested that this document should print fine in Internet Explorer; other browsers like Firefox might split figures across page breaks when printing. If you prefer, a PDF version of this paper [1] is also available.

NOTE: While the installation instructions and screenshots in this document are for Microsoft Windows XP, JDeveloper 10g and Oracle JHeadstart 10g work on any platform where JDeveloper 10g is supported, including Windows NT/2000/XP, Linux, Mac OSX, Solaris, and HP-UX. The installation steps are virtually identical on the other platforms.

NOTE: This tutorial complements the JHeadstart Developer's Guide [2], which thoroughly explains all of the features of JHeadstart 10g in detail.

Contents

Overview of Oracle ADF and Oracle JHeadstart 10g

On their own, the Oracle Application Development Framework (ADF) together with the Oracle JDeveloper 10g IDE give developers a productive, visual environment for building richly functional J2EE applications without having to implement J2EE design patterns and low-level plumbing code by hand. As its name implies, Oracle JHeadstart 10g offers a significant additional productivity advantage in creating sophisticated web-based, J2EE business applications. Standing squarely on the shoulders of the base Oracle ADF framework, Oracle JHeadstart adds an additional capability for generating fully-working ADF UIX or JSP web tier for the data model exposed by your ADF application module. The ADF web applications you generate with JHeadstart can easily support multi-row editing, LOV's with validation, tree and shuttle displays, and consistent search pages, among many other features.

JHeadstart automates creating and iteratively evolving the web tier of your ADF application based on a declarative application structure definition. Once you've allowed JHeadstart to generate the bulk of your application's web user interface, you can spend your time using JDeveloper's productive environment to tailor the results or to concentrate your effort on real showcase pages that need special attention.

Like all Oracle ADF-based applications, those leveraging Oracle JHeadstart are J2EE compliant and therefore easy to deploy to any J2EE application server. They also inherit Oracle ADF's implementation all of the best practice J2EE design patterns [3] required to implement robust, functional business applications.

As we'll see in this step-by-step demo, the JHeadstart Application Generator does not generate any Java code. Instead, it generates web pages, ADF metadata describing the data needed on those pages, Struts metadata describing the page flow, and translatable message bundle files. We'll also see that all of the basic functionality provided by the underlying Oracle ADF framework components in the demo do not require any generated Java code either. We hope you'll walk away impressed by what you can do without writing a single line of Java code using this powerful combination of J2EE tools and frameworks from Oracle. Any lines of code that you would eventually write in a real application would be squarely focused on enhancing all of the

built-in functionality provided with your own custom business application logic.

NOTE: Oracle JHeadstart 10g for ADF is a separate extension for Oracle JDeveloper 10g for which a fully-functional trial version is available for your evaluation purposes. Complete information on pricing, support, and additional services available for JHeadstart 10g are available in the JHeadstart Frequently Asked Questions [4] document on OTN.

TIP: After you've followed the demo steps yourself, the same steps work well as a scripted demo you can show to others to spread the good word about the many powerful features provided by the combination of Oracle JDeveloper 10g, Oracle ADF, and Oracle JHeadstart working together.

Tutorial Setup

This section outlines the steps you'll need to follow to get your machine ready to go through this tutorial. We recommend not skipping any steps in this section without reading them!

Download the Tutorial Files

If you are reading this tutorial online, download the jhs-step-by-step.zip [5] file that contains the database setup scripts. Extracting this zip file into the root directory of your `C:\` drive will create a `jhs-step-by-step` directory. You can browse the `index.html` page to read this same tutorial *offline*. The database setup scripts (described more in detail below) are in the `hr_schema` subdirectory.

Start with JDeveloper 10g, Release 10.1.2

The demo steps require Oracle JDeveloper 10g, release 10.1.2. If you have a version of Oracle JDeveloper installed, you can verify what version it is by selecting the ***Help*** | ***About*** option from the main menu.

If you need to download JDeveloper 10g, Release 10.1.2, look for the correct 10.1.2 version on this JDeveloper Downloads [6] page. (Hint: It might not be the first one in the list!)

To install JDeveloper, create a convenient directory (e.g. `C:\jdev1012`) and extract the `jdev1012.zip` file into this directory using the standard Java `jar` utility, WinZip, or other zip-compatible tool you prefer.

To run JDeveloper, double-click on `jdevw.exe` in the `C:\jdev1012\jdev\bin` folder as shown in Figure 1.

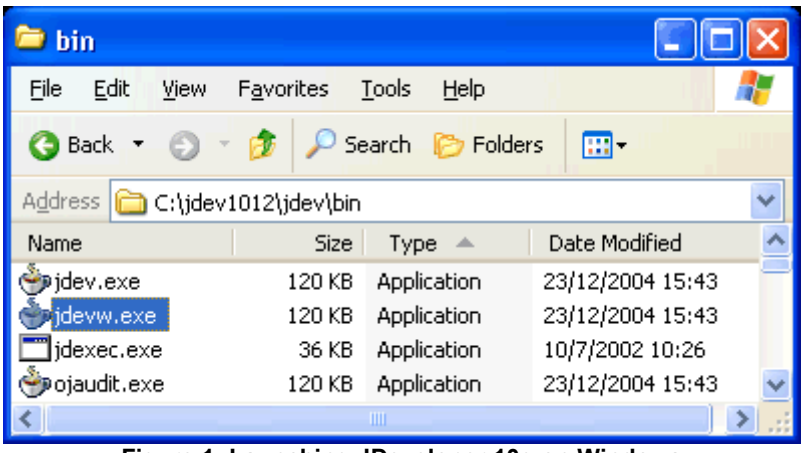


Figure 1: Launching JDeveloper 10g on Windows

NOTE: On Unix-based platforms, run the `./jdev/bin/jdev` shell script to launch JDeveloper 10g.

Download and Install Oracle JHeadstart 10g for ADF, Release 10.1.2

Oracle JHeadstart is an Oracle JDeveloper extension (a.k.a "plug-in") that you install into the JDeveloper 10g IDE. To install the JHeadstart 10g extension into your JDeveloper 10g 10.1.2 environment, perform the steps below.

NOTE: These instructions assume you've installed JDeveloper 10g into the `C:\jdev1012` directory. Substitute your own JDeveloper home directory for `C:\jdev1012` if you already had JDeveloper 10.1.2 installed.

- ***Download the JHeadstart 10g extension for JDeveloper 10g***

  You find it under the ***Downloads*** heading on the JHeadstart 10g Product Center [7] on OTN. The file will be named `jhs1012.zip`.

- ***Exit from JDeveloper 10.1.2 Before Installing the JHeadstart Extension***

  Make sure JDeveloper 10.1.2 is not currently running.

- ***Extract the JHeadstart Distribution***

  Create a directory to contain the JHeadstart installation (e.g. `C:\JHeadstart`) and unzip the downloaded `jhs1012.zip` file into this directory, preserving directory structure.

- ***Copy JHeadstart Extension Files Under JDeveloper Extensions Directory***

  First, create a subdirectory named `jheadstart` under the directory for JDeveloper extensions (`C:\jdev1012\jdev\lib\ext\jheadstart`).

  Then, copy the JHeadstart extension JAR files (`C:\JHeadstart\config\jdevaddins\*.jar`) to the `C:\jdev1012\jdev\lib\ext\jheadstart` directory.

- ***Optionally, Enable Oracle Designer Integration***

  This tutorial does not delve into JHeadstart's support for generation from an existing Oracle Designer repository. However, if later you want to be able to experiment with this feature to try generating complete ADF web applications from repository module components, you can optionally perform this setup step now.

  Copy the `C:\JHeadstart\config\jdevaddins\migration.xml` file to the `C:\jdev1012\jdev\system10.1.2.0.0.1811` directory.

  NOTE: On Linux this directory is in the home directory of the user that is running JDeveloper: $HOME/jdevhome/system10.1.2.0.0.1811.

- ***Set the JHeadstart Installation Home Preference in JDeveloper***

  After starting JDeveloper 10.1.2, select ***Tools | Preferences...*** from the main menu. Select the ***JHeadstart Settings*** category, and enter `C:\JHeadstart` for the value of the ***JHeadstart Home*** property.

When the value of the JHeadstart Home corresponds to a correct installation directory for JHeadstart, you'll see the JHeadstart build number reflected in the preference panel as shown in Figure 2.

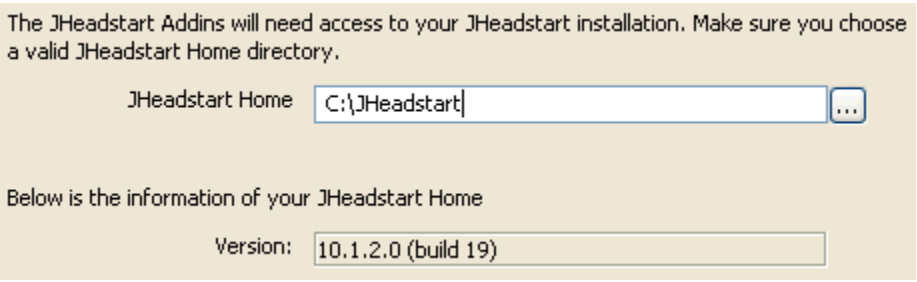


Figure 2: JHeadstart Build Number Displays When Correctly Installed

Click ***(OK)*** to close the Preferences dialog.

Setup the Oracle HR Schema and Sample Data

The demo steps are based on the standard Oracle sample `HR` schema of human resources information. For your convenience, this tutorial comes with the SQL scripts to create the HR schema tables if you didn't install the sample schemas when you installed your database.

NOTE: If the steps in this section do not work for you, then as a fallback please refer to the Installing the Sample Schemas and Establishing a Database Connection [8] tutorial and follow the instructions for the database version you are using.

Create the HR Schema If Necessary

If you don't already have an `HR` user account created in your database, you can follow these steps to create it.

```
C:\jhs-step-by-step> sqlplus /nolog
SQL> connect sys as sysdba
SQL> create user hr identified by hr;
created.
SQL> alter user hr default tablespace users;
altered.
SQL> grant connect, resource to hr;
SQL> connect hr/hr
connected.
SQL> quit
```

Create the HR Schema Sample Tables

In the `hr_schema` subdirectory of this tutorial, you'll find the `hr.sql` script. This script drops, recreates, and repopulates all the tables in the `HR` sample schema. Change directory to the hr_schema subdirectory, and run the script as the `HR` user, with the command:

```
C:\jhs-step-by-step> cd hr_schema
C:\jhs-step-by-step\hr_schema> sqlplus hr/hr @hr.sql
```

NOTE: If you are using an Oracle 9i database, you can ignore the error you'll receive at the end for the `PURGE RECYCLEBIN` command. That command is specific to Oracle 10g.

Define a JDeveloper Connection for the HR Schema

Select ***View | Connection Navigator*** to show the ***Connection Navigator*** and follow these steps:

- ***Create a New Database Connection***

  Click on the ***Database*** folder, and select ***New Database Connection...*** from the right-mouse menu as shown in Figure 3.

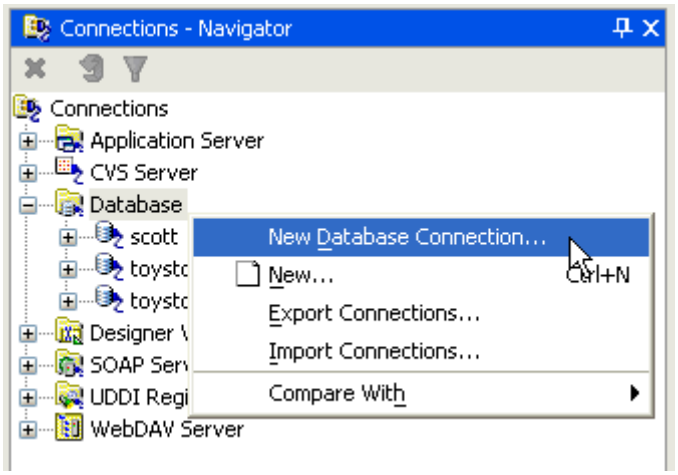


Figure 3: Defining a New Database Connection

- ***Enter a Connection Name***

  On step 1 of the wizard, as shown in Figure 4, provide a connection name of `hr` and click ***(Next>)***.

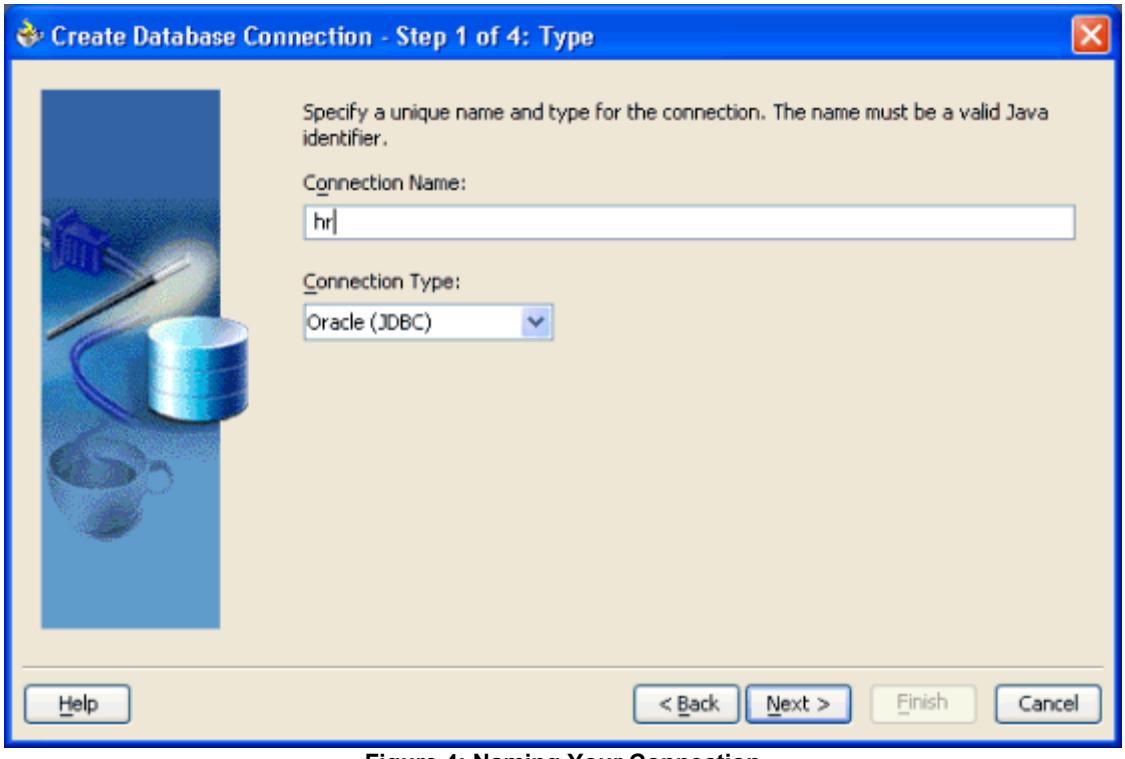


Figure 4: Naming Your Connection

- ***Provide the Username and Password Details***

  On step 2, provide the username and password information for the hr connection. Make sure to check the ***Deploy Password*** option as shown in Figure 5, then press ***(Next>)***.

  NOTE: It's OK to deploy the password in the connection definition since it gets saved in an encryped format.

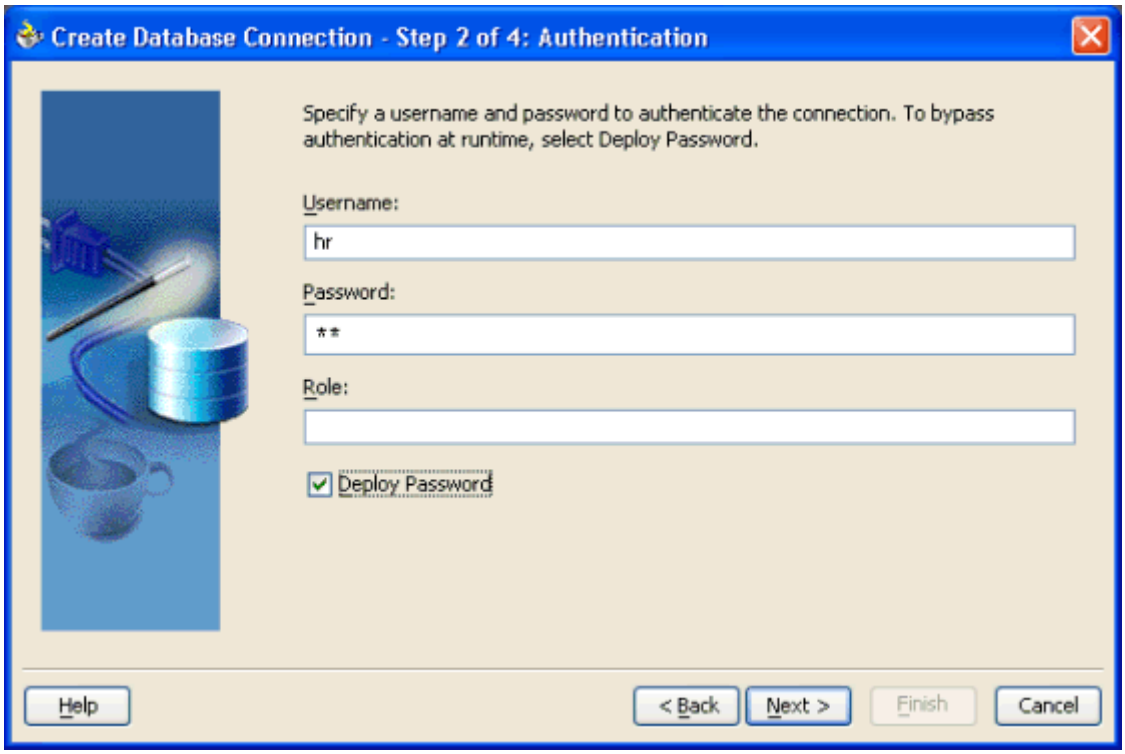


Figure 5: Providing Username/Password for the Connection

- ***Provide Host, Port, and SID Information***

  In step 3, provide the host, port, and SID information for the database where you've installed the HR schema tables. The defaults shown in Figure 6 are right for a default Oracle database installation on the current machine. You may have to change the settings if you're database is on another machine or has a different SID. Press ***(Next>)*** when done with this step.

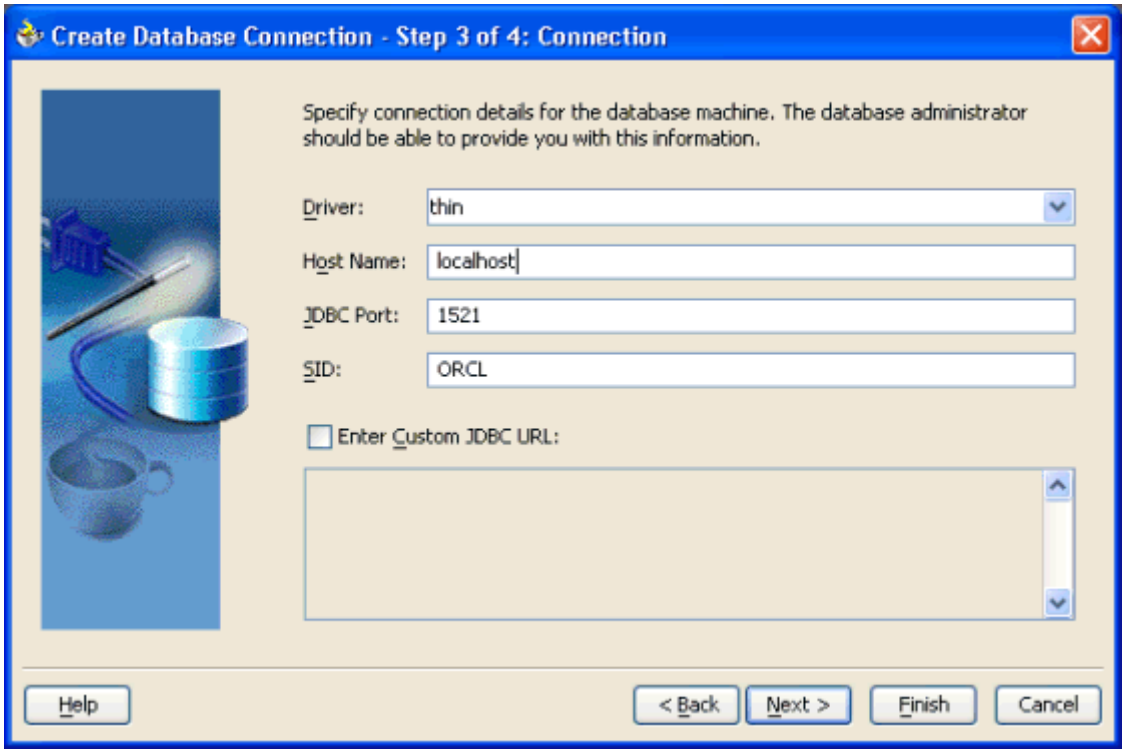


Figure 6: Providing Database Connection Details

- ***Test the Connection***

  As shown in Figure 7, click the ***(Test Connection)*** button to test the connection. If you don't see the "Success!" message, press ***(<Back)*** and double-check the information you've provided on the previous wizard pages. When you see "Success!", click ***(Finish)*** to complete the process of defining a connection.

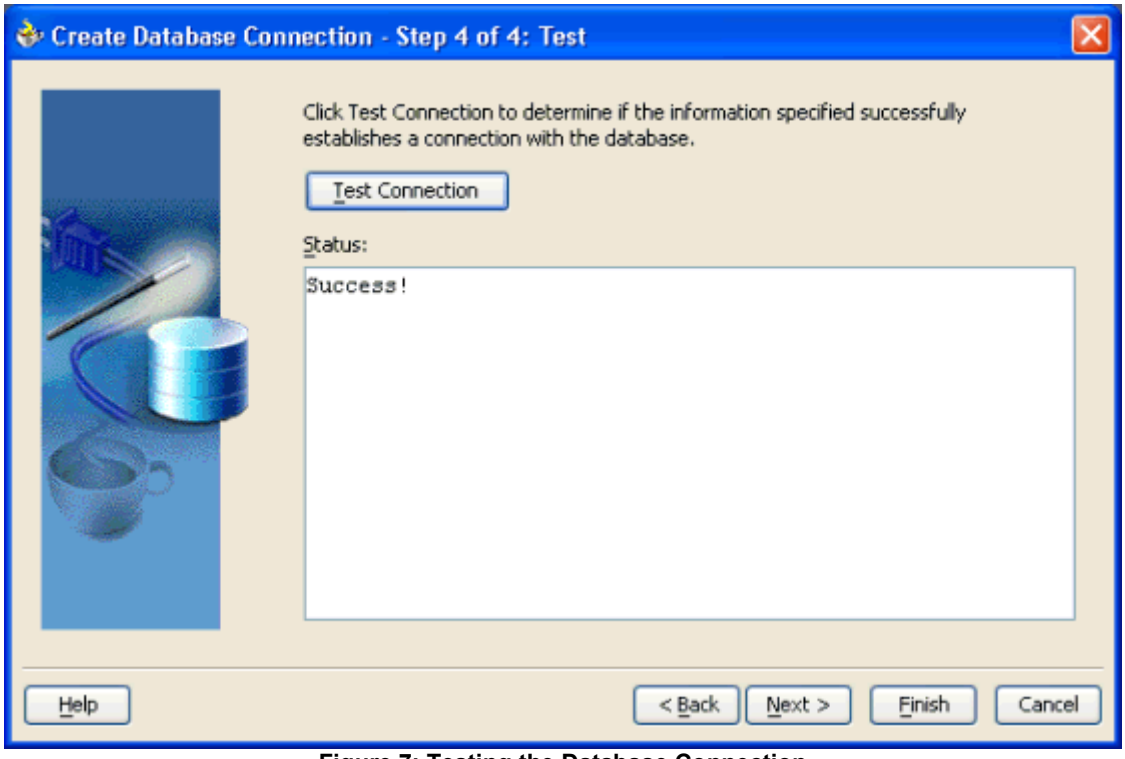


Figure 7: Testing the Database Connection

Set ADF Business Components Java Generation Preferences

By default, the ADF Business Components design time is configured to generate skeleton Java classes where developers could eventually write their code. Producing these skeleton classes by default is not required, and there is an IDE preference to control it. If you don't generate them by default, you can easily add them on a case-by-case basis should custom Java code ever become required for a given component type.

In order to better understand what ADF application functionality requires Java code and what does not, it's useful to set the ADF Business Components Java generation preferences at the IDE level so that the wizards and editors create no custom Java classes by default. By doing this, at the end of the demo we can see what Java classes actually are part of our working, finished application to get a better understanding of what combined, built-in functionality Oracle ADF and Oracle JHeadstart 10g provide.

To change this preference, select ***Tools | Preferences...*** from the JDeveloper main menu, and click on the ***Business Components*** category. *Uncheck* any of the checked boxes in the panel at the right as shown in Figure 8, then click ***(OK)***.

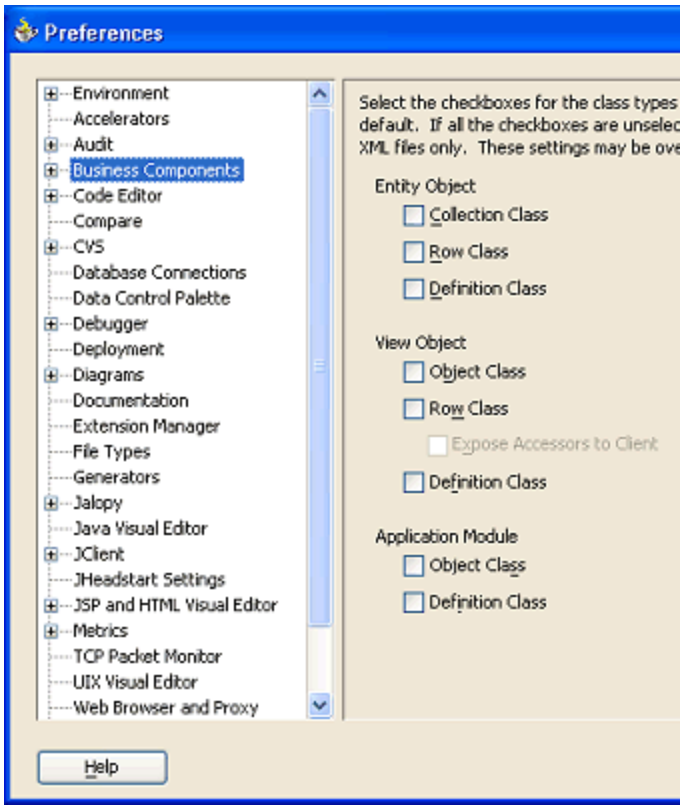


Figure 8: Setting ADF Business Components to Generate No Java By Default

Step 1: Create Default Web Application

In this section we will:

- Create a new workspace,
- Create the ADF Business Components to handle our backend database access, and
- Generate a default set of web pages with JHeadstart

Then, we'll run the application inside JDeveloper 10g to see what default behavior we get before starting to iteratively modify the application to further tailor it to work like our end users want.

Step 1a: Create and Configure a New Workspace

1. ***Create a new Application Workspace***

Select ***File > New...*** and pick ***Application Workspace***. Use the template named ***Web Application [default]*** as shown in Figure 9. To make it easier to follow along in this tutorial, please enter the ***Application Name*** (`MyDemo`), ***Directory Name*** (`C:\MyDemo`), and ***Application Package Prefix*** (`mydemo`) exactly as shown. This way you'll be sure to create something that matches what the rest of the tutorial will be showing you.

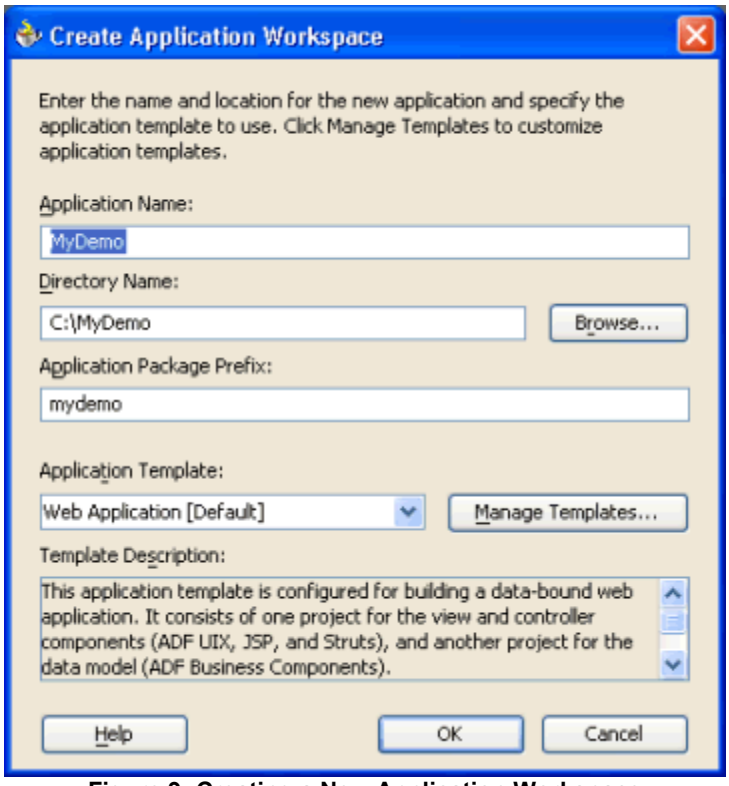


Figure 9: Creating a New Application Workspace

When you click ***(OK)*** to create the workspace, JDeveloper will create your new `MyDemo` workspace that will contain two (initially empty) projects named `Model` and `ViewController`.

2. ***Give Your Web Application a Meaningful Web Context Root Name***

   With the `ViewController` project still selected in the navigator, select ***Project Properties...*** from the right-mouse menu. Visit the ***J2EE*** category and notice that the default web context root name is `MyDemo-ViewController-context-root`. The J2EE web context root name is the first part of your application's web URL that users will see, so having a short, meaningful name is preferrable to this long default name. So let's set the ***J2EE Web Context Root*** to just "`MyDemo`" instead. Technically, this step is optional, but it gives your application a nicer-looking runtime URL and it's nice to know where you would change it for your own applications.

3. ***Simplify Running the Application By Changing a Project Setting***

   While still in the Project Properties dialog, to make it easier to quickly run the application, select the ***Runner*** category, and *uncheck* the checkbox ***Attempt to Run the Active File before Default***.

Step 1b: Create Default ADF Business Components

The ADF Business Components handle all of the database access for you in a way that is cleanly separated from the user interface. The ***application module*** provides the transactional component clients use to browse and modify view object data.

The ***view object*** performs SQL queries and coordinates with entity objects to handle updates. The ***entity object*** encapsulates business domain data and validation for rows in a table. In this step we'll use

NOTE: For additional background to ADF Business Components, including information on how their functionality maps to features of Oracle Forms, see my ongoing columns in the Oracle Magazine DEVELOPER: Frameworks [9] series.

1. ***Create Default Business Components for Tables in HR Schema***

   - ***Run the Business Components from Tables Wizard***

     Select your `Model` project and choose ***New...*** from the right-mouse menu. In the New Gallery, expand the ***Business Tier*** node, select the ***Business Components*** category, and choose ***Business Components from Tables***.

   - ***Set the Database Connection to Use***

     When the ***Business Components Project Initialization*** dialog appears, select the name of the `HR` connection you defined above as shown in Figure 10.

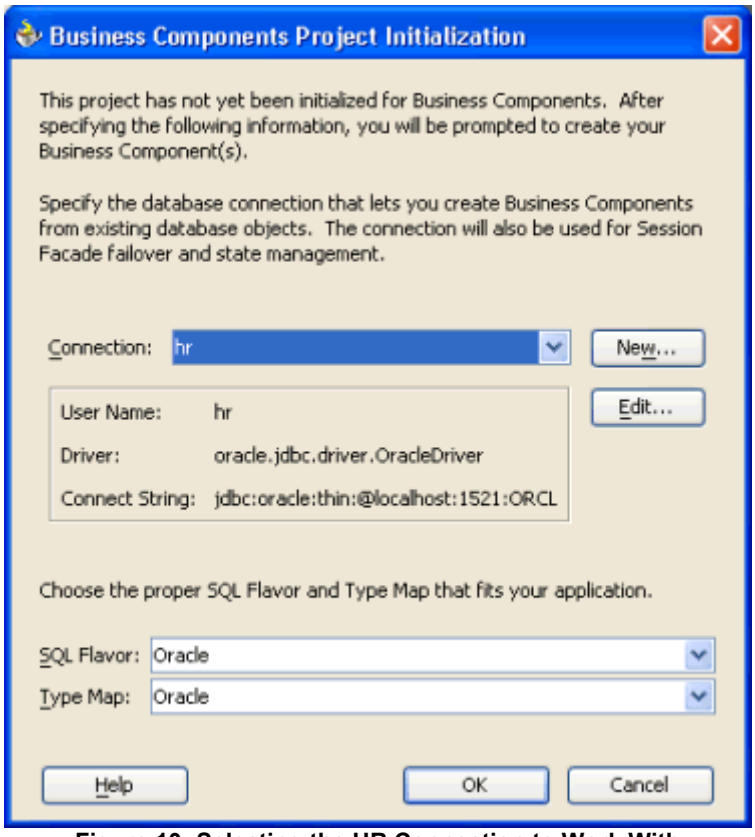


Figure 10: Selecting the HR Connection to Work With

   - ***Create Entity Objects for Selected HR Tables***

     On the ***Step 1 of 4: Entity Objects from Tables*** page of the wizard, as shown in Figure 11 shuttle the six tables `COUNTRIES`, `DEPARTMENTS`, `EMPLOYEES`, `JOBS`, `LOCATIONS`, and `REGIONS` tables into the ***Selected*** list.

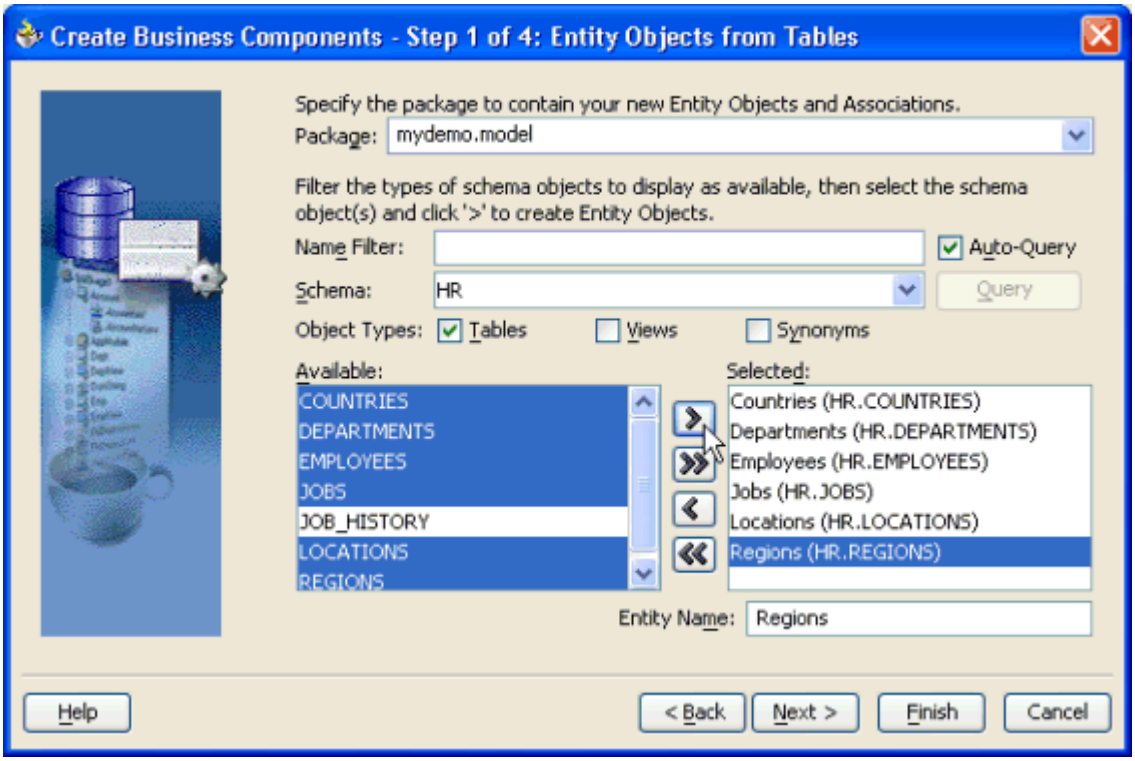


Figure 11: Selecting Tables for Which to Create Entity Objects

NOTE:	In this tutorial, to keep things simple we'll create all of our business components in the single `mydemo.model` package in the `Model` project. This is the default package name that will appear in the ***Package*** field at the top of each panel of this wizard. In practice, for applications of any larger size it is best practice to use different Java packages to organize your business components into smaller functional groups.

- ***Create Updateable View Objects for All Entity Objects***

  On the ***Step 2 of 4: Updateable View Objects from Entity Objects*** page of the wizard, as shown in Figure 12 shuttle all of the available entity object names into the ***Selected*** list in order to create default view objects for them.

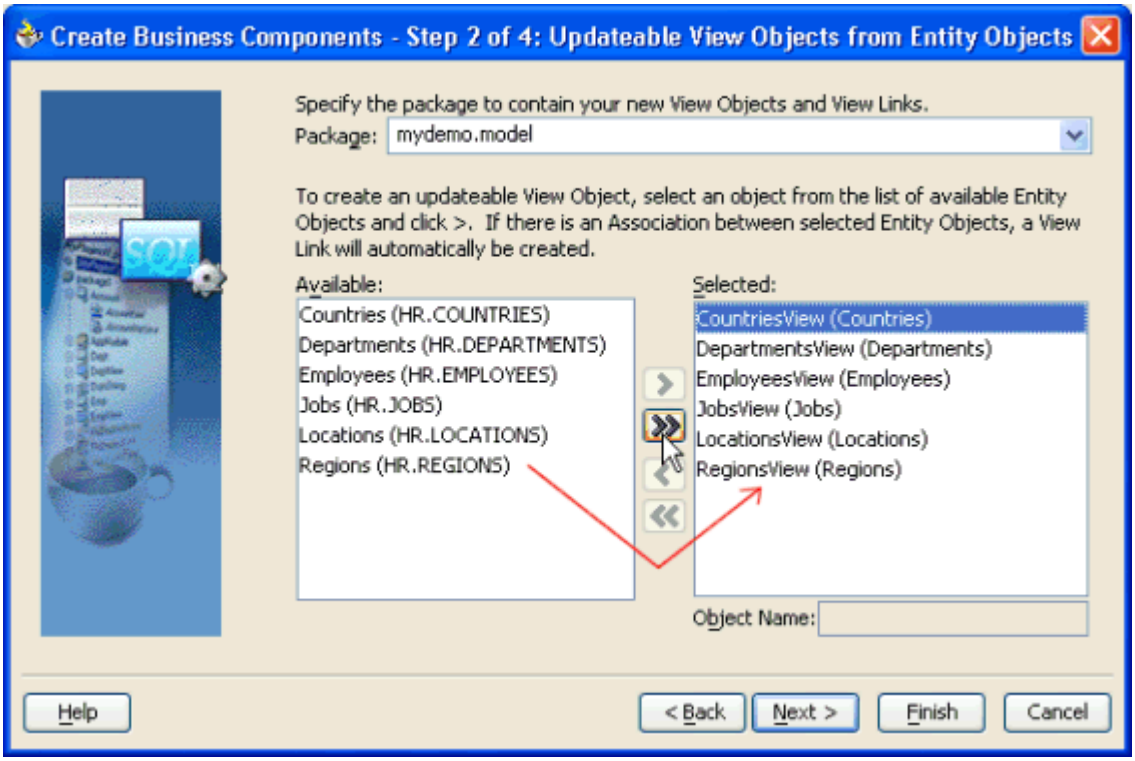


Figure 12: Creating Updateable View Objects for All Entity Objects

- ***Skip Past the Read-Only View Objects Panel***

  As shown in Figure 13, just skip past the ***Step 3 of 4: Read-Only View Objects from Tables*** page of the wizard since we don't need any read-only view objects for this demo.

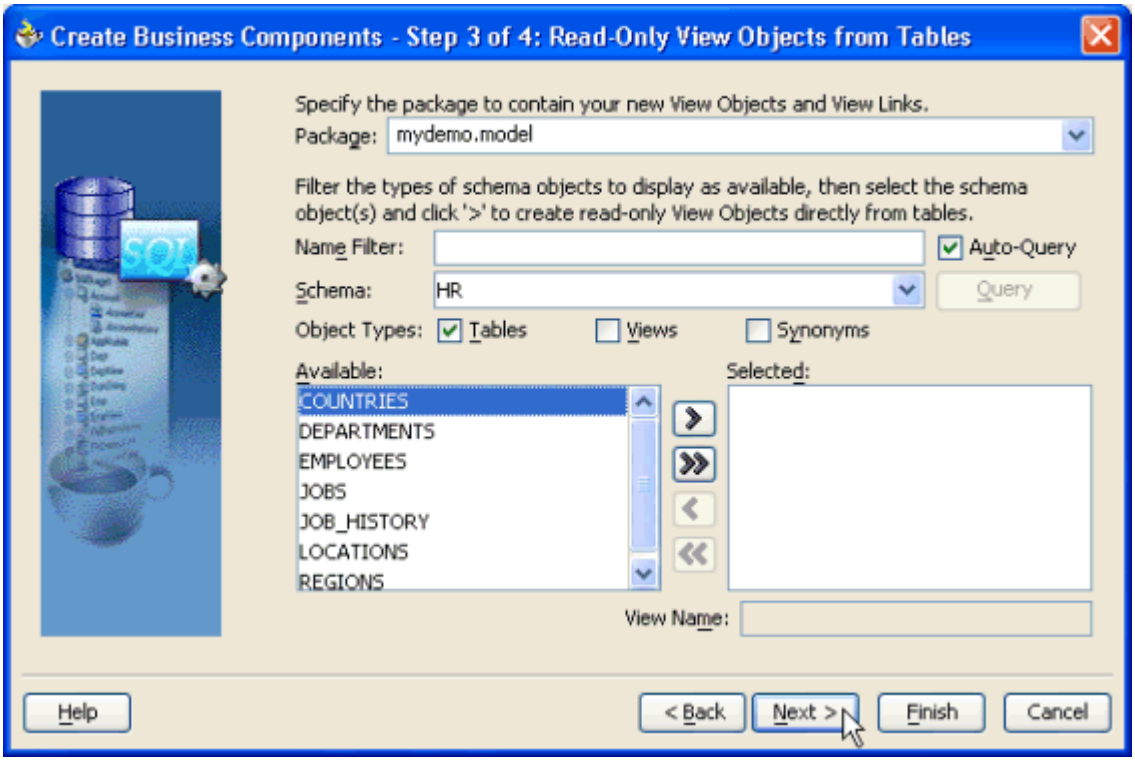


Figure 13: Skipping the Read-Only View Objects Panel

- ***Give Your Application Module Component a Meaningful Name***

  On the ***Step 4 of 4: Application Module*** page of the wizard, as shown in Figure 14 choose a meaningful name for your application module like `HRModule`.

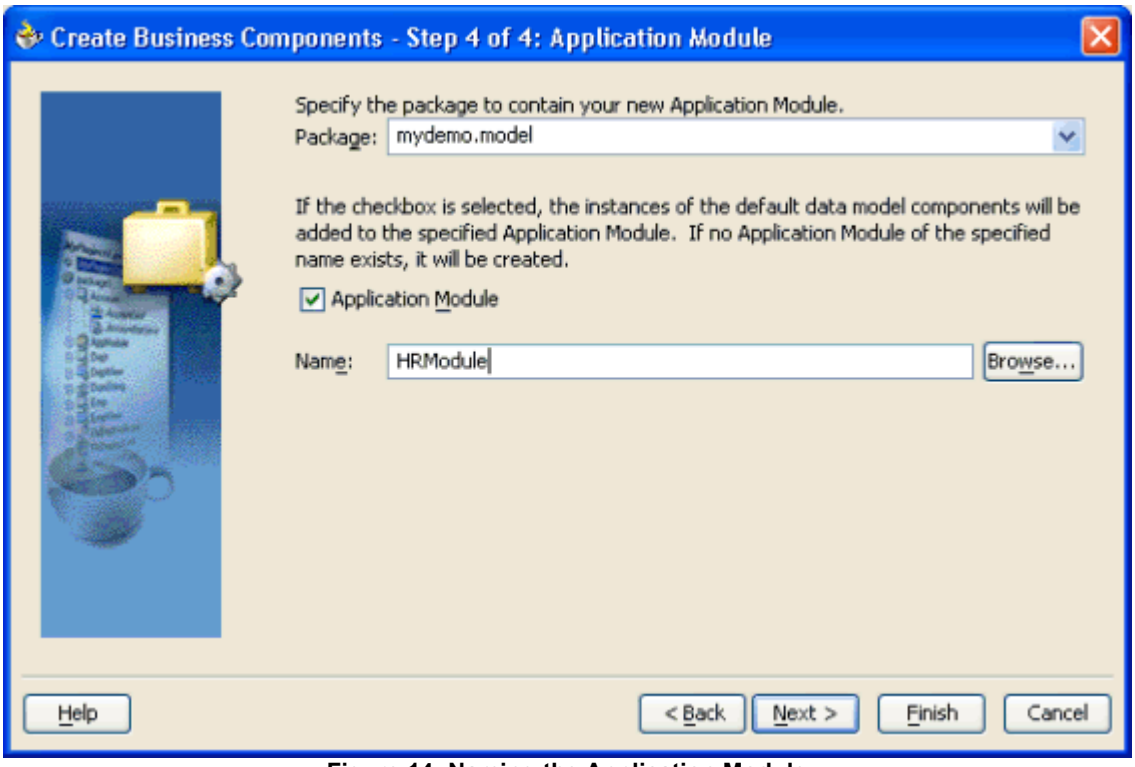


Figure 14: Naming the Application Module

- ***Review the Business Components to Generate, and Finish***

  Click ***(Next)*** to review the business components that you've selected to generate and you'll see the ***Finish*** page as shown in Figure 15. Then, click ***(Finish)*** to create all of your business components.

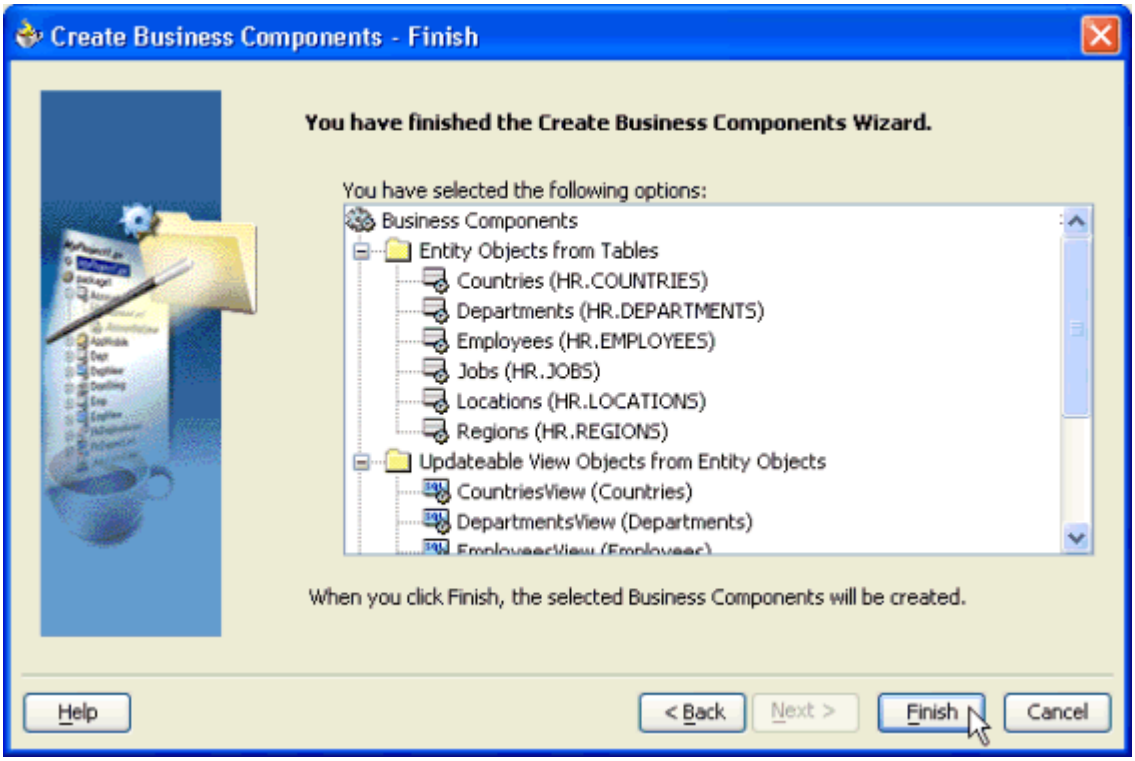


Figure 15: Reviewing the Business Components To Create

> **NOTE:** For a more detailed explanation of this step, see the "Setting Up the Business Components Package" section in ***Chapter 2. Getting Started*** from the JHeadstart Developer's Guide [2].

2. ***Add Five-level Master/Detail to the Application Module's Data Model***

   In the Application Navigator, inside your `Model` project find the `HRModule` component. It's in the `mydemo.model` package. Edit the `HRModule` application module by double-clicking it. This will open the ***Application Module Editor***. Visit the ***Data Model*** panel, and as shown in Figure 16 you'll see both the list of ***Available View Objects*** from the current project, and the named, master/detail-coordinated *usages* of those view objects in this application module's ***Data Model***. The view object usages in the data model are also known as view object *instances* since at runtime they represent an instance of the reusable view object component.

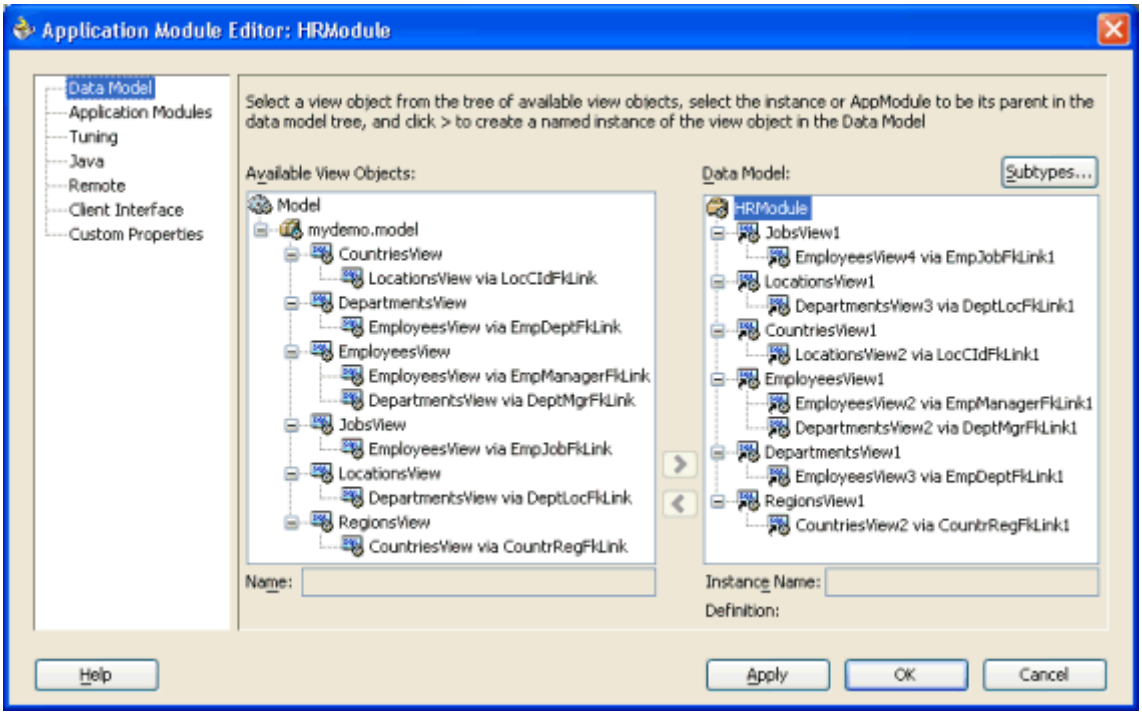


Figure 16: Initial HRModule Data Model

We're going to modify the default data model to add nested view object usages in order to get a 5-level deep nesting of region → country → location → department → employee.

We start by adding a `LocationsView` view object instance as a child/detail of `CountriesView2`, by doing the following:

- In the ***Data Model*** tree control, select the existing view instance `CountriesView2` that will become the new master (for the next-level detail we're about to add).
- Expand the `CountriesView` view object in the ***Available View Objects*** list
- Select the `LocationsView` view object that is indented under ***CountriesView*** in the ***Available View Objects*** list. This indenting means that it is an *available* detail view for any instance of a `CountriesView` master in the data model.
- Click the ***(>)*** button to shuttle a new `LocationsView3` instance into the data model as detail of the existing, selected `CountriesView2`.

We can repeat the process to add an instance of `DepartmentsView` as a detail of the `LocationsView3`, and then again to add an instance of `EmployeesView` as a detail of the `DepartmentsView4`.

When finished adding the three extra levels, your application module's data model will look like what you see in Figure 17.

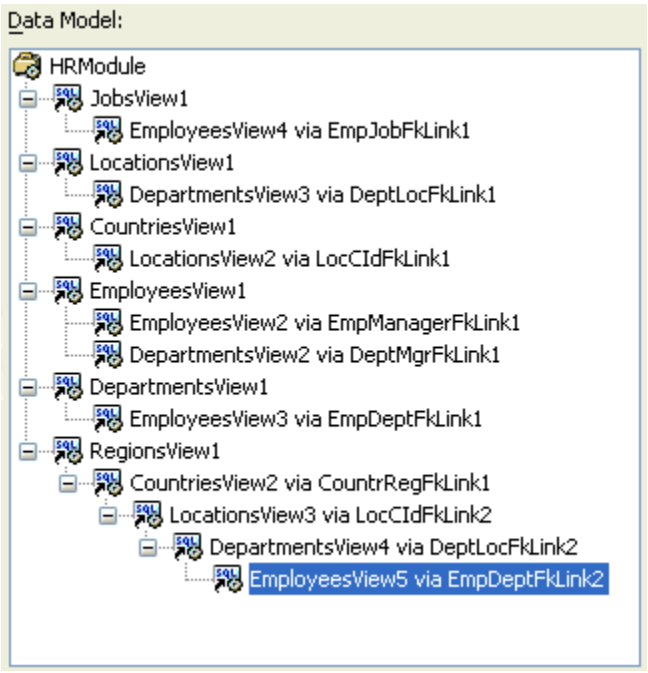


Figure 17: Default Application Module Modified to Have Five-Level Master/Detail

3. ***Optimize Your Configuration Settings for a Web Application***

The steps in this section are best-practice settings for your application module components used in web applications.

- ***Set the Business Components Locking Mode to Optimistic***

To use the recommended row-locking mode for web applications, set the `jbo.locking.mode` configuration property to `optimistic`. Do this by selecting your `HRModule` component in the Application Navigator and selecting ***Configurations...*** from the right-mouse menu. In the ***Configuration Manager*** that appears, as shown in Figure 18 click the ***(Edit)*** button to edit the `HRModuleLocal` configuration.

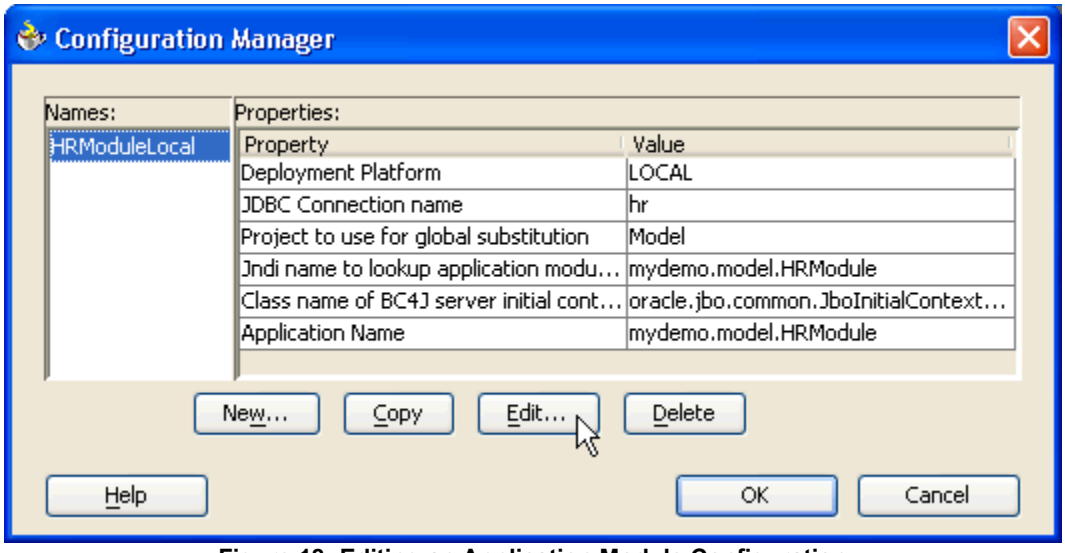


Figure 18: Editing an Application Module Configuration

As shown in Figure 19, select the ***Properties*** tab in the ***Oracle Business Component Configuration*** dialog and find the configuration property named `jbo.locking.mode` in the alphabetical list. Change its value from `pessimistic` to `optimistic` by typing the new value into the ***Value*** side of the property table, then click into another row of the property table to accept the change.

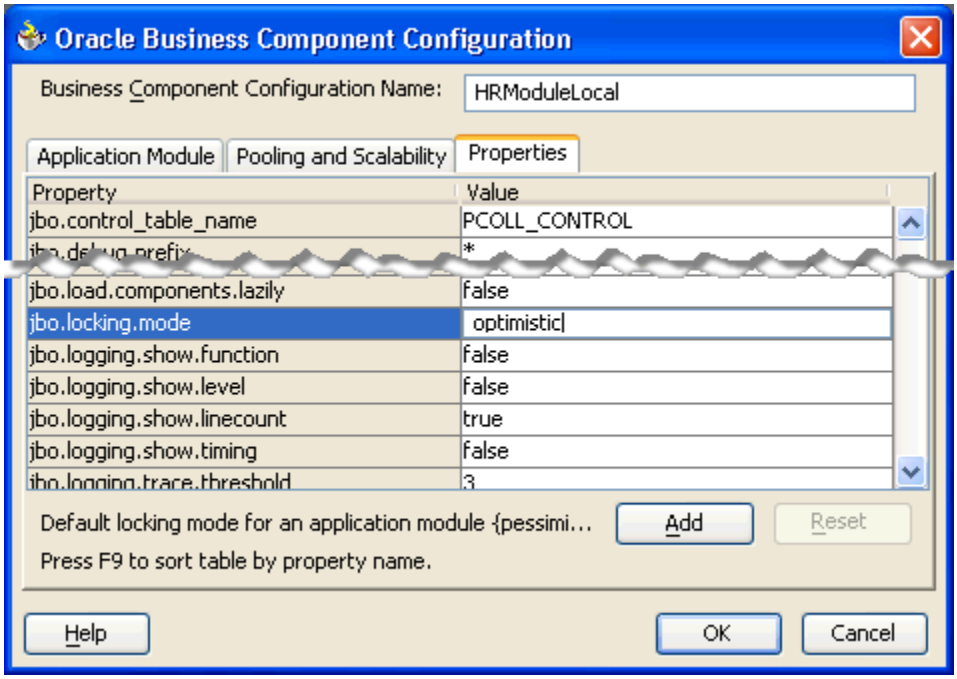


Figure 19: Setting jbo.locking.mode to optimistic

- ***Disable Failover Feature When Not Needed***

In addition, to avoid unnecessary runtime overhead on a single-server configuration — including when you are just running or debugging your application inside the JDeveloper IDE — you can disable the ADF failover behavior. With the configuration editor still open on the HRModuleLocal configuration, as shown in Figure 20 select the ***Pooling and Scalability*** tab and uncheck the ***Failover Transaction State Upon Managed Release*** checkbox.

Then click ***(OK)*** twice to save the configuration and close the configuration manager.

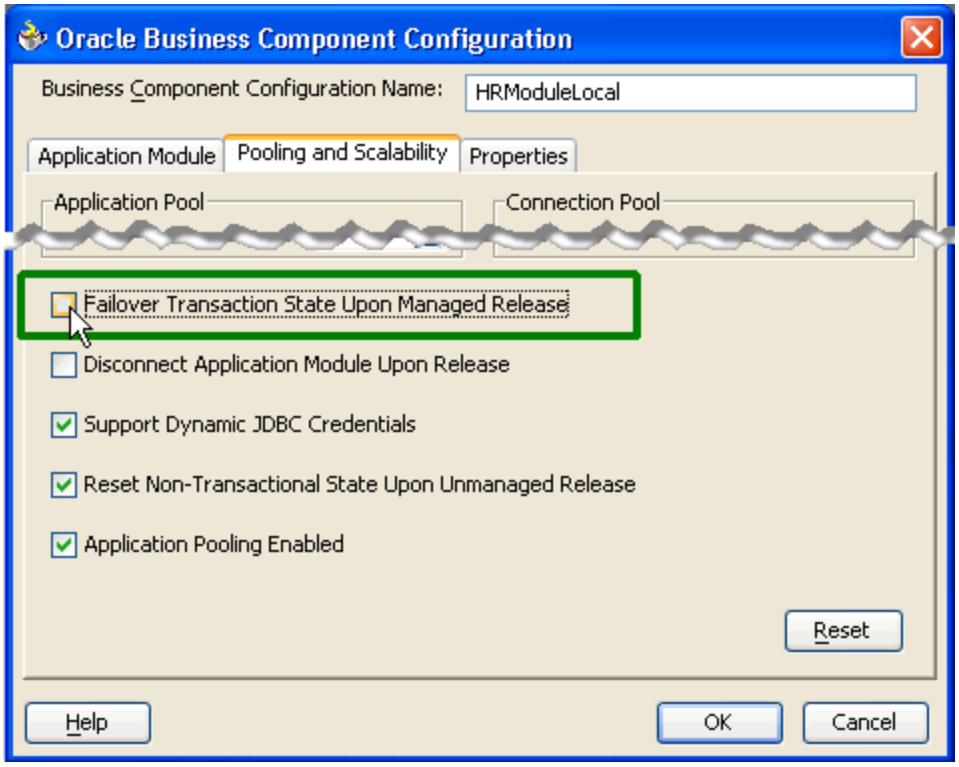


Figure 20: Disabling Failover When Not in Farm/Cluster Environment

Step 1c: Generate Default Web Tier with JHeadstart

The Oracle JHeadstart extension for JDeveloper uses metadata to capture the high-level structure of your desired web application. The JHeadstart Application Generation then uses that application structure metadata to generate the set of web pages that comprise your web application user interface. In this section we'll enable our project to use JHeadstart, ask JHeadstart to create a default application structure metadata file, then kick off the JHeadstart Application Generator to see what kind of web application we get before proceeding to tailor the output further in subsequent steps of the tutorial.

1. ***Enable JHeadstart on the ViewController Project***

   Select the `ViewController` project in the navigator and choose ***Enable JHeadstart on this Project*** from the right-mouse menu. Click ***(Next>)*** from the welcom page, then click ***(Finish)*** to proceed with enabling JHeadstart.

   As shown in Figure 21, after clicking ***(Finish)*** the first time the wizard will proceed to create a number of necessary files and configure your project appropriately to contain them. Each step it performs is listed in the text box in this panel of the wizard. When done, click ***(Finish)*** again to close the wizard.

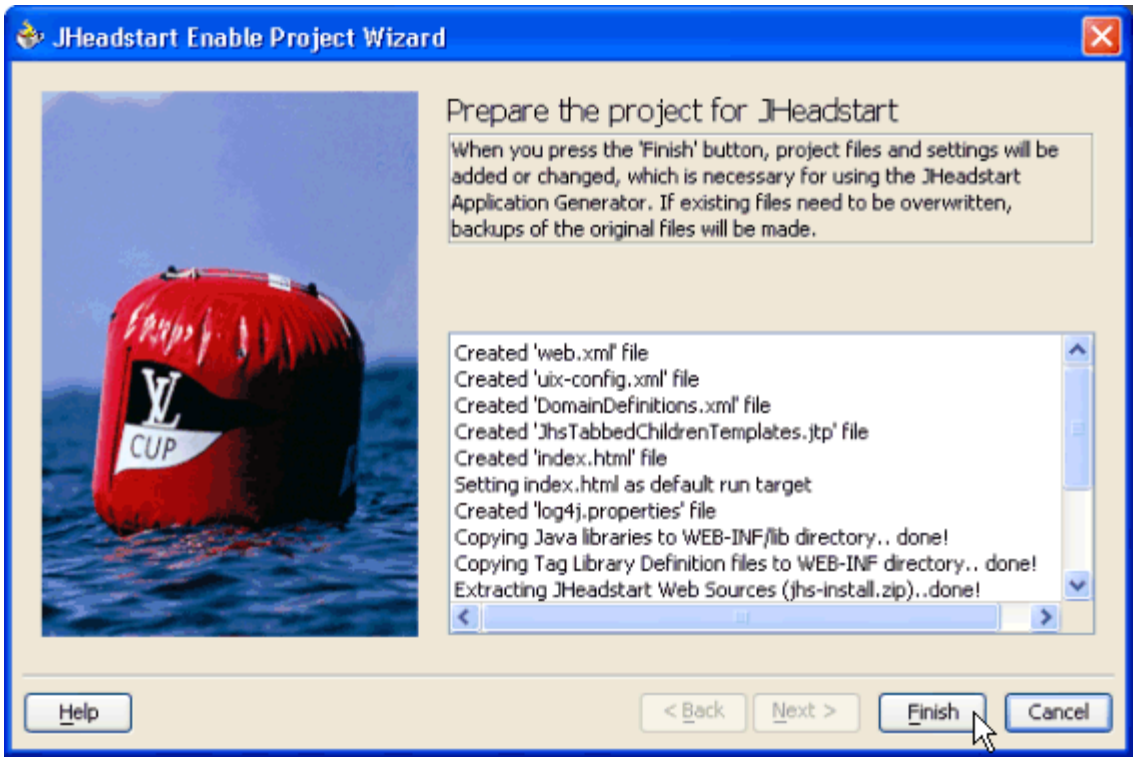


Figure 21: Enabling JHeadstart on ViewController Project

2. ***Create the Default JHeadstart Application Structure File***

   Select the `ViewController` project in the navigator, and choose the ***New JHeadstart Application Structure File*** item from the right-mouse menu. After clicking ***(Next>)*** to leave the welcome screen, on the next panel select the `HRModule` application module to use and click ***(Next>)*** again. On the following panel, as shown in Figure 22 check the ***Specify the name of the Application Structure file*** checkbox to override the default structure file name to name it `ApplicationStructure.xml`.

   By chosing a name that begins with "A", it will show up alphabetically at the start of the ***Miscellaneous Files*** folder in the ***Application Navigator***, which makes the file easier to find later. If you prefer using the ***System Navigator*** and you enable its ***Show Categories*** mode, the JHeadstart application structure file will show under the node ***JHeadstart Design Files***.

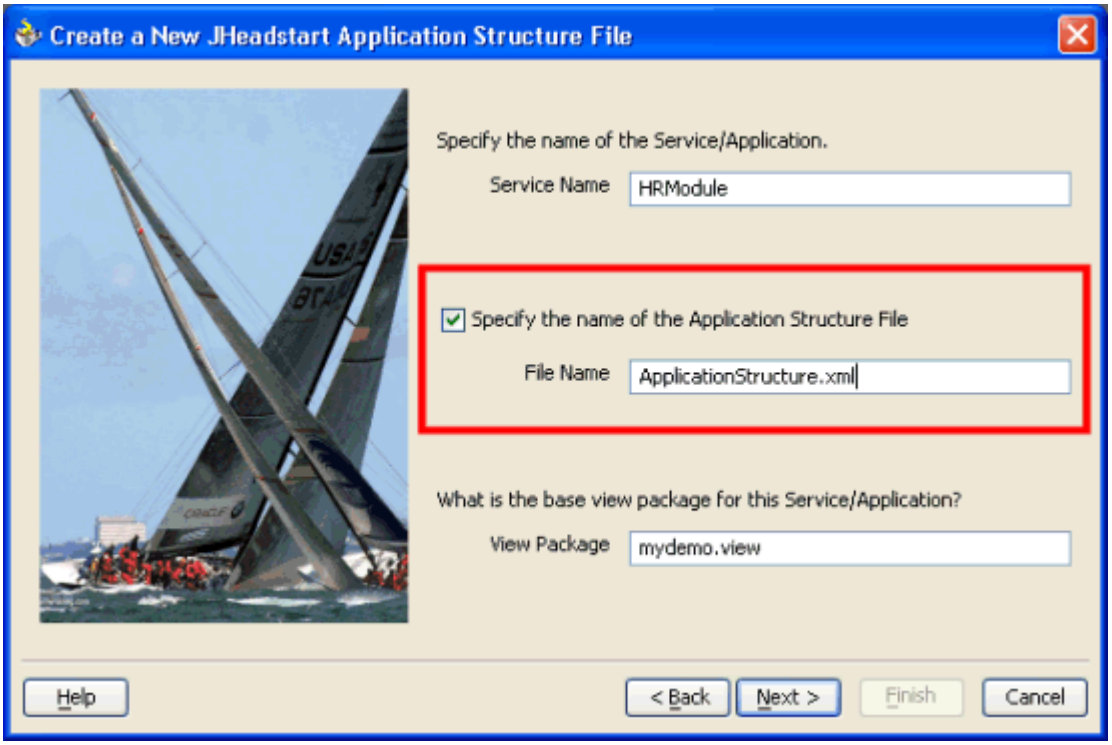


Figure 22: Overriding the Default Name of the Application Structure File

Click ***(Next>)*** again and on the final panel click ***(Finish)*** to create the new application structure file. As shown in Figure 23 the wizard will display each of the steps it performs. When done, click ***(Finish)*** again to close the wizard.

Figure 23: Creating the Default Application Structure File

3. ***Observe the Default Application Structure***

Notice that JHeadstart has used the structure of the application module's data model to create the default application

structure for the web tier. In practice you will end up iteratively changing the default application structure, but having a nice default structure to start with is a big plus as we'll see.

By selecting the `ApplicationStructure.xml` file in the navigator, and choosing ***Edit Application Structure File*** from the right-mouse menu, you will see the 5-level nesting of the view object instances for regions, countries, locations, departments, employees as shown in Figure 24. You also notice that the ***JHeadstart New Application Structure File*** wizard has created you a rich set of default lookup definitions where foreign key relationships exist in the base business components. These lookup definitions are leveraged by the JHeadstart Application Generator to generate drop-down lists or text fields with pop-up List-of-Values (LOV) windows. We'll see both kinds of these in the course of this demo.

TIP: If you accidentally double-click on the JHeadstart application structure file it will open in the code editor and you can see it's an XML file. No harm done. Just close the editor, and use the ***Edit Application Structure File*** option from the right-mouse menu instead.

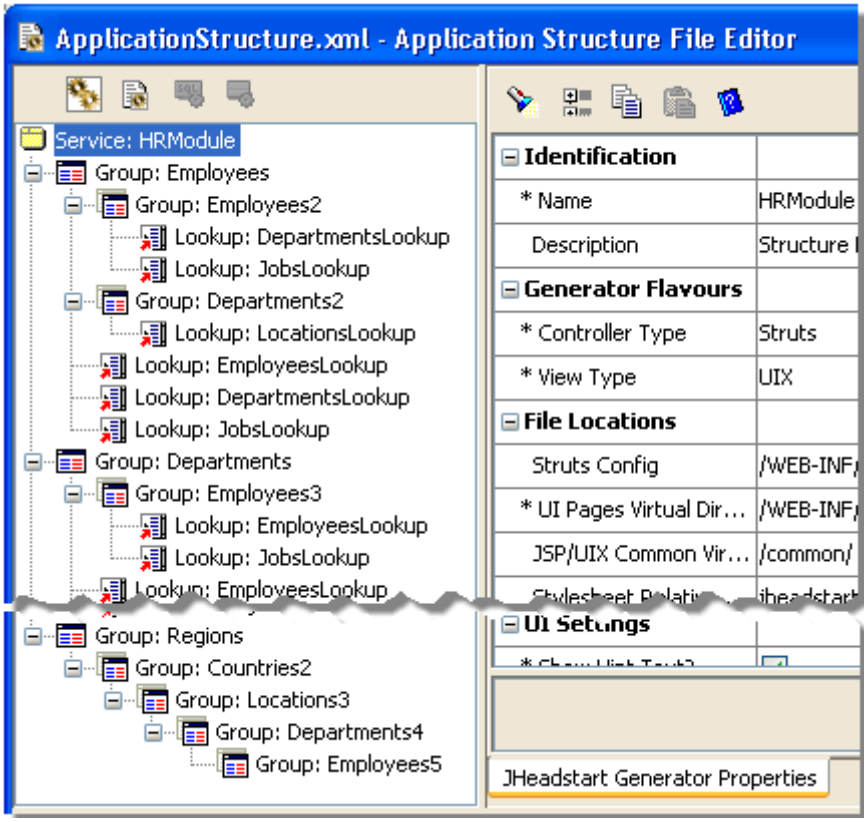


Figure 24: Application Structure Editor Showing Default Structure

4. ***Generate the Application***

   Right-click the Application Structure File and choose ***Run JHeadstart Application Generator*** from the right-mouse menu. When the generation is completed, an alert will appear letting you know the process completed ok.

Step 1d: Run the Application

Run the `ViewController` project by selecting it in the application navigator and then pressing `F11` (or clicking on the toolbar run icon). JDeveloper will launch your default browser, startup the embedded OC4J container with your web application deployed, and show you the generated "launch page" that JHeadstart creates for you as you see in Figure 25. Just click on the ***Start HRModule*** link to run the main application home page.

Figure 25: JHeadstart Application Launch Page

TIP:

If *instead* of seeing the application start page you see some kind of timeout error in your browser as shown in Figure 26, then you'll need to change a configuration property of JDeveloper's embedded J2EE container (OC4J) to avoid this problem. This typically occurs when you are on a network using DHCP to get a dynamically-assigned IP address.

Figure 26: Running Web App in JDeveloper Times Out in Browser

Before changing the necessary setting, terminate the currently running embedded server by selecting the ***Run | Terminate > Embedded OC4J Server***.

Next, select the ***Tools | Embedded OC4J Server Preferences...*** main menu item in JDeveloper, and change the ***Host Name of IP Address Used to Refer to Embedded OC4J*** setting to ***Specify Host Hame***, and enter a hostname of `localhost` as shown in Figure 27. Then, try rerunning the application and it should work this time.

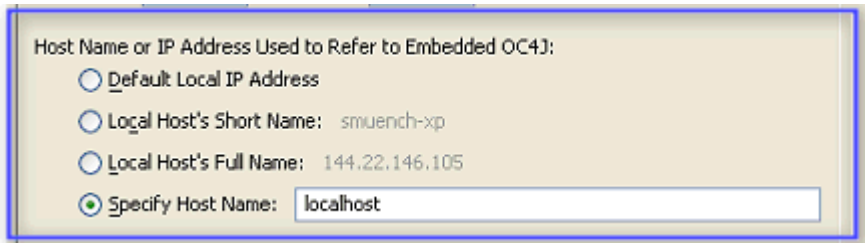


Figure 27: Changing IDE Preference for Embedded OC4J Host Name

When the application starts, you'll see your initial JHeadstart-generated web application that will look like what you see in Figure 28.

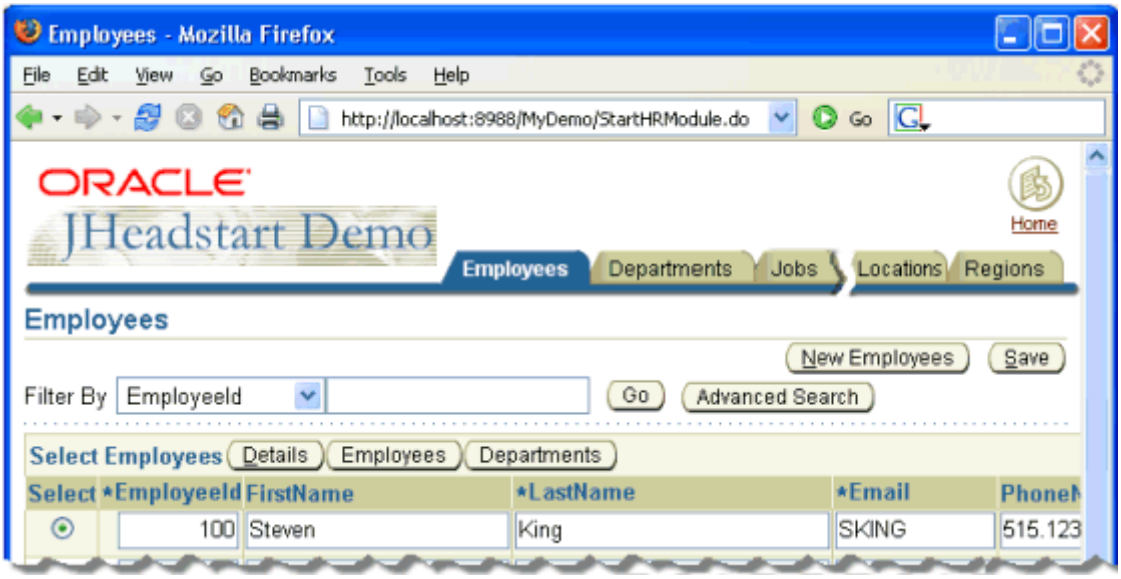


Figure 28: Initial JHeadstart-Generated Web Application

Step 1e: Observe Default Application Functionality

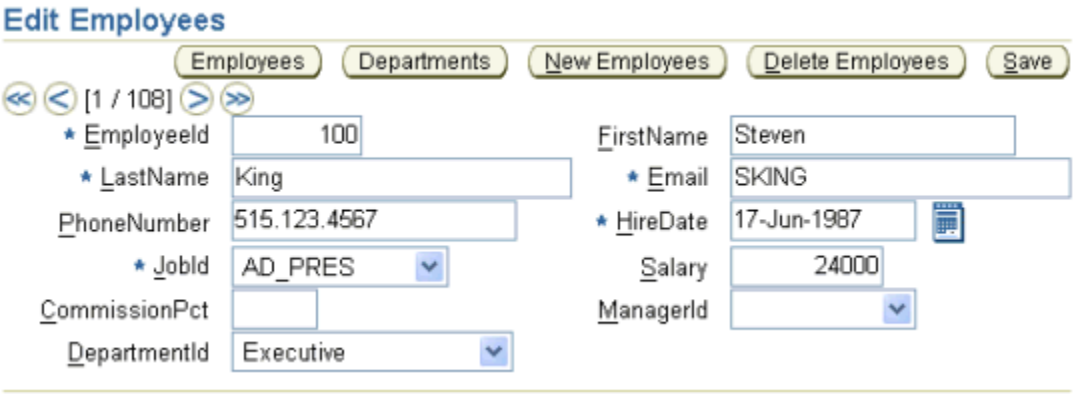
By playing with the default application yourself, you can experience some of the built-in features that Oracle ADF and JHeadstart support by default, and which the Oracle JHeadstart Application Generator generates for you based on your declarative application structure file.

Highlights of these features include:

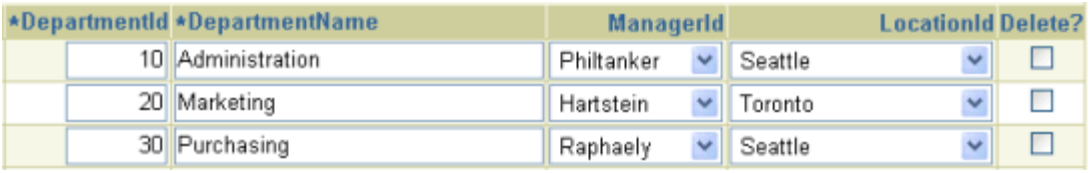
- ***Switch Between Top-Level Application Functionality with Tabs***

- ***Browse Data and Drill-Down to Details or Related Information***

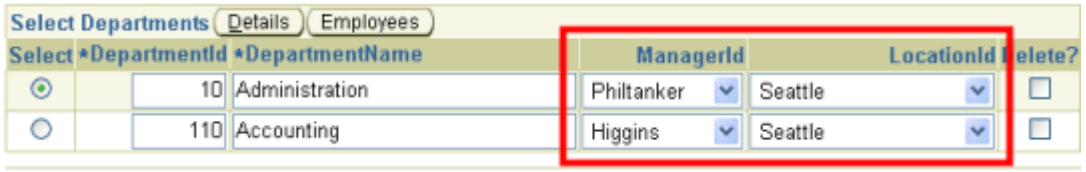
- ***Create and Edit Data for Any Table Out of the Box***



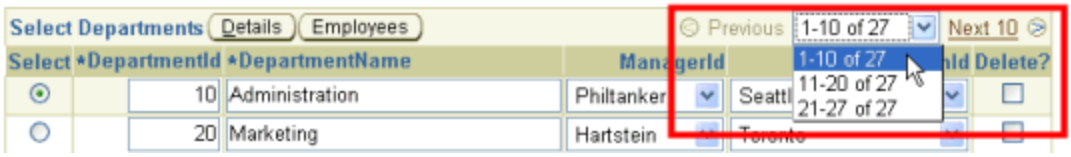
- ***Insert, Update, and Delete Multiple Rows on a Page***



- ***Select Related Data from Automatically Created Lookups***



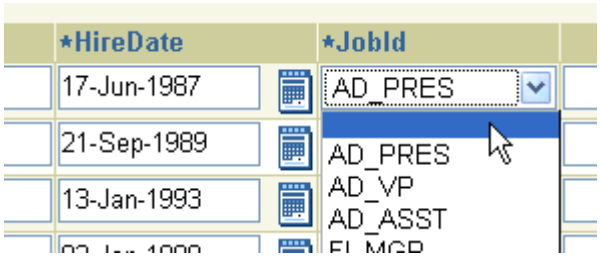
- ***Scroll Through Data Without Refreshing Entire Page***



- ***Rapidly Find Data Using Quick Search Region***

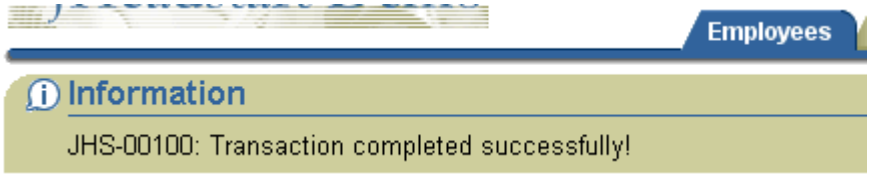
- ***Search More Precisely Using Advanced Search Region***

- ***Easily Set Optional Fields to Null Using Drop-Down Lists***



- ***Avoid Accidentally Losing Pending Changes When Switching Top-Level Application Function***

- ***Never Wonder Whether Changes are Saved with Positive User Feedback***



Step 2: Change Layout Styles and Query Behavior

In this step of the demo, we'll change a number of declarative application structure properties about the `Employees`, `Departments`, `Jobs`, and `Regions` groups to affect how the JHeadstart Application Generator generates the web tier pages. We'll wait until making them all before re-running the application generator.

To make the changes described here, make sure you have the ***JHeadstart Application Structure File Editor*** dialog open. If you don't, just locate the `ApplicationStructure.xml` file in your `ViewController` project and select ***Edit Application Structure File*** from the right-mouse menu.

NOTE: Since the ***JHeadstart Application Structure File Editor*** dialog is modeless, you can keep it open and `Alt+Tab` between it and the main JDeveloper IDE window.

Step 2a: Change How Employees Group Gets Generated

- ***Use "Detail Disclosure" to Hide Less Common Employee Fields***

  As shown in Figure 29, in the `Employees` group, check the ***Detail Disclosure*** checkbox (in the ***Table Layout*** property category).

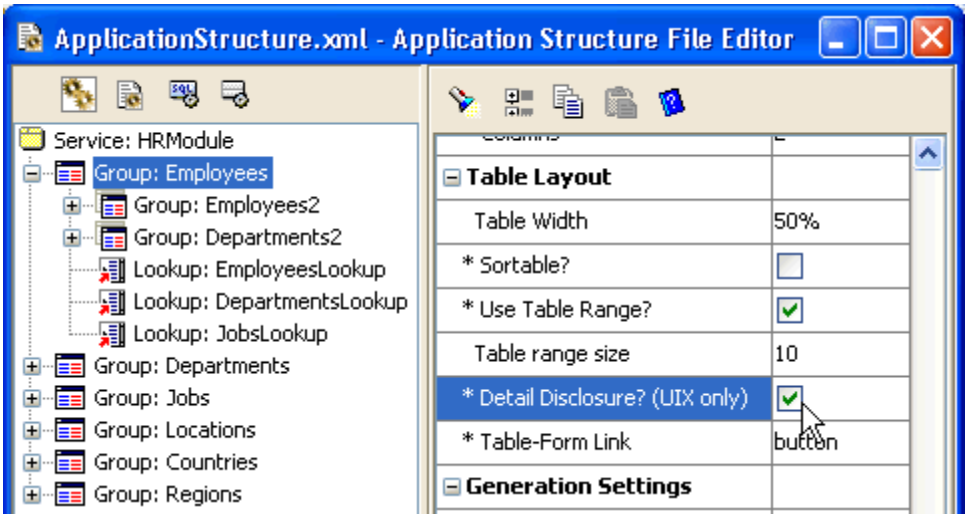


Figure 29: Selecting Detail Disclosure for Employees Group

TIP: If you have trouble finding a property, you can click on the "Searchlight" find icon in the ***JHeadstart Application Structure File Editor*** and type in some characters of the name you're looking for.

TIP: Every property in the JHeadstart editors is documented with a helpful usage message in the help zone below the property table as shown in Figure 30.

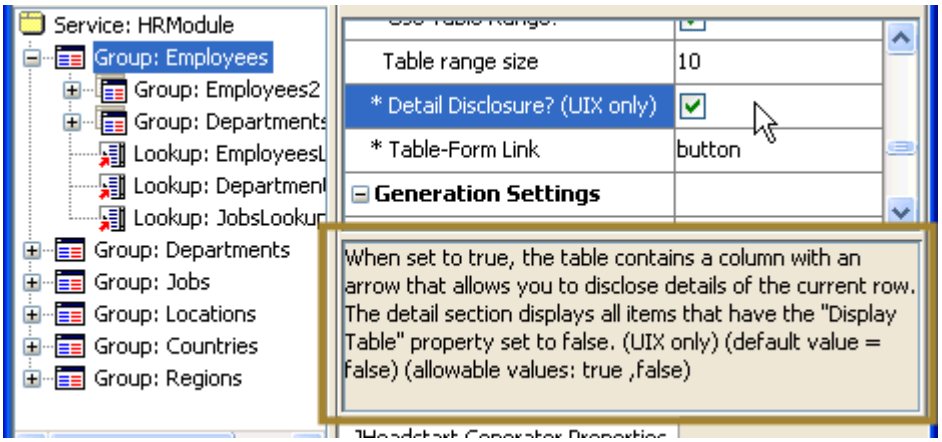


Figure 30: Detailed Usage Information for Every Property

Next, we need to indicate which attributes should be hidden by default when displayed in a table. Since these are attribute-specific properties, we edit them on the view object level for the view object related to our `Employees` group.

To accomplish this, select the `Employees` group, and click the button with the View Object icon as shown in Figure 31 to open the ***JHeadstart ADF BC Properties Editor*** dialog (also a modeless dialog).

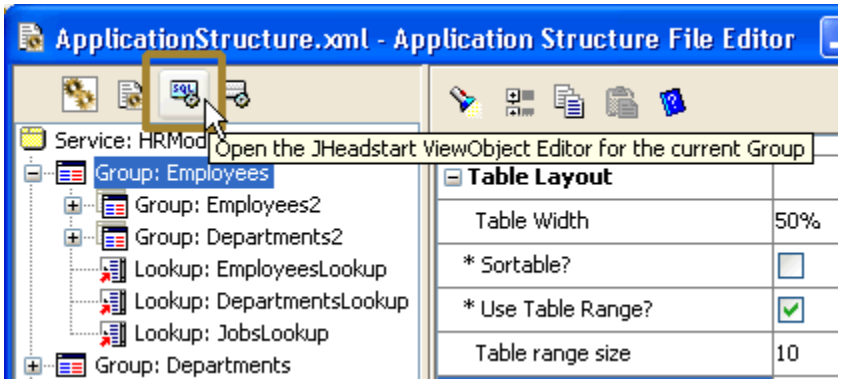


Figure 31: Opening the JHeadstart View Object Editor Dialog

In the ***JHeadstart ADF BC Properties Editor***, set the ***Display in Tables*** property to `false` for all attributes *except*: `EmployeeId`, `FirstName`, `LastName`, `Email` and `Salary`. As shown in Figure 32, we can can multi-select attributes using the mouse in combination with the `Shift` or `Control` keys, then set the desired property.

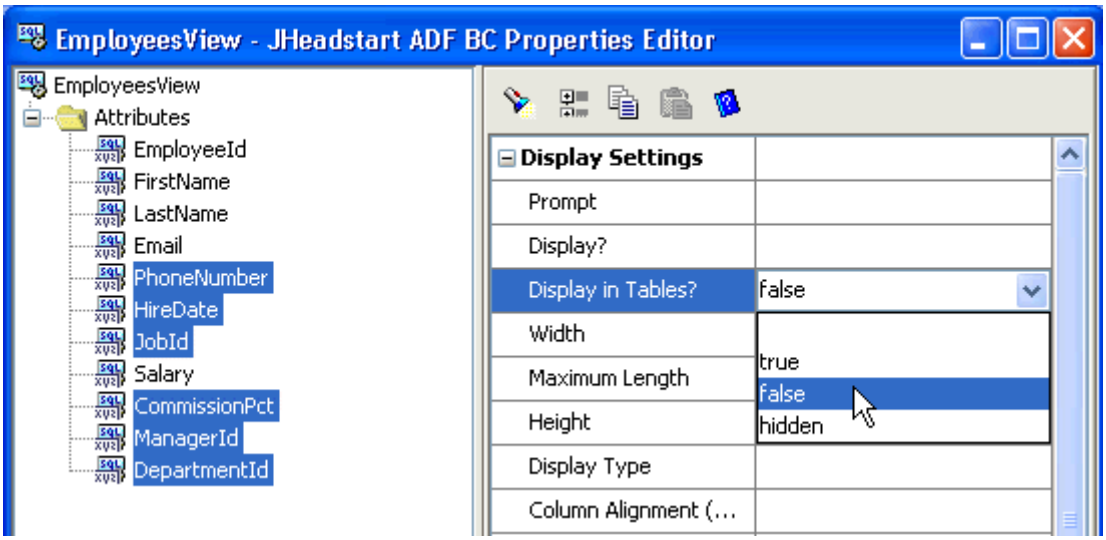


Figure 32: Quickly Setting a Property for Multiple Attributes

- ***Set the Prompt of DepartmentId to be More User-Friendly***

  While still in the ***JHeadstart ADF BC Properties Editor*** for `EmployeesView`, in the ***Display Settings*** category, as shown in Figure 33 set the ***Prompt*** property of `DepartmentId` to be the more user-friendly string "`Department`". This will make more sense to the user since what they see at runtime is a dropdown list of department names for this field.

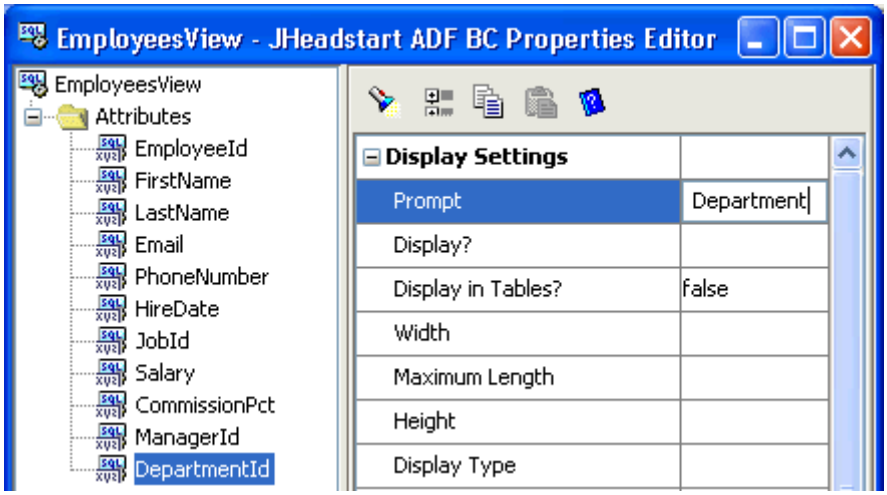


Figure 33: Making DepartmentId Prompt More User-Friendly

- ***Allow End-User to Choose Query Operator for LastName***

  Set the ***Query Operator*** property of the `LastName` attribute to "`setByUser`" as shown in Figure 34. This property is in the ***Query Settings*** category.

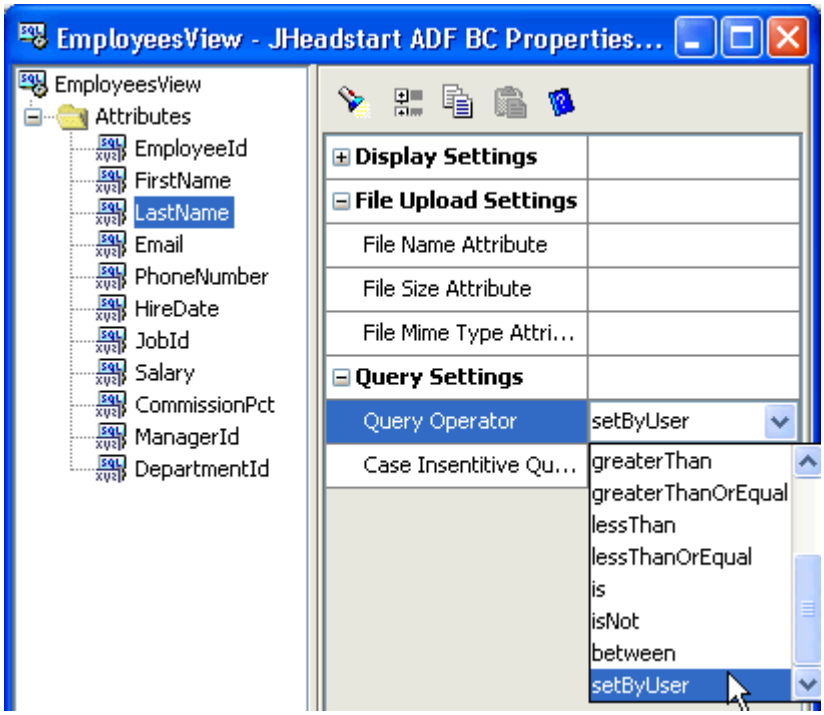


Figure 34: Allowing User to Choose LastName Query Operator

- ***Allow End-User to Perform "Between" Search on Salary***

  Set the ***Query Operator*** property of the `Salary` attribute to "`between`" as shown in Figure 35.

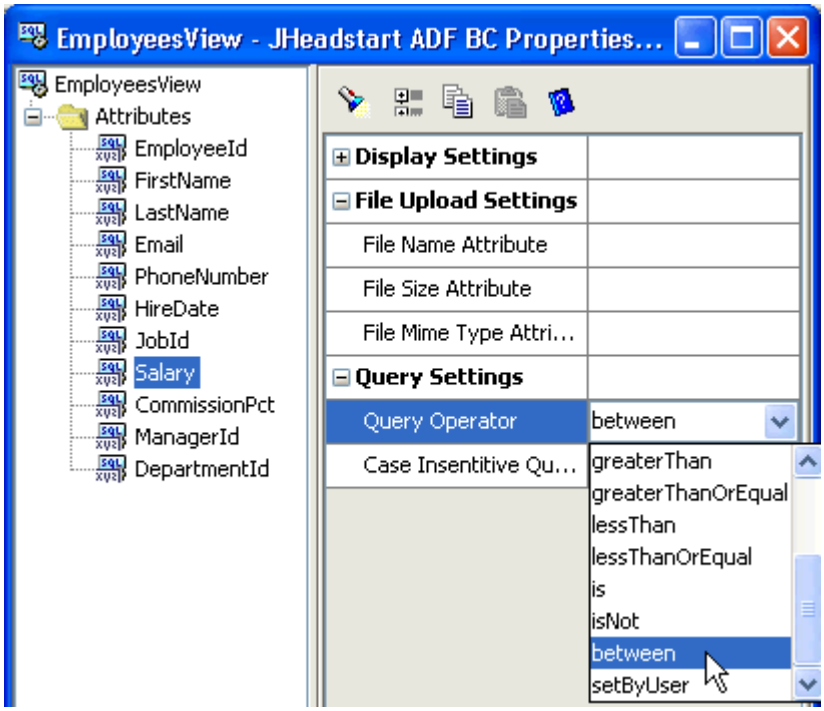


Figure 35: Allowing User to Search on Salary Range

- ***Make the Table Display for Employees be Browse-Only***

  Back in the ***JHeadstart Application Structure File Editor***, with the `Employees` group still selected, as shown in Figure 36 uncheck the ***Multi-Row Insert Allowed***, ***Multi-Row Update Allowed*** and ***Multi-Row Delete Allowed*** checkboxes for the `Employees` group. These properties are in the ***Operations*** category.

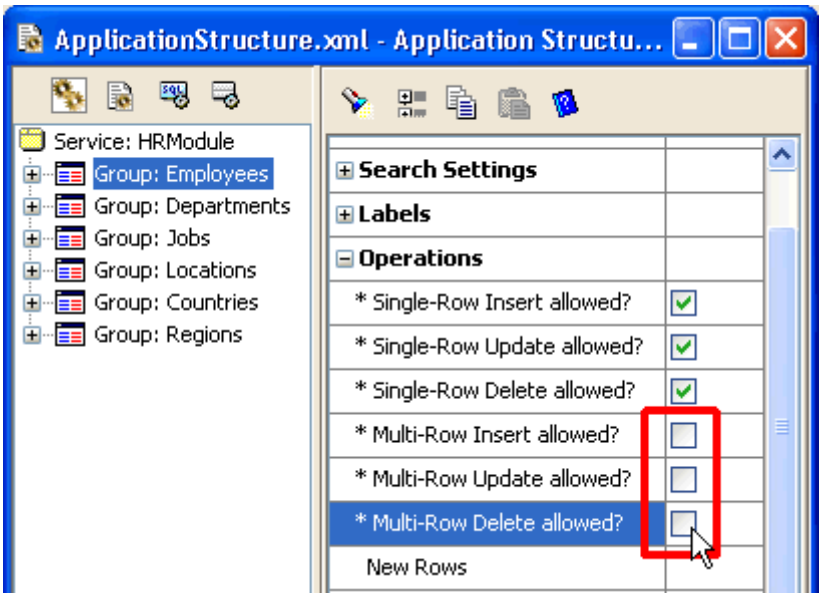


Figure 36: Making the Table Display for Employees Browse-Only

- ***Force User to Find Employees Instead of Auto-Querying Them***

  Uncheck the ***Auto Query*** checkbox in the ***Search Settings*** category, as shown in Figure 37.

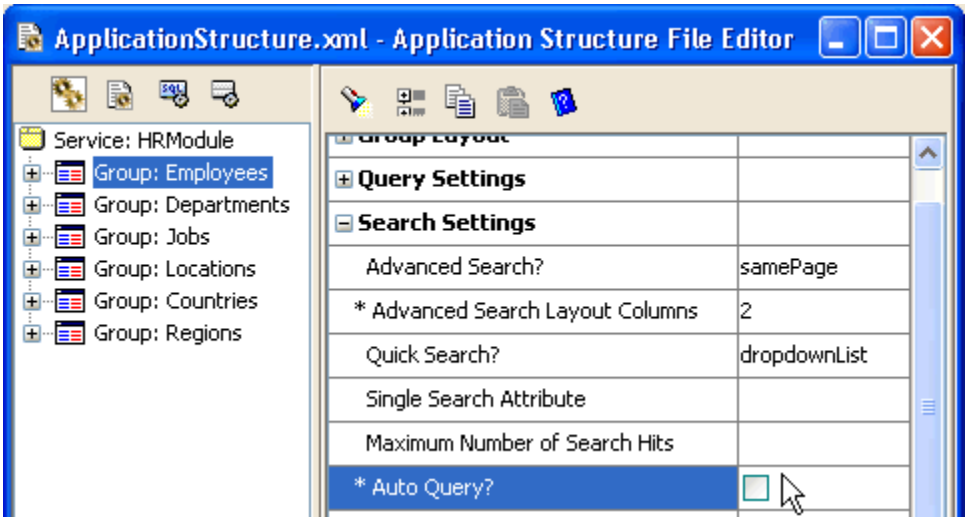


Figure 37: Avoiding an Automatic Employees Query

- ***Force User to Refine Search Criteria to be More Selective***

  As shown in Figure 38, set the ***Maximum Number of Search Hits*** to `50`. This property is also in the ***Search Settings*** category.

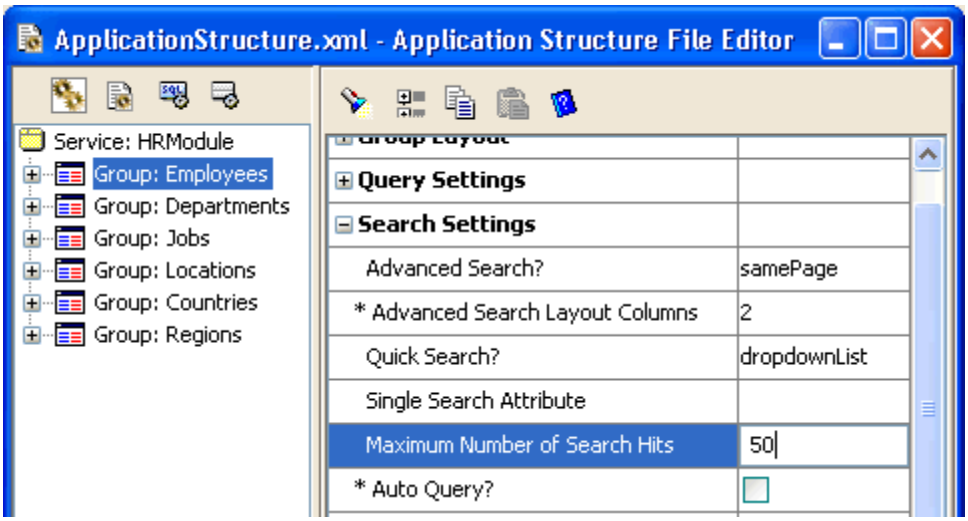


Figure 38: Limiting the Maximum Number of Employees Search Hits

- ***Change Tab Name and Display Titles of Child Groups***

  Expand the top-level `Employees` group and set the ***Tabname***, ***Display Title (Plural)***, and ***Display Title (Singular)*** properties (in the ***Labels*** category) as shown in Figure 39 for the `Employees2` child group. Since this group represents employees managed by the current employee in the `Employees` group, we choose a more understandable name like "Subordinates".

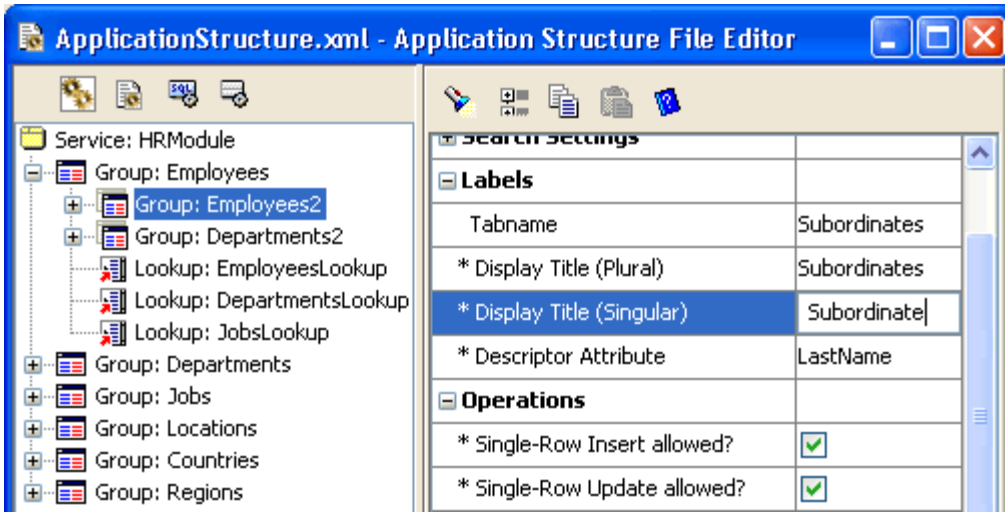


Figure 39: Changing Tab and Display Titles for Employees2 Child Group

Repeat the same steps to change the ***Tabname***, ***Display Title (Plural)***, and ***Display Title (Singular)*** properties of the `Departments2` group to be "Managed Departments" as shown in Figure 40.

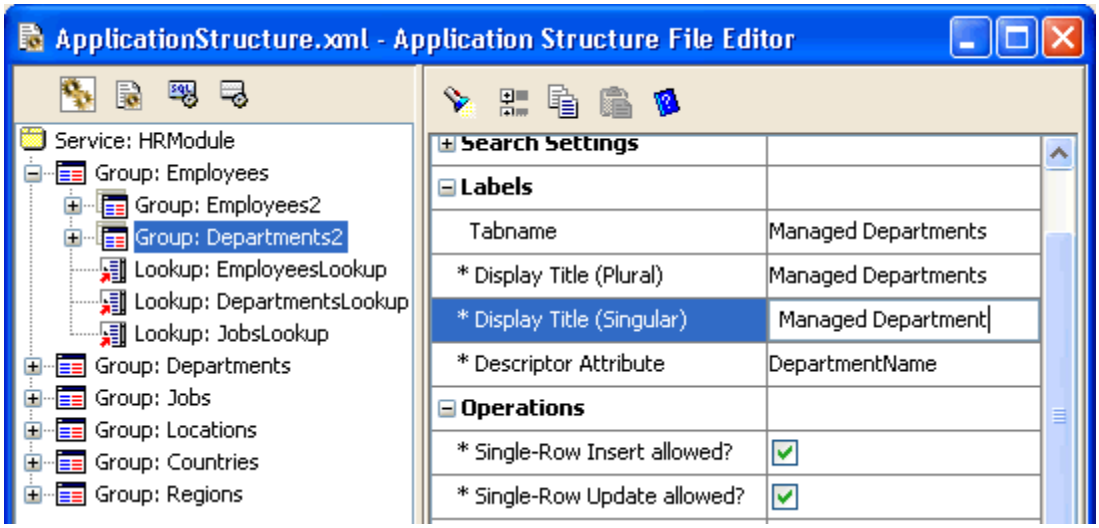


Figure 40: Changing Tab and Display Titles for Departments2 Child Group

Step 2b: Change How the Departments Group Gets Generated

- ***Allow End-User to Sort Columns on the Departments Browse Page***

  After selecting the top-level `Departments` group in the ***JHeadstart Application Structure File Editor***, check the ***Sortable*** checkbox in the ***Table Layout*** category as shown in Figure 41.

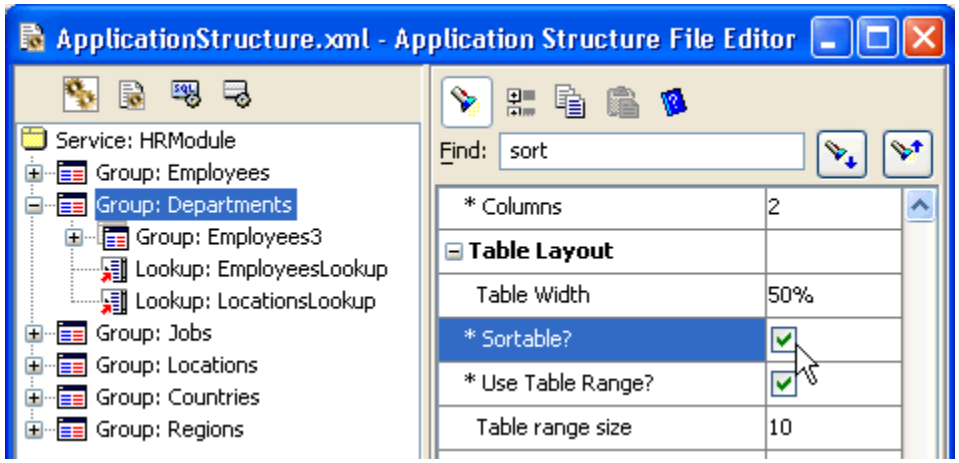


Figure 41: Making the Table Columns Sortable in the Departments Group

- ***Display Employees in Department on Same Page***

  In child group `Employees3` check the ***Same Page*** checkbox in the ***Group Layout*** category as shown in Figure 42.

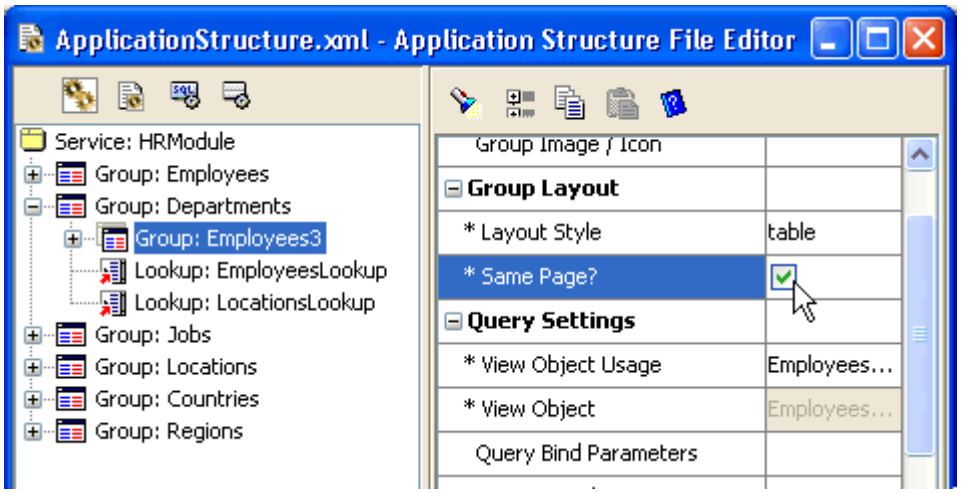


Figure 42: Showing a Child Group's Information on Same Page as Parent

Step 2c: Change How the Jobs Group Gets Generated

To apply the changes in this section, make sure you've selected the top-level `Jobs` group in the ***JHeadstart Application Structure File Editor***.

- ***Allow End-User to Select Job to Edit Using a List Display***

  As shown in Figure 43, select the `Jobs` group and set its Layout Style property to `select-form`.

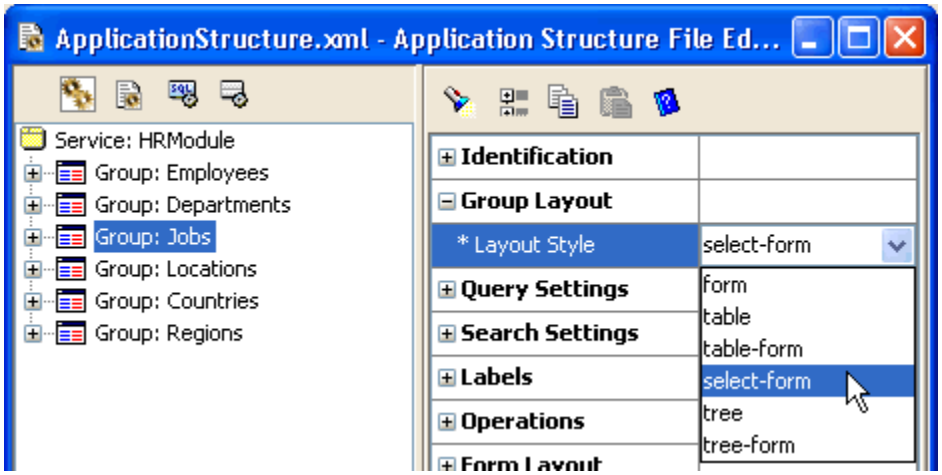


Figure 43: Using a List to Select the Current Job to Edit

- ***Disable Advanced Search for Jobs***

  As shown in Figure 44, set the ***Advanced Search?*** property of the Jobs group to `none`. The property is in the ***Search Settings*** category.

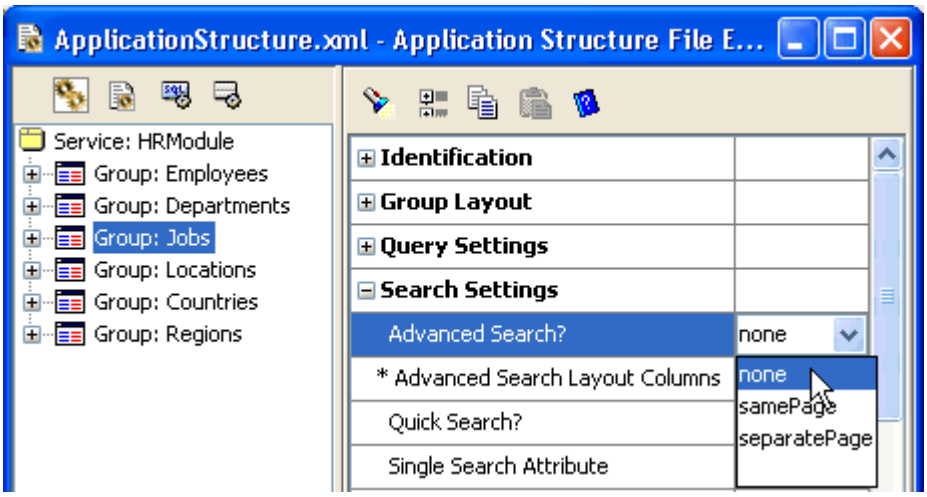


Figure 44: Disabling Advanced Search for Jobs Page

- ***Limit Quick Search Feature to the JobTitle Field***

  First, set the ***Quick Search?*** property of the `Jobs` group to `singleSearchField` as shown in Figure 45.

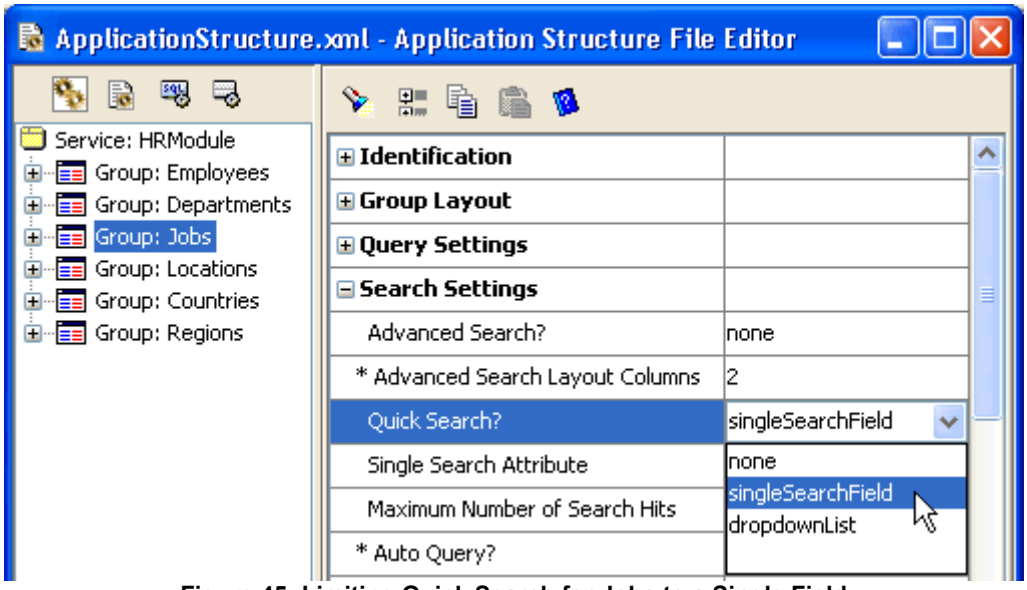


Figure 45: Limiting Quick Search for Jobs to a Single Field

Then, set the ***Single Search Attribute*** property to `JobTitle` as shown in Figure 46. These properties are also in the ***Search Settings*** category.

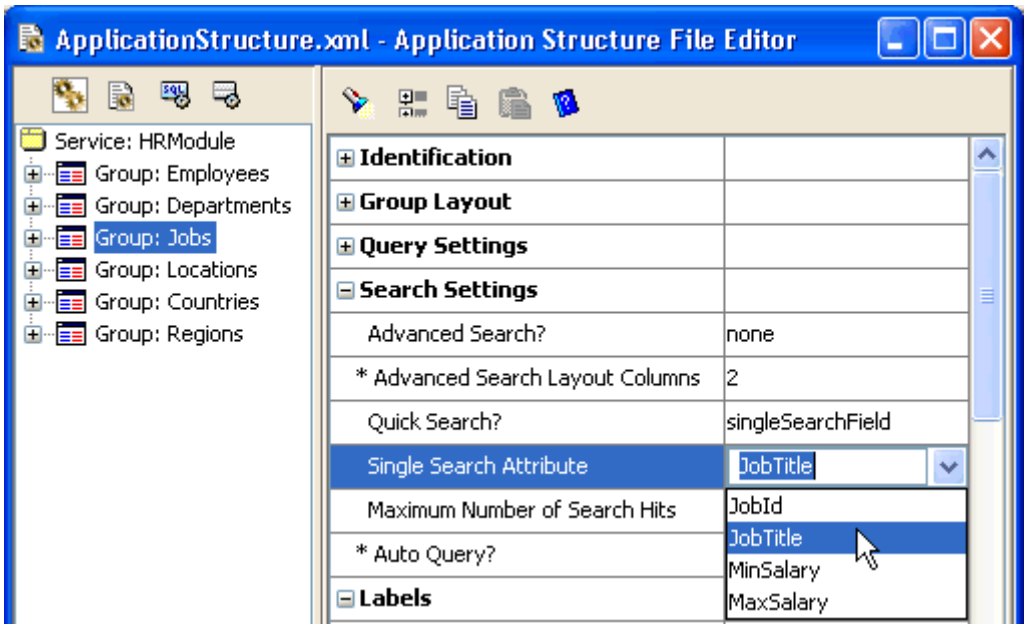


Figure 46: Specifying Attribute to Use for Single-Field Quick Search

- ***Define JobTitle as the Descriptor Attribute for the List***

  For each group you can specify the attribute JHeadstart will use to show context information. This attribute is called the Descriptor Attribute. The ***New Application Structure File Wizard*** derives a default ***Descriptor Attribute*** for each Group based on its underlying view object. If you want to use a different attribute, just set the Descriptor Attribute property yourself to the attribute you prefer.

  As shown in Figure 47, set the ***Descriptor Attribute*** property of the `Jobs` group to `JobTitle`.

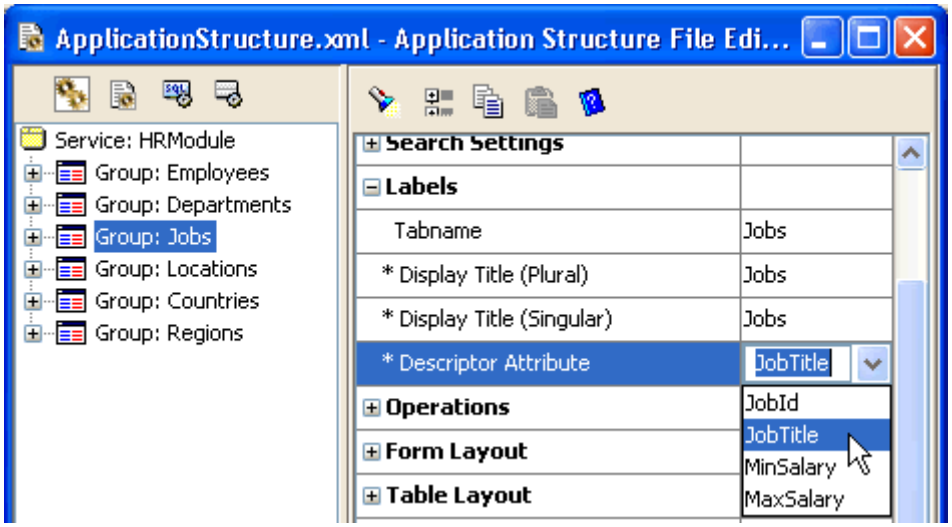


Figure 47: Changing the Descriptor Attribute

Step 2d: Change How the Regions Group Gets Generated

We're going to change the `Regions` group, and its four child groups, to display as a tree with multi-level form editing by doing the following steps...

- ***Make Regions Edit Fields Layout in a Single Column***

  As shown in Figure 48, set the ***Columns*** property of the `Regions` group to `1` in the ***Form Layout*** category.

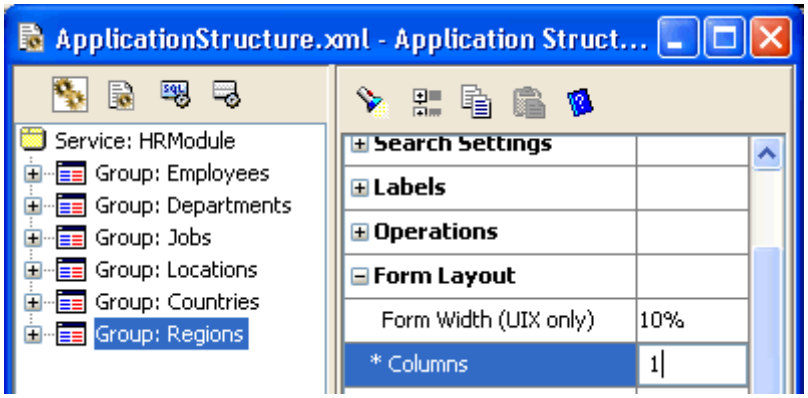


Figure 48: Setting Edit Form Fields to Layout in Single Column

- ***Disable Searching on Regions Group***

  The user will see all the regions in the tree so searching won't be that useful in this case. We'll disable both Quick Search and Advanced Search for the `Regions` group by setting the ***Quick Search*** and ***Advanced Search*** properties both to `none` as shown in Figure 49.

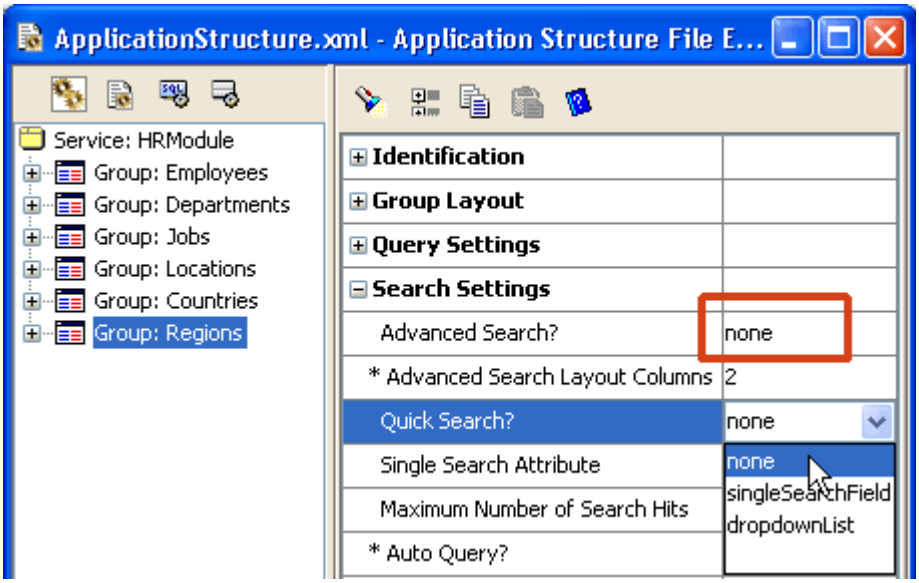


Figure 49: Disabling Search for Regions Group

- ***Shorten Display Width of RegionId***

  Click on the View Object toolbar icon to bring up the ***JHeadstart ADF BC Properties Editor*** for the group's `RegionsView` view object. Then set the ***Width*** property in the ***Display Settings*** category for the `RegionId` attribute to `5` as shown in Figure 50.

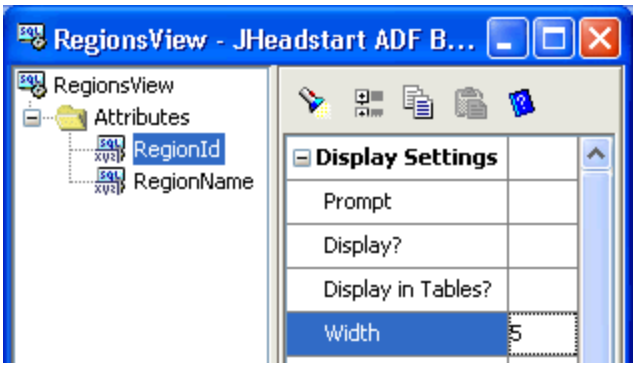


Figure 50: Setting the Display Width of an Attribute

- ***Change Regions Group to Display as a Tree with Edit Form***

  First we set the ***Descriptor Attribute*** of the group to use the `RegionName` as shown in Figure 51.

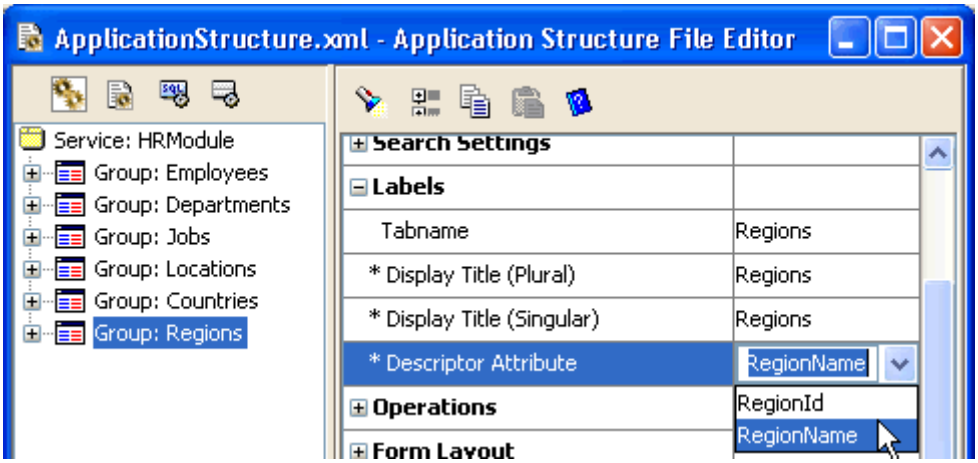


Figure 51: Setting RegionName as Descriptor Attribute

Then as shown in Figure 52 we set the ***Layout Style*** property to `tree-form`, set the ***View Object Usage*** (to be used for editing) to `RegionsViewLookup`, and set the ***Tree View Object Usage*** to `RegionsView1`.

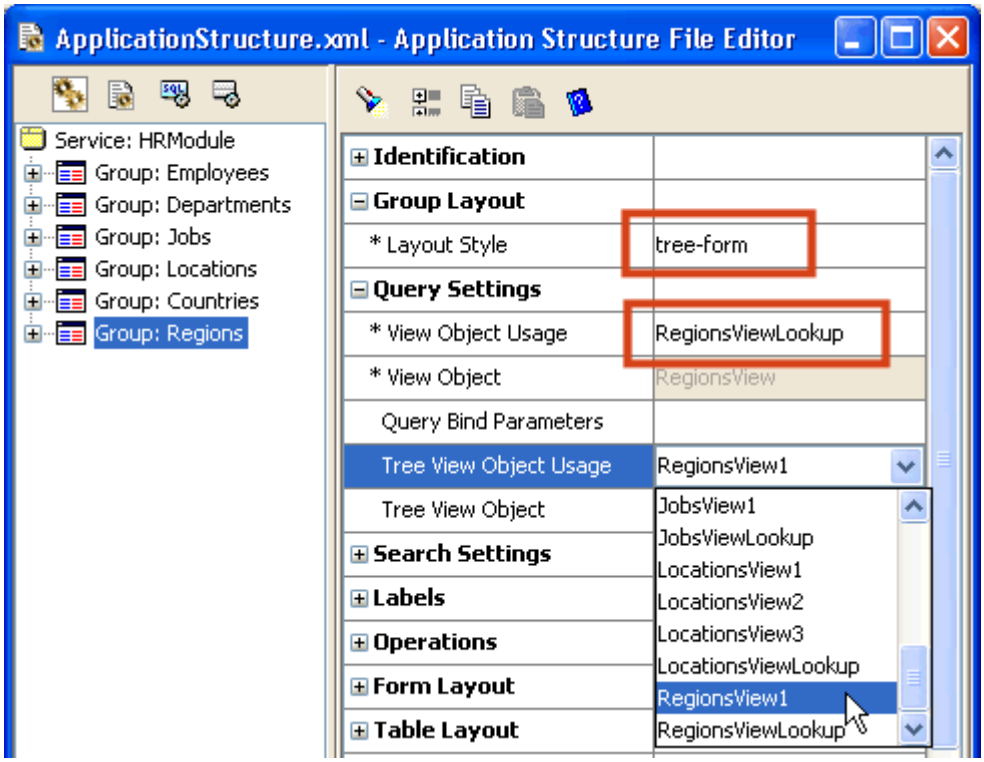


Figure 52: Setting Up Regions to Display as a Tree

- ***Change Countries2 Group to Display as a Tree with Edit Form***

Next we setup the `Countries2` child group to also display as a tree by setting the four properties as shown in Figure 53. Specifically, we set the ***Layout Style*** to `tree-form`, the ***View Object Usage*** to `CountriesViewLookup`, the ***Tree View Object Usage*** to `CountriesView2`, and the ***Descriptor Attribute*** to `CountryName`

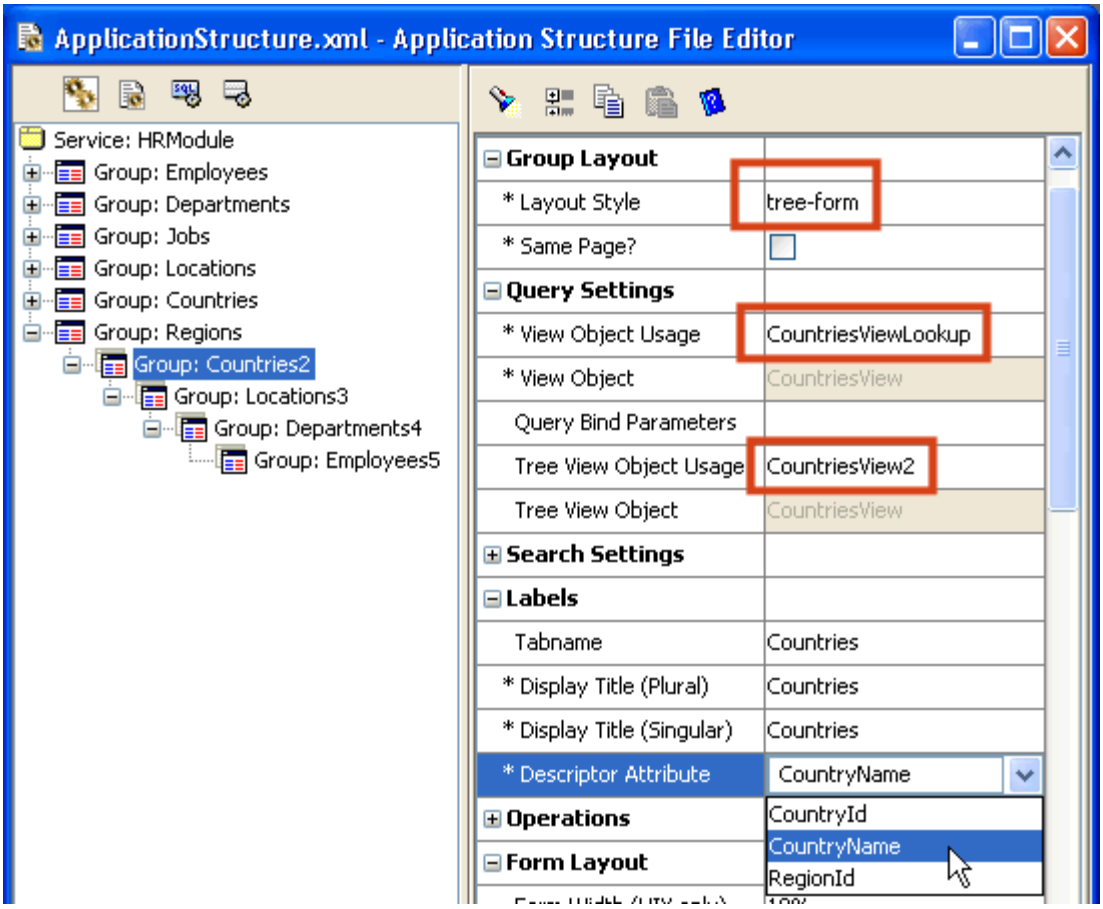


Figure 53: Setting up Countries2 Group as a Tree

- ***Repeat to Change Remaining Child Groups to Tree***

  After selecting the 3rd-level child group `Locations3`, we set its ***Layout Style*** to `tree-form`, its ***View Object Usage*** to `LocationsViewLookup`, the ***Tree View Object Usage*** to `LocationsView3`. The ***Descriptor Attribute*** is already defaulted to `City`, so we don't need to change it.

  Next, after selecting the 4th-level child group `Departments4`, we set the ***Layout Style*** to `tree-form`, the ***View Object Usage*** to `DepartmentsViewLookup`, the ***Tree View Object Usage*** to `DepartmentsView4`. The ***Descriptor Attribute*** is already defaulted to `DepartmentName`, so we don't need to change it.

  Lastly, after selecting the 5th-level `Employees5` child group, we set the ***Layout Style*** to `tree-form`, the ***View Object Usage*** to `EmployeesViewLookup`, the ***Tree View Object Usage*** to `EmployeesView5`. The ***Descriptor Attribute*** is already defaulted to `LastName`, so we don't need to change it.

Step 2e: Regenerate and Run the Application

At this point we're done making our goodly number of iterative, declarative changes to the application structure, so all that's left is to regenerate the application using the JHeadstart Application Generator, and running it to see the effects of our changes.

- ***Regenerate the Application***

  Right-click the Application Structure File and choose ***Run JHeadstart Application Generator*** from the right-mouse menu. When the generation is completed, an alert will appear letting you know the process completed ok.

- ***Run the Application***

  Run the `ViewController` project by selecting it in the application navigator and then pressing `F11` (or clicking on the toolbar run icon).

After regenerating, once the application is running in your default browser, we can try the following things to see how our changes to the application structure were realized in the generated pages...

- ***User Needs to Search for Desired Employees***

  On the initial ***Employees*** tab, notice that employees no longer get auto-queried, as shown in Figure 54.

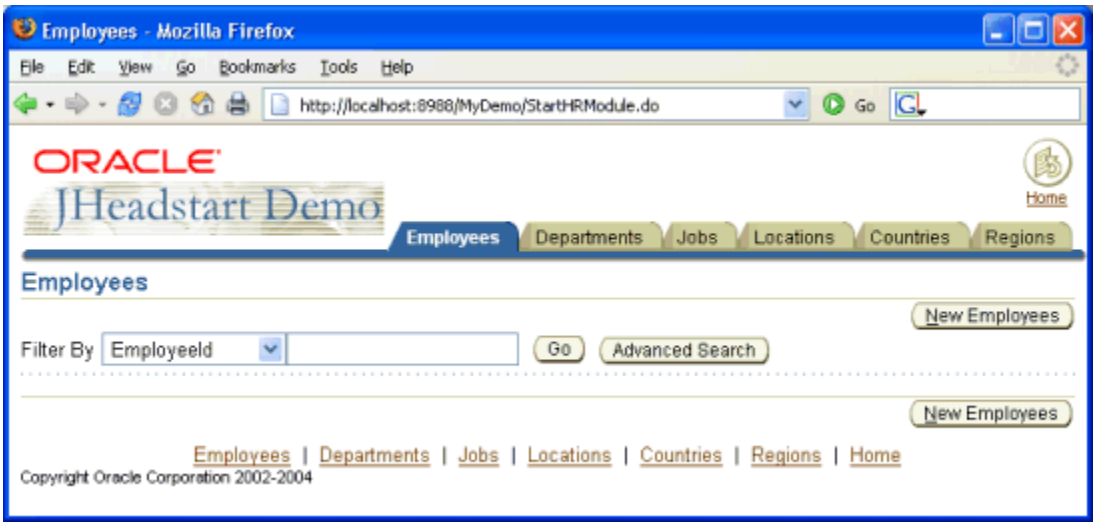


Figure 54: User Needs to Search for Desired Employees

- ***User Warned to Refine Employees Search Criteria***

  Using the Quick Search region for Employees, press the ***(Go)*** button without entering any search criteria. This would cause 108 rows to be queried, and since that is greater than our limit of 50, you get an error as shown in Figure 55.

Figure 55: User Warned to Refine Employees Search Criteria

- ***Employee Table is Browse-Only with Detail Disclosure***

  Set the ***Filter By*** field to `LastName` name, enter a search criteria of "`p`", and press ***(Go)***. As shown in Figure 56, the Employees table is now browse-only and supports a ***Hide***/***Show*** button in each row to expand or collapse the visibility of the less frequently used detail information.

  Also note in the expanded detail information for Joshua Patel that the prompt for the `DepartmentId` field is now ***Department*** as we set above, as well as the custom names ***Subordinates*** and ***Managed Departments*** for the buttons that will drill-down to detail information.

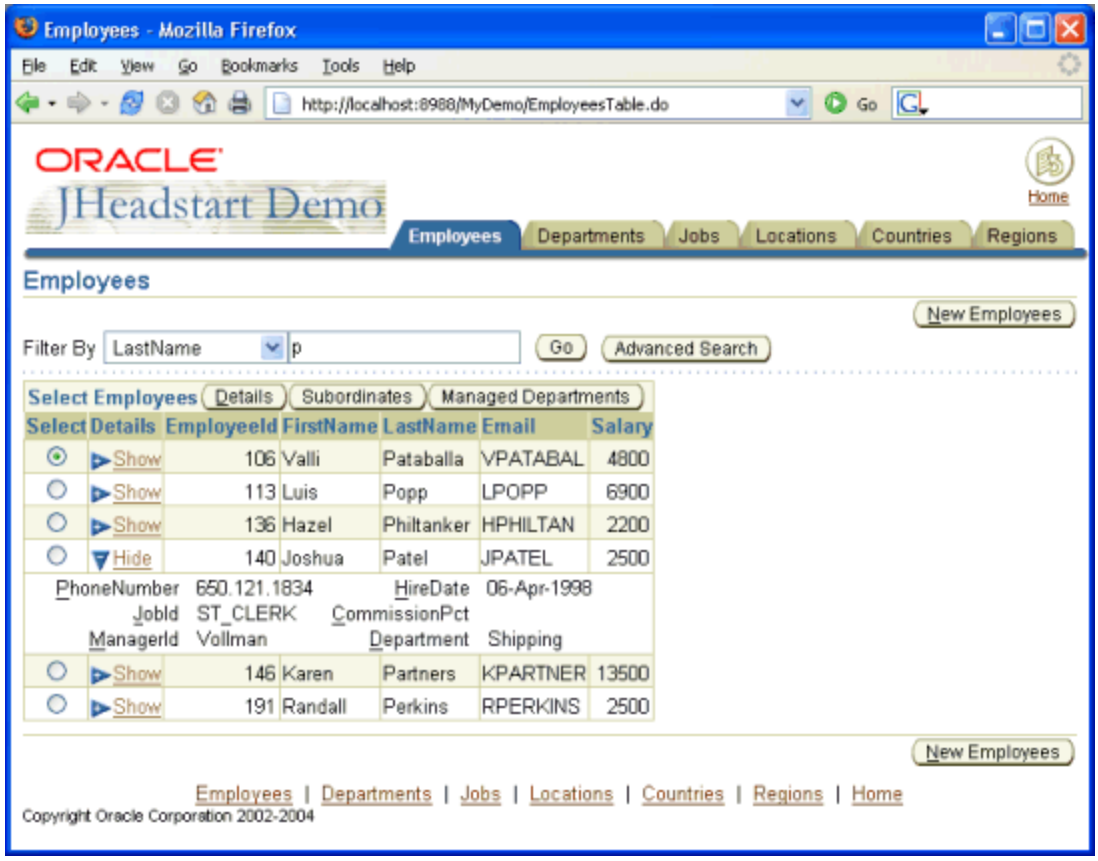


Figure 56: Employee Table is Browse-Only with Detail Disclosure

- ***User Can Perform Advanced Search With Custom Criteria Treatment***

  Click the ***Advanced Search*** button next to the Quick Search area to see the advanced search criteria for Employees. Notice as shown in Figure 57 that we can set the query operator for `LastName` to "***contains***" and enter a criteria like "`ta`" to find any last name that contains those consecutive letters. We can enter low and high values to perform a between search on Salary.

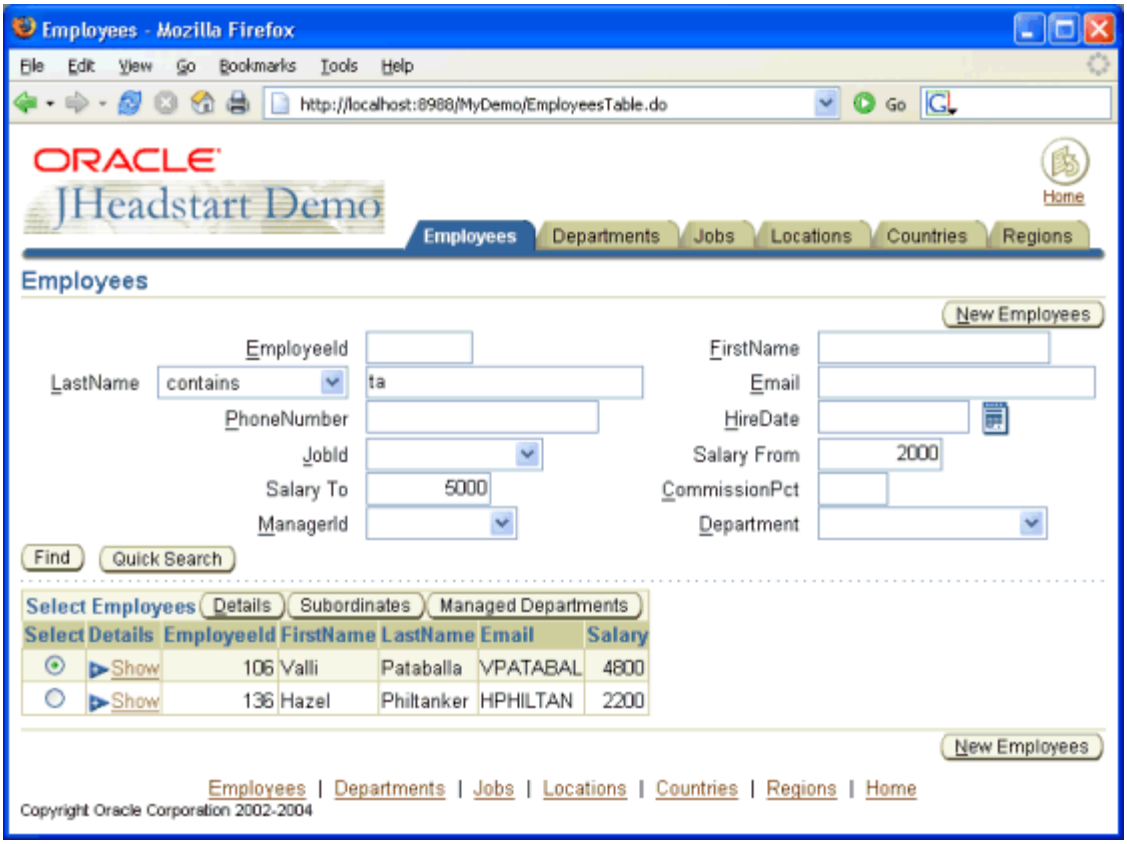


Figure 57: Advanced Search with Custom Criteria Treatment

- ***User Can Sort on Any Column in Departments Table***

  Click on the ***Departments*** tab, set the ***Filter By*** field to `DepartmentName` and enter a criteria of "`%es%`", then click ***(Go)***. You can now click on any of the column headings in the table, like the DepartmentName column for example, and sort the table by that column value as shown in Figure 58.

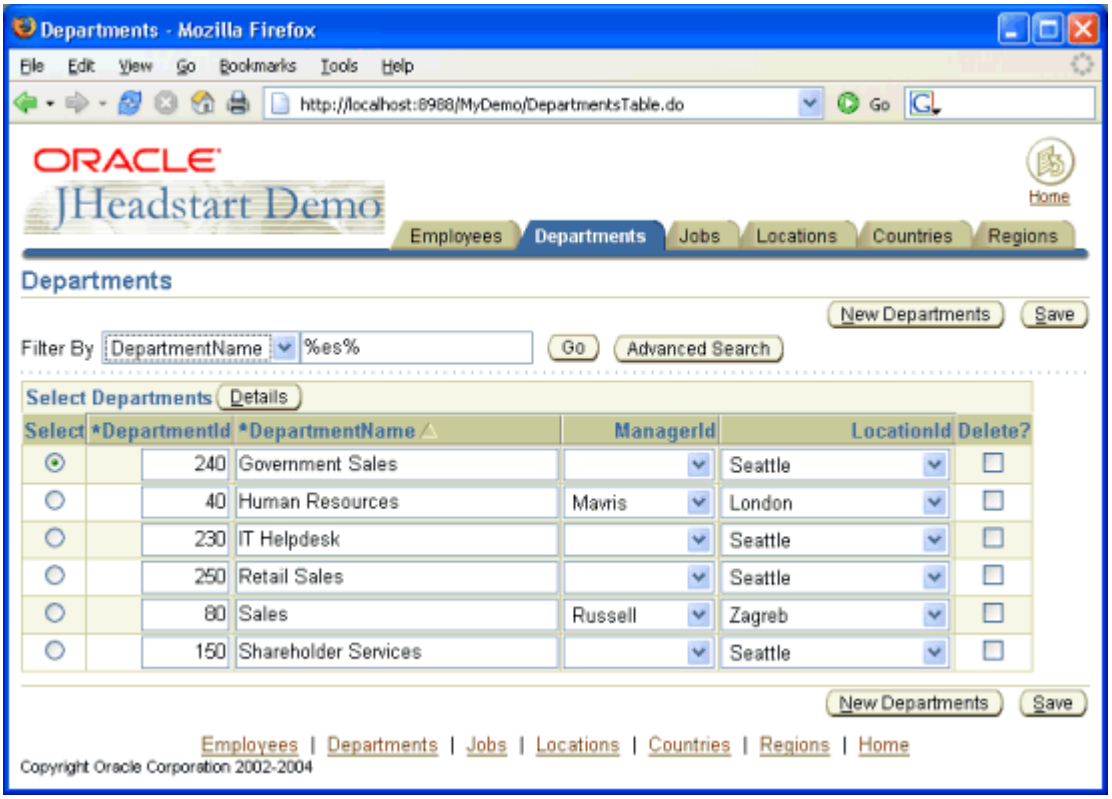


Figure 58: User Can Sort on Any Column in Departments Table

- ***User Works with Employees in Department on Same Page***

  Still on the ***Departments*** tab, select a department, then click on the ***Details*** button. As shown in Figure 59, this now shows the department details, along with the relevant employees details on the same page.

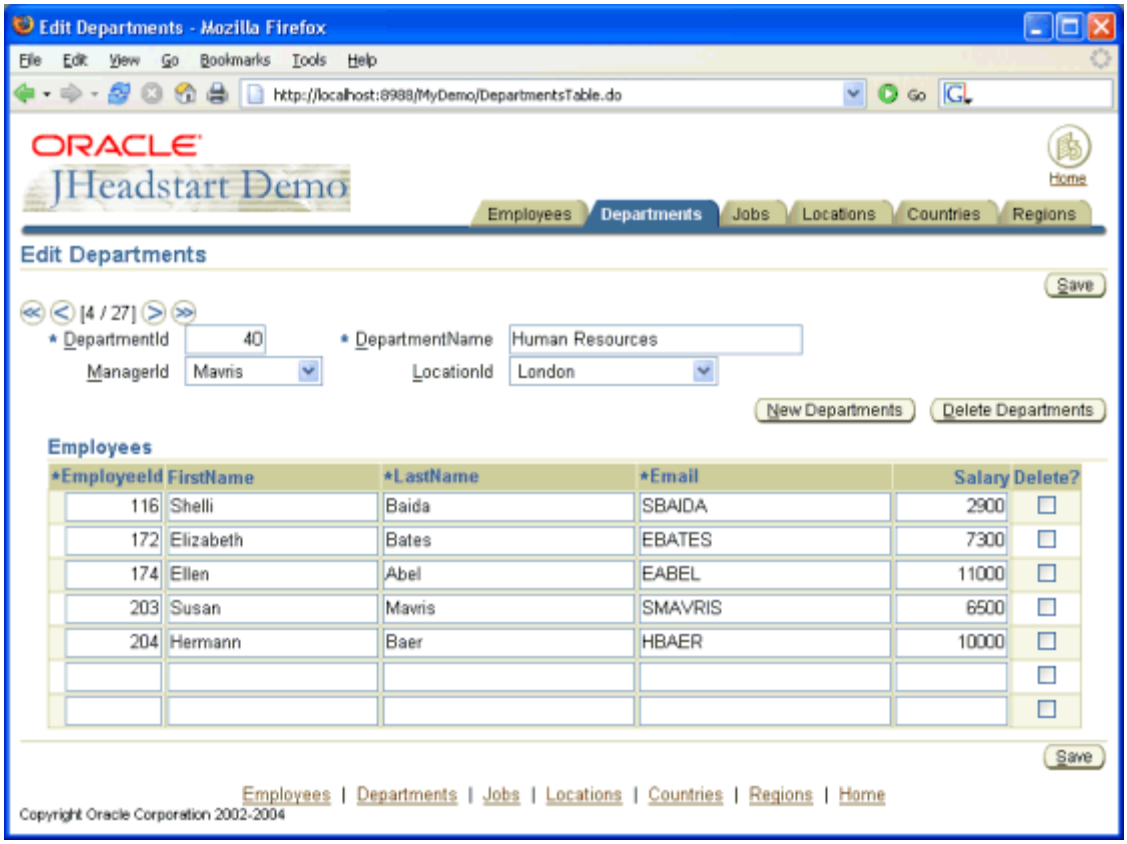


Figure 59: User Works with Employees in Department on Same Page

- ***User Selects Job to Edit from Simple List***

  Click on the ***Job*** tab and notice, as shown in Figure 60,that instead of the default table display to browse and select a Job to edit, the user now just selects the job name to edit from a simple list. Clicking on the ***(Edit)*** button brings you to the edit page for that job.

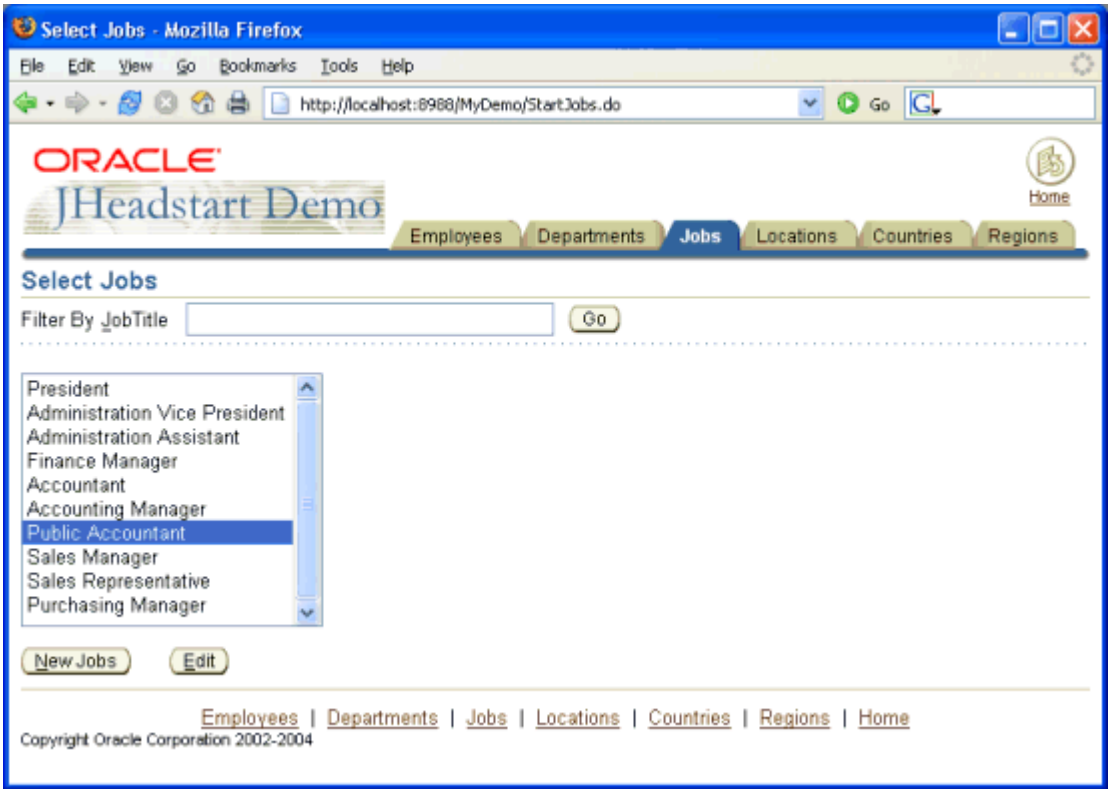


Figure 60: User Selects Job to Edit from Simple List

- ***User Navigates Five-Level Hierarchy Using Tree Display***

  Click the ***Regions*** tab, and you can drill down to find employees in departments in locations in countries in those regions. As shown in Figure 61, you can edit the data at any level.

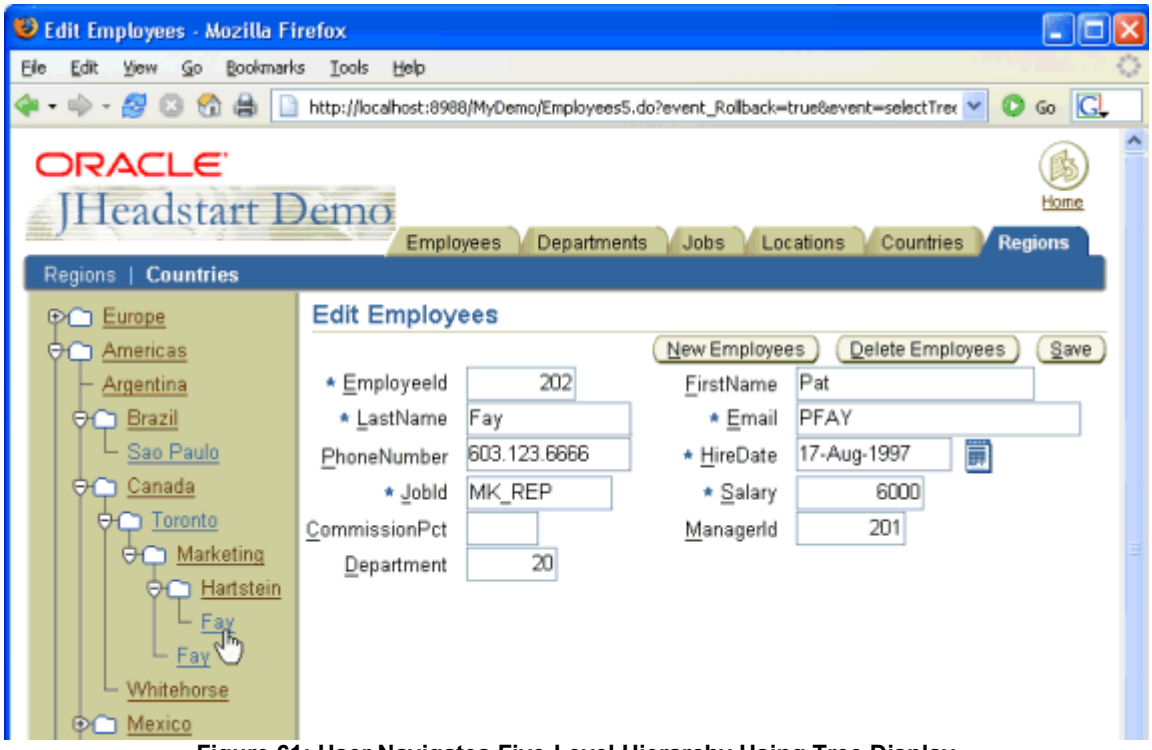


Figure 61: User Navigates Five-Level Hierarchy Using Tree Display

Step 3: Create Department Manager List of Values (LOV)

In this step, we are going to change the lookup definition for the manager of a department so that it behaves as a popup LOV window instead of as a dropdown list. When valid choices for a foreign key value are large in number, this kind of LOV window is more appropriate than a dropdown list. Specifically, we want to change the ***ManagerId*** dropdown list shown in Figure 62 to be a text field with a popup LOV instead.

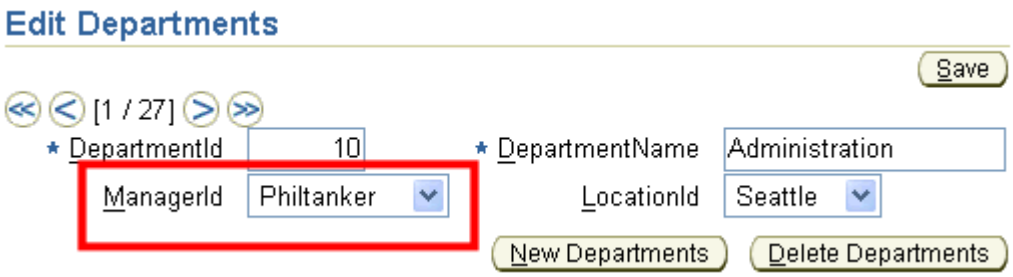


Figure 62: EmployeeList Lookup as Dropdown List for Setting ManagerId

If we were to choose to show the `ManagerId` as a text field, the value that the user will see would be the numeric id of the manager which is not exactly what we want. It would be a nicer for our end-users to show the manager's *last name* in the text field, and hide the numeric `ManagerId` value altogether. Then the user will see the last name, and popup the LOV to select last names from list. This will provide a lot better usability for our application. The following sections lead you through the few steps required with JHeadstart to accomplish this.

Step 3a: Add Manager Name to Departments Query

If we're going to show the name of a department's *manager*, we need to edit the definition of our `DepartmentsView` view object — back in our `Model` project — to include that bit of information.

1. ***Add Employees Entity Object to the DepartmentsView View Object***

   Select the `mydemo.model.DepartmentsView` view object in the application navigator, and double-click it to launch the ***View Object Editor***. On the ***Entity Objects*** panel of the editor, notice that the `Departments` entity object is already in use in this query. To add the `Employees` entity object as a second entity usage, as shown in Figure 63 select it in the ***Available*** list, and press ***(>)*** to shuttle it into the ***Selected*** list.

   Notice that by default the second entity object participating in this view object is marked as being ***Reference*** information and not updateable. While you can override this default setting, in this case the information we need to display from the `Employees` entity is read-only reference information — showing the department manager's last name — so we'll leave the default setting intact.

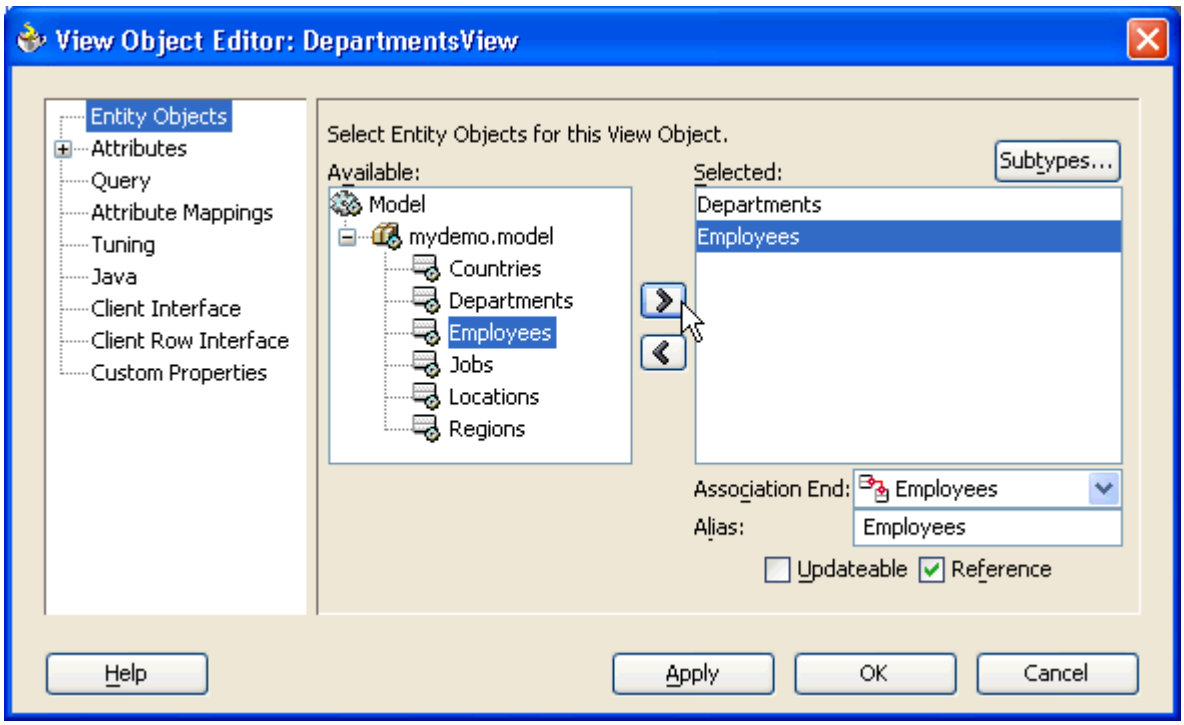


Figure 63: Adding Employees Entity to DepartmentsView

2. ***Add LastName Attribute to the Attributes List***

   On the ***Attributes*** panel of the editor, select the `Employees` entity's `LastName` attribute in the ***Available*** list, and press ***(>)*** to shuttle it into the ***Selected*** list as shown in Figure 64. Notice that the primary key attribute (`EmployeeId`) is also automatically added by the wizard.

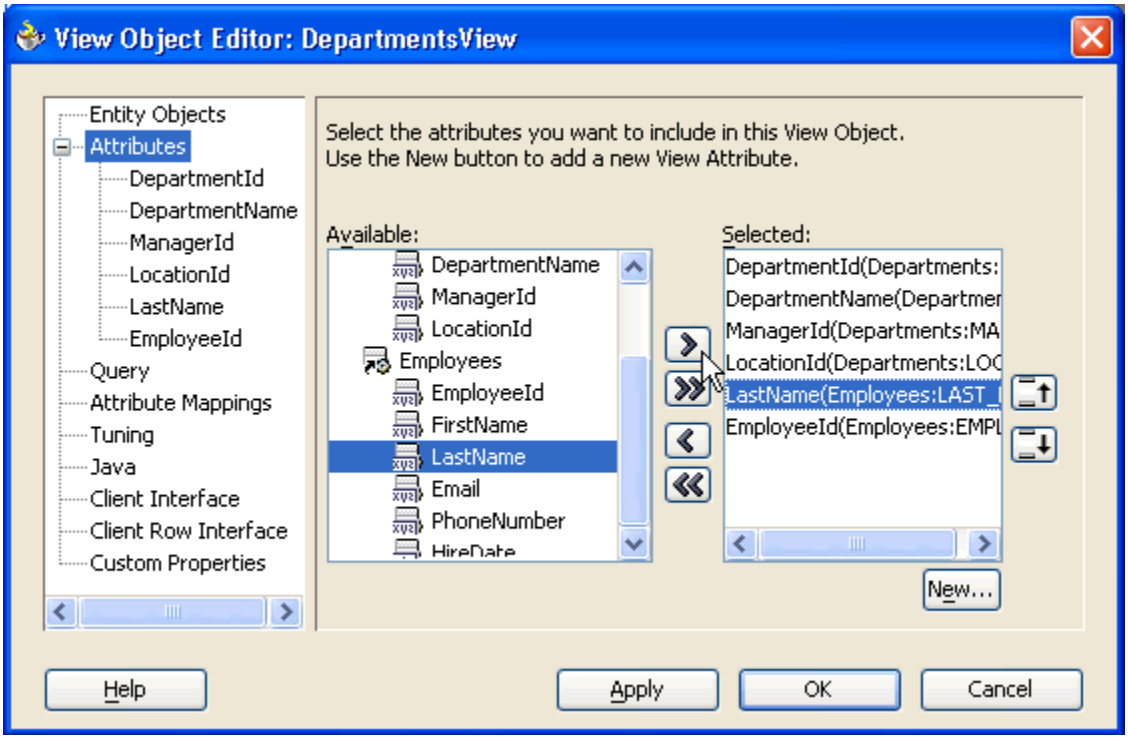


Figure 64: Adding the Employees.LastName Attribute to the List

3. ***Rename the LastName Attribute to ManagerName***

To make it clearer that this employee last name is the name of the department's *manager* in this particular view object, we can rename the attribute from `LastName` to `ManagerName`. To do this, select the ***LastName*** attribute name indented under the ***Attributes*** node in the tree at the left, and type the new name attribute name into the ***Name*** field as shown in Figure 65.

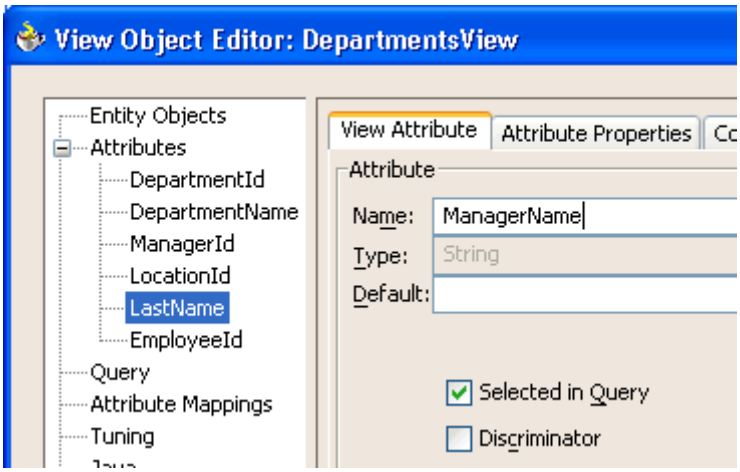


Figure 65: Renaming the LastName Attribute to ManagerName

4. ***Change SQL Query to Outer Join to EMPLOYEES Table***

Since departments are allowed to have no manager, their `ManagerId` foreign key attribute value might be `null`. To insure that we query all departments, whether or not they have a corresponding employee as their manager, we need to change the SQL query for this `DepartmentsView` to be an outer join.

To accomplish this, visit the ***Query*** panel of the editor, and type a "`(+)`" after the `EMPLOYEE_ID` column name the ***Where*** clause box as shown in Figure 66. Make sure there is a space between `EMPLOYEE_ID` and the "`(+)`" characters.

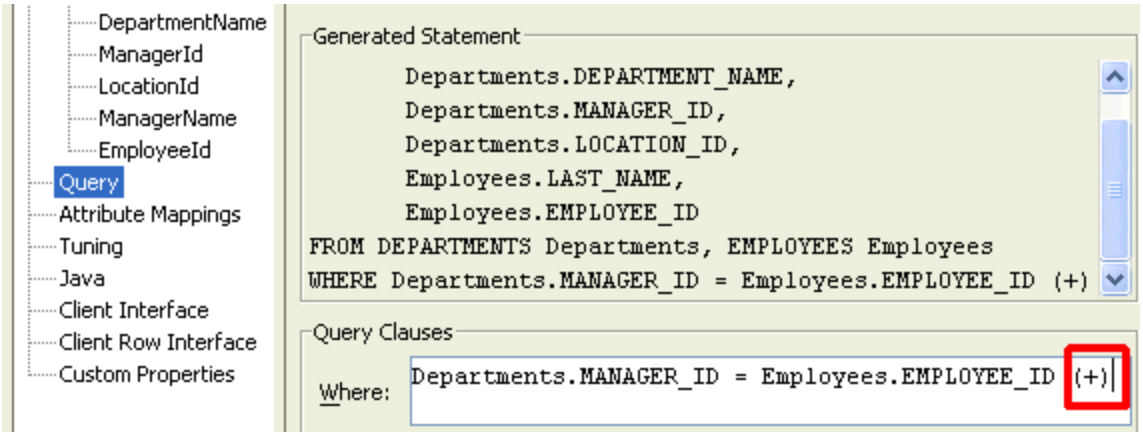


Figure 66: Changing the DepartmentsView to an Outer Join

Finally, click ***(OK)*** to accept all your changes to the `DepartmentsView` view object definition. When prompted to check your SQL statement's syntax, click ***(Yes)***.

Step 3b: Change the EmployeesLookup Definition to an LOV

With our view object modified to include the `ManagerName` attribute, we're ready to change the `EmployeesLookup` defintion to change the lookup display style from ***choice*** to ***LOV*** to get our list of values.

Back in the ***Application Structure File Editor***, select the `EmployeesLookup` that is a child of the `Departments` group as

shown in Figure 67. Then, make the following three changes:

1. Change the ***displayType*** to `lov`.
2. Change the ***Base Display Attribute*** to `ManagerName`.
3. Check the ***Use LOV for Validation?*** checkbox.

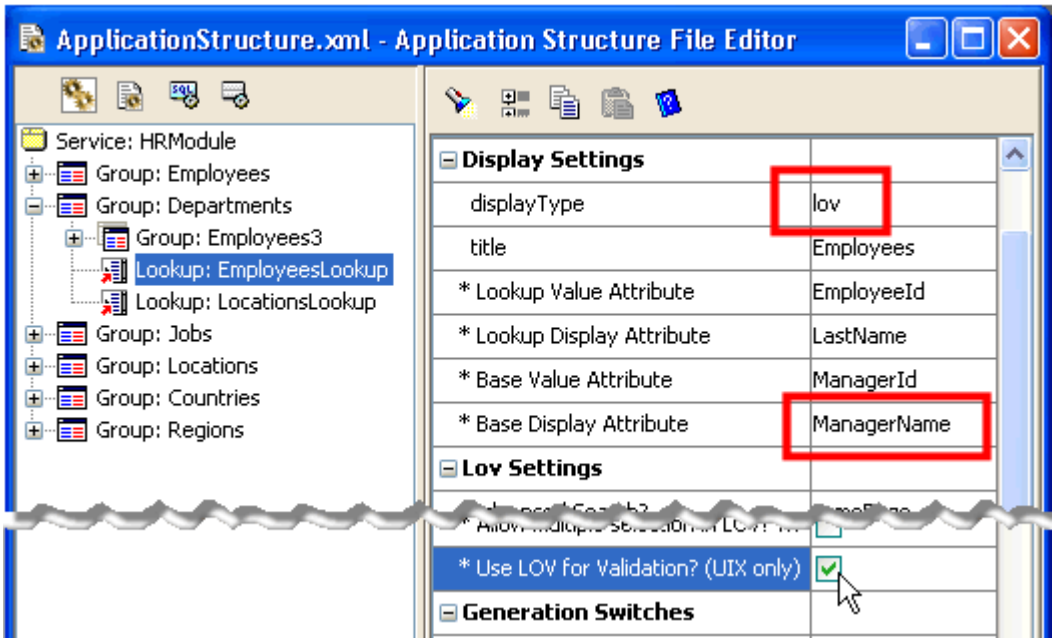


Figure 67: Changing the Lookup Style to LOV

In order to completely hide the numeric `ManagerId` and `EmployeeId` fields in the `DepartmentView`, we just need to set their ***Display*** property to false. We can do this easily by performing the following steps:

1. Select the top-level `Departments` group
2. Click the "SQL" icon in the ***Application Structure File Editor*** toolbar to launch the ***JHeadstart ADF BC Properties Editor*** for the related `DepartmentsView` view object.
3. Select both the `ManagerId` and `EmployeeId` attributes (by holding the `Ctrl` key while clicking) and set their ***Display*** property to `false` as shown in Figure 68.

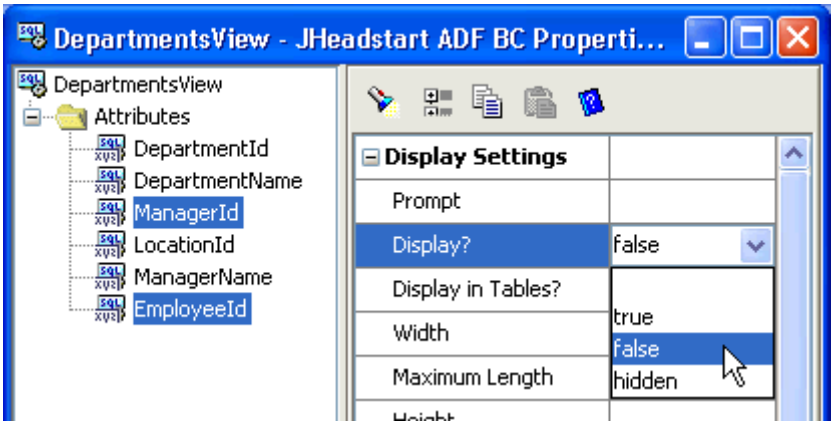


Figure 68: Hiding the Numeric Id Attribute in DepartmentsView

Step 3c: Regenerate and Run the Application

We're done setting up our LOV lookup field, so let's regenerate the application and run it. So far we've run the JHeadstart Application Generator from the right-mouse menu of the application structure file in the application navigator. As shown in Figure 69, you can perform the same task by clicking on the ***Run the JHeadstart Generator*** button in the ***Application Structure File Editor*** toolbar.

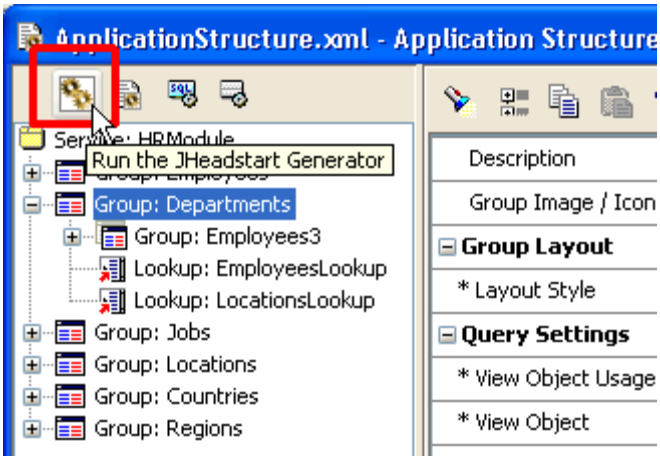


Figure 69: Running the JHeadstart Application Generator

When generation finishes, run the application again by clicking on the `ViewController` project in the application navigator and press `F11`.

Figure 70 shows what the ***Departments*** tab looks like after we've used the Quick Search to find only departments whose `DepartmentName` contains "`es`".

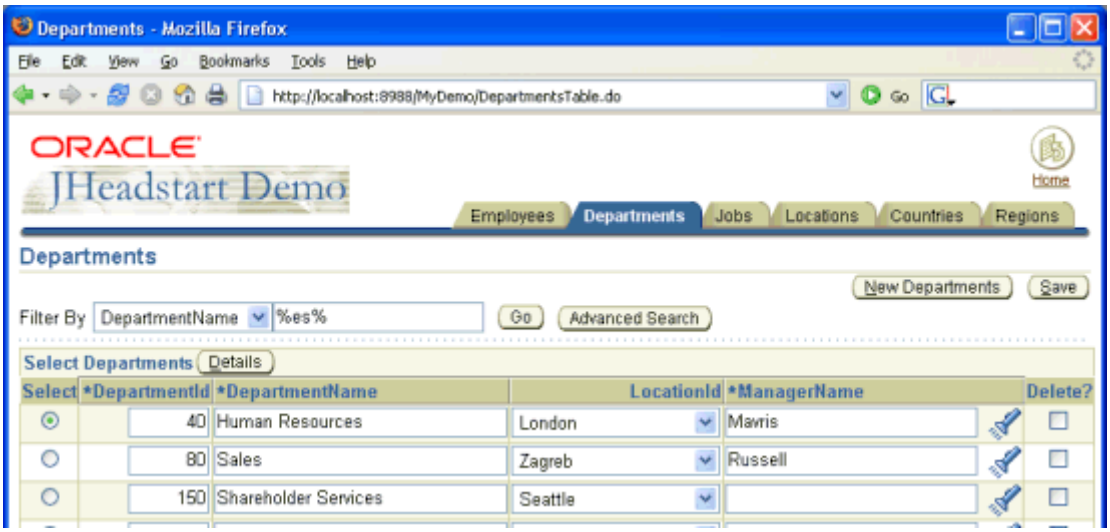


Figure 70: ManagerName LOV Lookup in Departments Browse Page

The changes we made to the `EmployeesLookup` lookup definition above show up in the browse page with a ***ManagerName*** LOV field in every row. We can see that our outer join is working correctly, since the "Shareholder Services" department has no manager and it was included in the query results.

If you try typing a letter "`p`" in the `ManagerName` field for the "Shareholder Service" department — or alternatively, changing one of the existing manager names in a different department to the letter "`p`" — and then pressing `Tab` to leave the field, you'll see the LOV window pop-up automatically showing the filtered list of choices that start with the letter "`p`" as shown in Figure 71.

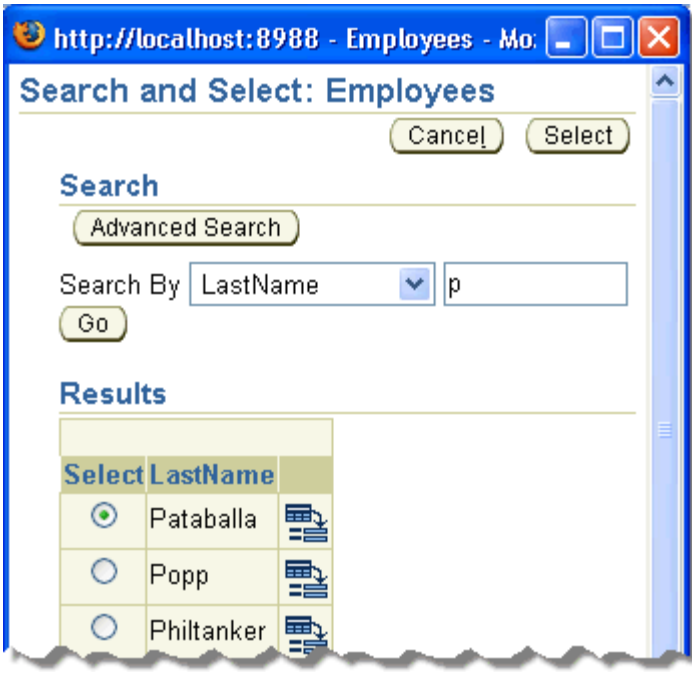


Figure 71: LOV Window Showing Filtered List of Matches

Back on the ***Departments*** browse page, if instead of typing *just* the letter "`p`" in one of the `ManagerName` fields, you type "ph" instead and `Tab` out of the field, you'll see another treat. Without bringing up the LOV window at all, the manager named Philtanker is automatically filled in. This is the "***Use LOV for Validation***" behavior we selected back in the JHeadstart lookup definition properties at work at runtime. Of course, you can also just click on the "searchlight" icon and pop-up the LOV window for you to filter and select the choice yourself.

NOTE: Although a roundtrip from the browser to the application server is made to check the number of matching rows, only the `ManagerName` field is actually refreshed on the page. This is accomplished through an ADF UIX feature called Partial Page Rendering (PPR) that we will expore in more detail in Step 6: Add a Conditionally-Dependent Field .

If we drill-down to edit the details for a department, as shown in Figure 72 the same `EmployeesLookup` changes we made above result in showing a `ManagerName` LOV field in the ***Edit Departments*** form as well.

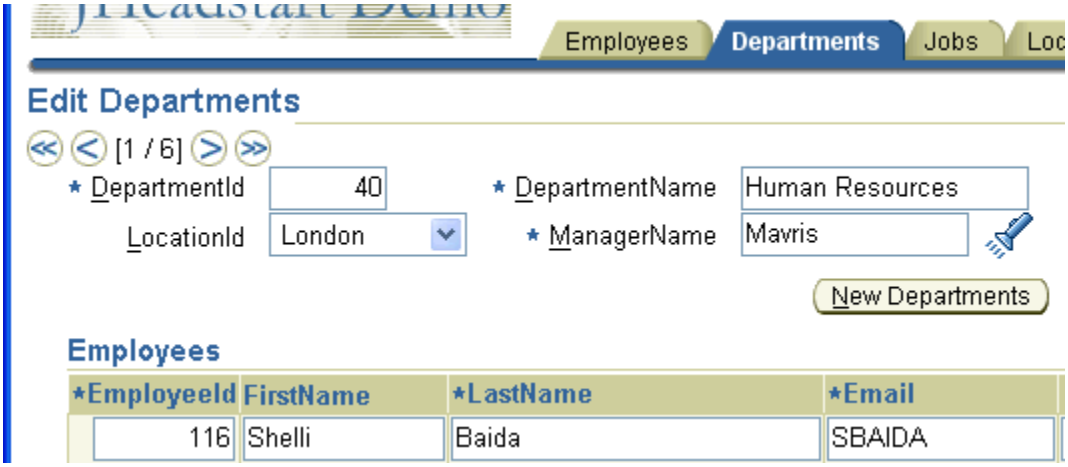


Figure 72: EmployeeLookup LOV Also Available in Edit Forms

Step 4: Shuttle Employees Between Departments

In this step, we will generate a so-called “parent-shuttle” control to allow the user to easily move employees from one department to another by selecting them from an ***Available*** list and shuttling them into a ***Selected*** list for the current

department.

TIP: The `UPDATE_JOB_HISTORY` trigger on the `JOB_HISTORY` table in the HR sample schema does not allow multiple department changes to an employee on the same day. You may need to disable or drop that trigger to avoid runtime errors while trying this part of the demo. You can drop the trigger from inside JDeveloper by:

- Select ***View | Connection Navigator*** from the main menu
- Expand the ***Database*** category
- Expand the `hr` connection definition
- Expand the ***Triggers*** category under that
- Select the `UPDATE_JOB_HISTORY` trigger
- Pick ***Drop*** from the right-mouse menu.

Step 4a: Create View Object to Query Available Employees

To be able to generate the shuttle, we first need to create an ADF view object that queries all employees who do *not* belong to the current department. We'll supply *which* department we mean by including a bind variable in the query and setting a property in the JHeadstart application structure to set its value to the current department automatically. This will provide the data to display in the ***Available*** list in the shuttle for the current department.

1. ***Create a New View Object***

   In your `Model` project in application navigator, select the `mydemo.model` package (under the ***Application Sources*** folder) and pick ***New View Object...*** from the right-mouse menu. If the wizard's welcome page appears, click ***(Next>)***.

2. ***Give the New View Object a Meaningful Name***

   On the ***Step 1 of 6: Name*** page of the wizard, set the name of the new view object to `EmpsNotInDept`. As shown in Figure 73, leave the defaults selected for the ***What kind of data do you need this View Object to manage?*** radio group.

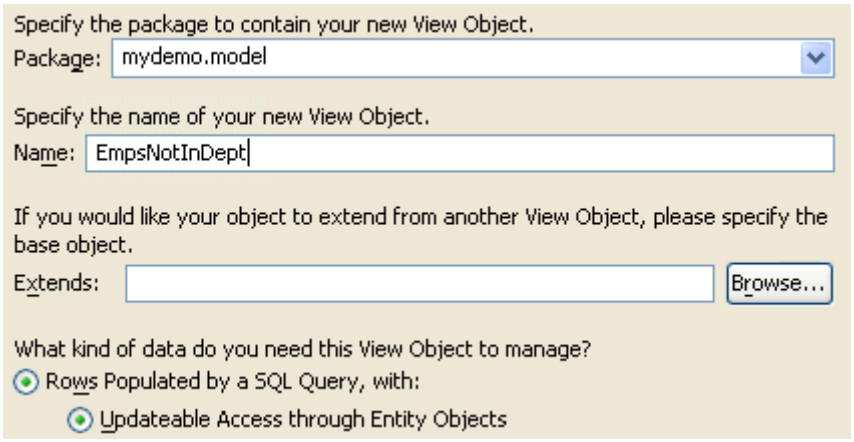


Figure 73: Assigning a Name to Our New View Object

3. ***Select the Employees Entity Object to Use in the Query***

   On the ***Step 2 of 6: Entity Objects*** panel of the wizard, as shown in Figure 74 select the `Employees` entity object in the ***Available*** list and press ***(>)*** to shuttle it into the ***Selected*** list.

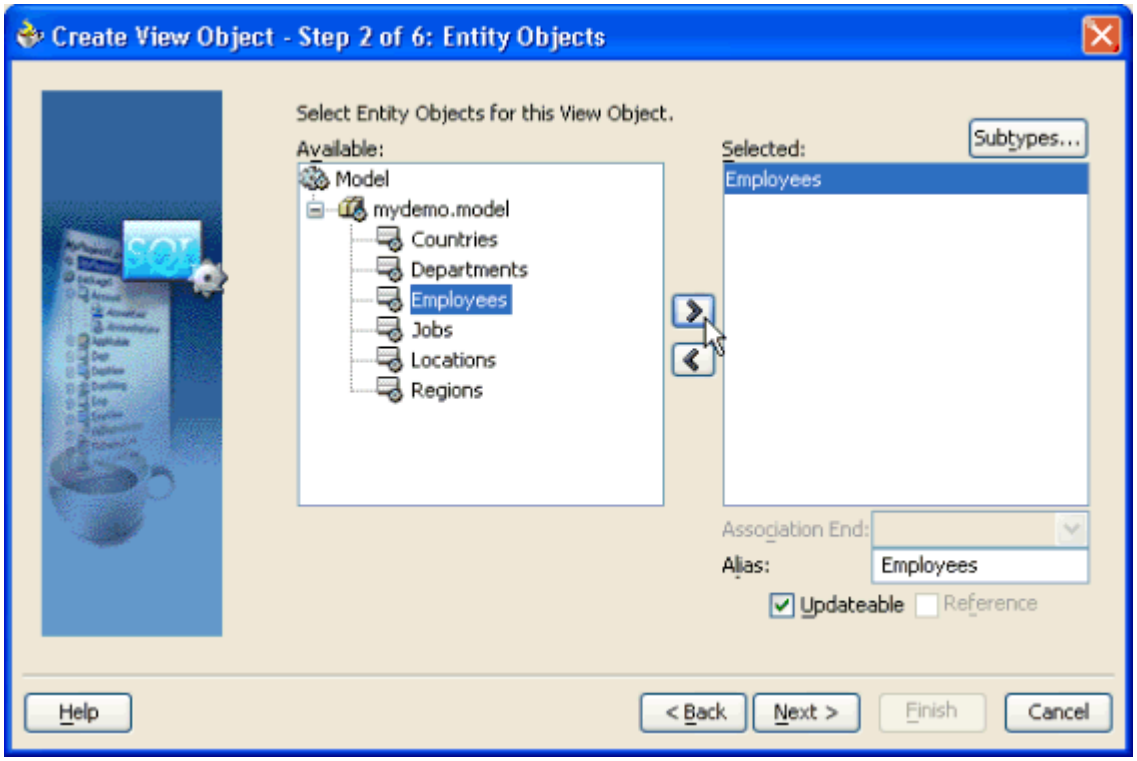


Figure 74: Selecting the Employees Entity Object for New View Object

4. ***Include the Necessary Attributes in the Query***

   On the ***Step 3 of 6: Attributes*** panel of the wizard, as shown in Figure 75 hold the `Ctrl` key down while you click the `EmployeeId`, `LastName`, and `DepartmentId` attributes, then press ***(>)*** to shuttle them into the ***Selected*** list.

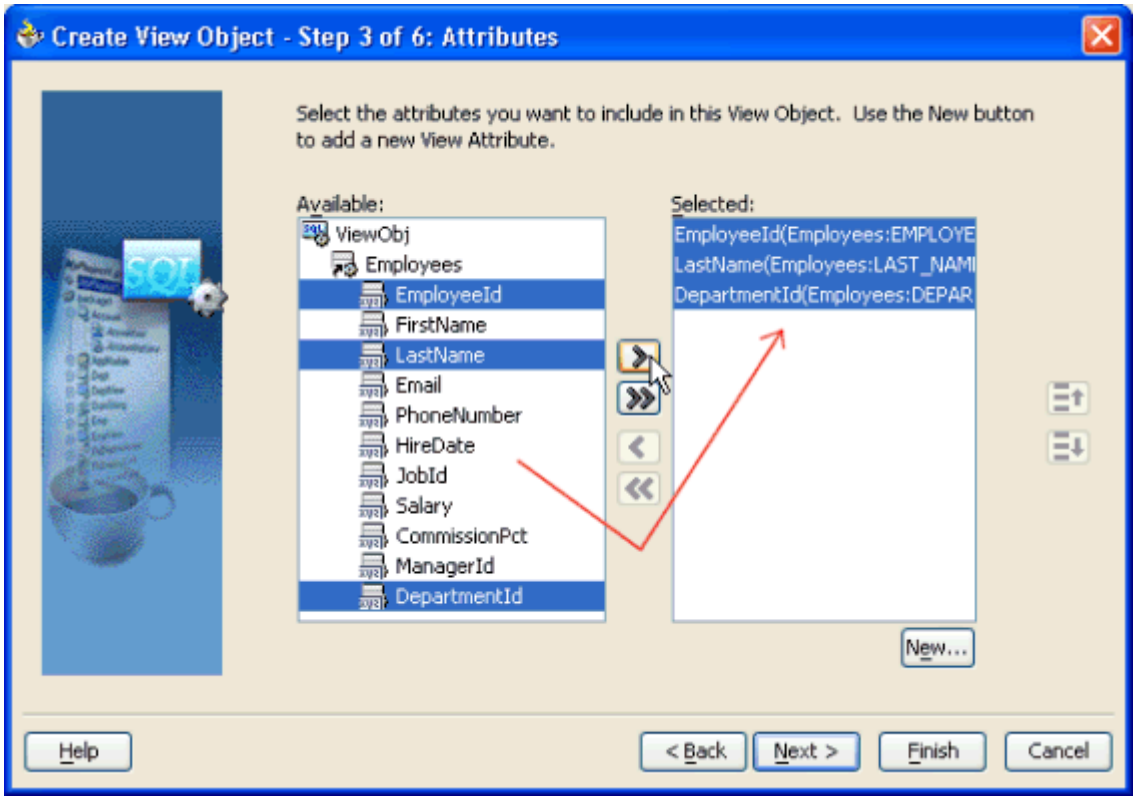


Figure 75: Picking the Attribute We Need in the Query

5. ***Skip the Attribute Settings Page***

   On the ***Step 4 of 6: Attribute Settings*** panel, just click ***(Next>)*** to get to the next panel. We don't have any attribute settings that we need to change.

6. ***Set a Custom Where Clause to to Filter Employees***

   On the ***Step 5 of 6: Query*** panel of the wizard, as shown in Figure 76 specify the following custom ***Where*** clause that will include only employees with a NULL department_id value, or employees that are not in the given department.

   ```
   (department_id is null or department_id != :1)
   ```

   The `:1` represents a SQL bind variable whose value we'll be assigning at runtime via an expression that represents the current deparment number value. We'll set up that aspect in just a second...

   Click ***(Finish)*** to end the ***Create View Object*** wizard, answer ***(Yes)*** when prompted to check the SQL statement for correctness, and click ***(OK)*** in the alert that confirms the query is valid.

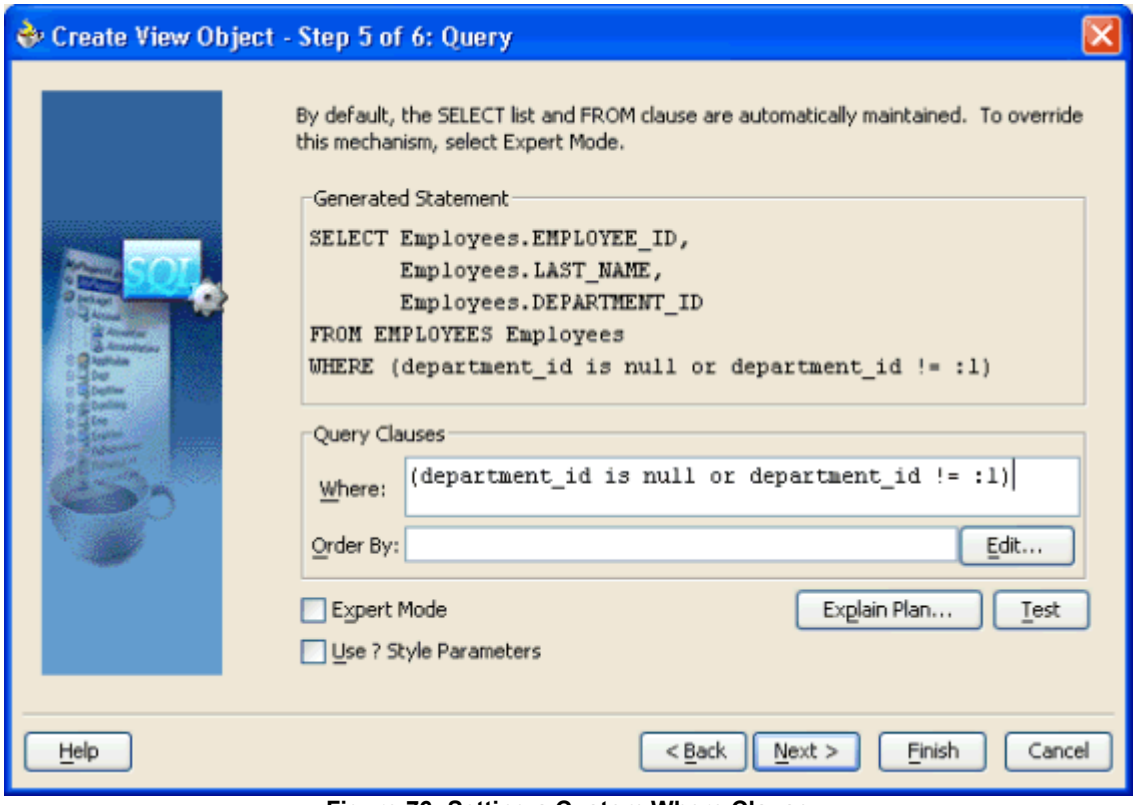


Figure 76: Setting a Custom Where Clause

7. ***Include Instance of EmpsNotInDept View Object in HRModule***

   For clients to work with the data this new view object provides, we need to include an instance of it in the data model of our `HRModule` application module component. To do this:

   1. Select the `HRModule` component in the application navigator, and double-click to edit it.
   2. On the ***Data Model*** panel of the editor, select the `EmpsNotInDept` view object in the ***Available View Objects*** list.
   3. Press ***(>)*** to shuttle an instance of this view object, which by default will be named `EmpsNotInDept1`, into the application module's data model.

Then click ***(OK)*** to save the changes to the `HRModule`. Finally, select ***File | Save All*** from the main menu (or click the "Save All" toolbar button), and we're ready to go modify the JHeadstart application structure to use this new view object as part of a shuttle.

Step 4b: Setting Up the Parent Shuttle

Back in the ***JHeadstart Application Structure File Editor***, we need to perform two setup steps for the shuttle: one for the `Employees3` group, and one for its nested `EmployeesLookup`.

- ***Setup the Employees3 Group as a Parent-Shuttle***

  Select the `Employees3` group that is a child of the top-level `Departments` group and as shown in Figure 77, perform the following four steps:

  1. Set the ***Layout Style*** to `parent-shuttle`
  2. Set the ***Tab Name*** to "Assign Employees to Department"
  3. Set the ***Display Title (Plural)*** to "Assigned"
  4. Uncheck the ***Use Table Range*** checkbox

     This way we'll see all employees assigned to the current department in the list instead of only the first pageful.

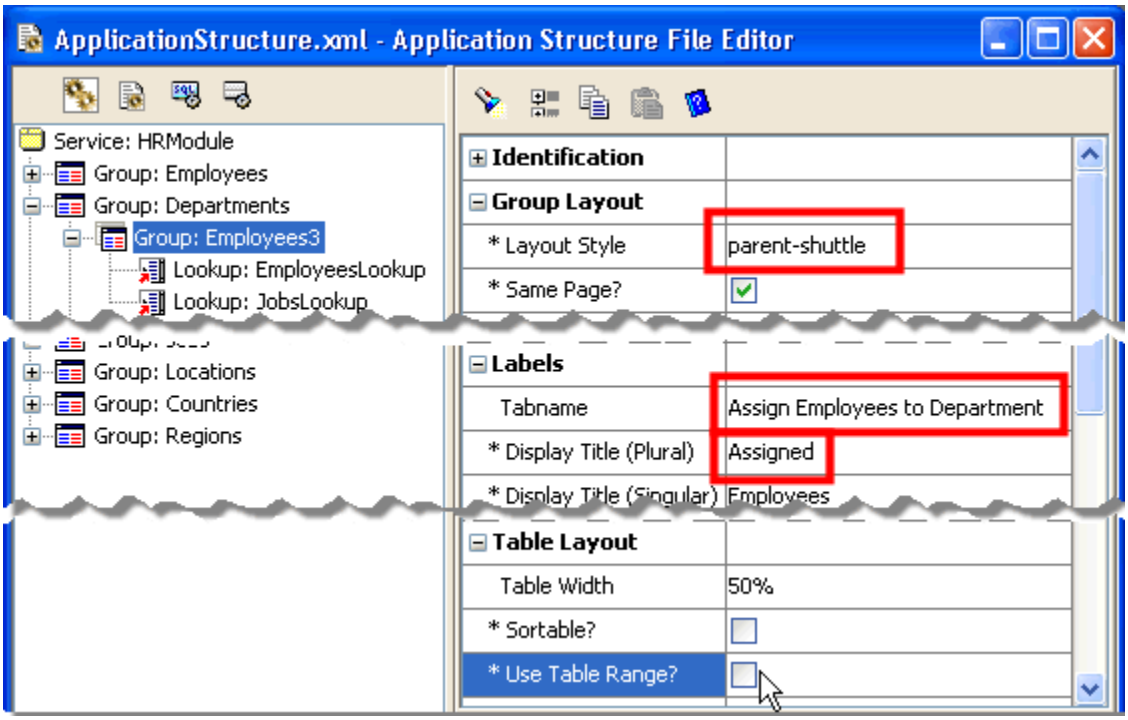


Figure 77: Setting Employees3 Detail Group to be Parent-Shuttle

- ***Setup the Nested EmployeesLookup for the Shuttle***

  Select the `EmployeesLookup` that is *indented* under the `Employees3` child group — *not* the one that's at the same level as Employees3 — and as shown in Figure 78, perform the following five steps:

1. Set the ***View Object Usage*** to `EmpsNotInDept1`
2. Set ***Query Bind Parameters*** to the expression:

   ```
   ${bindings.DepartmentsDepartmentId}
   ```

   This is the standard J2EE expression language syntax that declaratively represents the value of the current `DepartmentId` on the ***Departments*** page at runtime.

3. Set the ***title*** property to "Unassigned"
4. Set ***Advanced Search?*** to `none`
5. Set ***Quick Search?*** to `none`

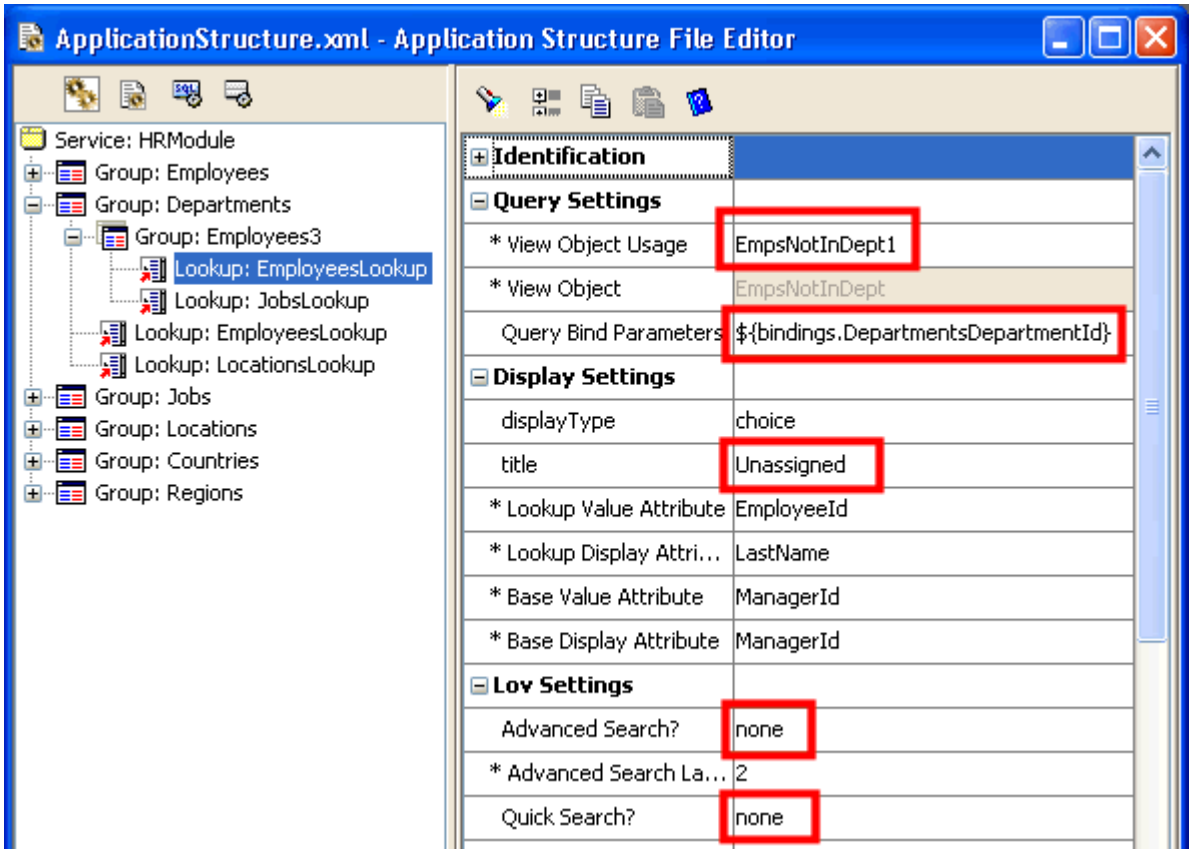


Figure 78: Setting Up the EmployeesLookup for the Shuttle

NOTE: While here we are using a "Parent Shuttle", JHeadstart's also supports generating an "Intersection Shuttle" to make handling many-to-many data entry easier. For example, given a student and a list of available courses, an intersection shuttle would allow you to select the courses the current student wants to attend, saving the data in an intersection table between Students and Courses. The JHeadstart Developer's Guide [2] describes the intersection shuttle and how to use it in detail.

Step 4c: Generate and Run the Application

Click on the ***Run the JHeadstart Generator*** button in the ***Application Structure File Editor*** toolbar to get JHeadstart to regenerate the web tier of the application now that we've completed the shuttle setup steps. When generation finishes, run the application again by clicking on the `ViewController` project in the application navigator and press `F11`.

At runtime, to see the shuttle control shown in Figure 79, do the following:

- Click on the top-level ***Departments*** tab
- Select a department by clicking in the ***Select*** radio group next to the desired one
- Click the ***(Details)*** button to drill-down to the ***Edit Departments*** page.

On the Edit Departments page, you can navigate back and forth between departments, and for each department the shuttle will show you the ***Unassigned*** and ***Assigned*** employees in that department. You can shuttle employees into or out of the department using the shuttle. Clicking the ***(Save)*** button will save your changes permanently.

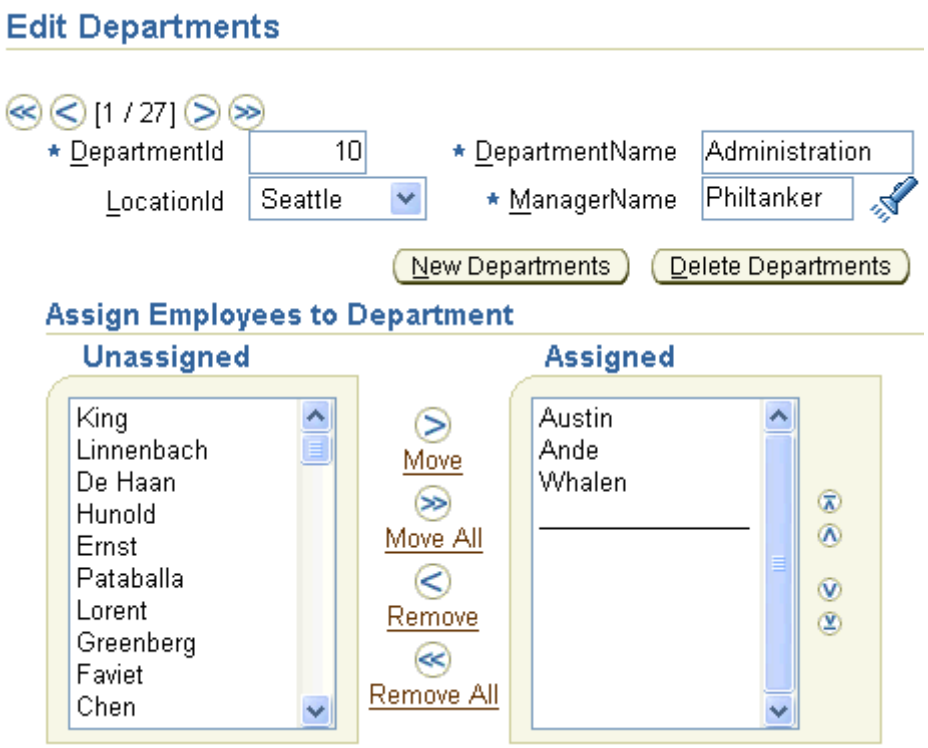


Figure 79: Parent Shuttle Control at Runtime

Step 5: Customize Generation Templates

In this step, we will use some customized JHeadstart Application Generator templates to illustrate the power of customization through generator templates. JHeadstart ships with some sample templates to illustrate some of the tokens you can use in these templates and the documentation covers all of the available tokens. The sample templates are called:

- `dataPageTabbedChildren.jut`
- `childGroupTabs.jut`
- `tabbedTableChildGroup.jut`

Together, these sample templates implement the generation of tabbed child groups. In other words, with these templates in effect, all child groups displayed on the same page as the parent group will show on in-page tabs, with only one child group visible at a time.

To indicate a set of templates to use during generation, you create a JHeadstart custom template properties file (`*.jtp`) that indicates which templates should be used for which of the standard page types. To use the sample templates above and generate tabbed child groups, you can use the sample custom template property file named `JhsTabbedChildrenTemplates.jtp` that JHeadstart will copy into the `./properties` subdirectory of your project when you use the ***Create New JHeadstart Application Structure File*** wizard for that project.

If you're curious to see the contents of a `*.jtp` file, find the `JhsTabbedChildrenTemplates.jtp` file under the

NOTE: ***Miscellaneous Files*** folder in the application navigator and double-click on it to see it in the code editor. It's a simple text file that pairs token names with individual template files.

Let's put these custom templates to work in order to generate a page for the `Employees` group that has its two child groups — `Employees2` and `Departments2` — appearing on the same page as their parent and organized into tabs.

Step 5a: Indicate Where Custom Templates Should Be Used

- ***Set Custom Template File to Use for Employees Group***

  In the ***JHeadstart Application Structure File Editor***, select the top-level `Employees` group, and find the ***Template Properties File*** property in the ***Generation Settings*** category. As shown in Figure 80 — see the tooltip for the full file name — set the property to the fully-qualfied path to the `JhsTabbedChildrenTemplates.jtp` file. The easiest way to do this is to click into the ***Template Properties File*** field, and press the ***(...)*** button to pick the file using a file browser.

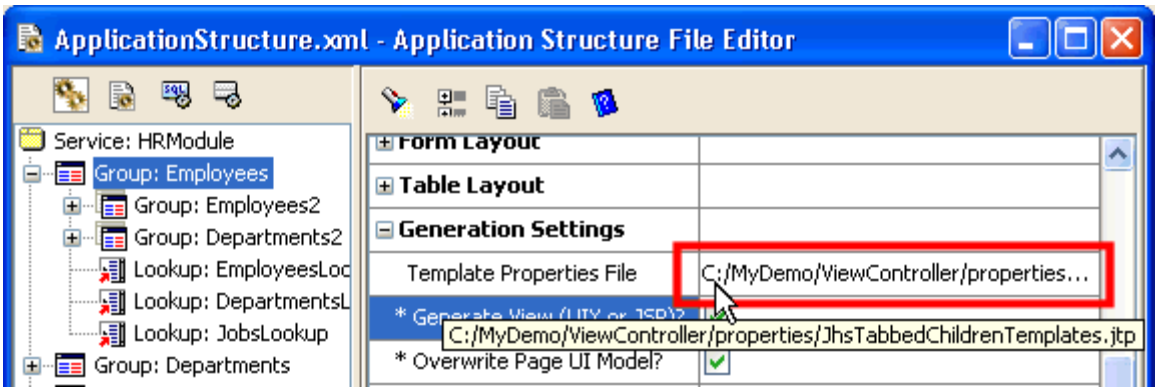


Figure 80: Setting Custom Template File for Employees Group

- ***Make Child Groups of Employees Appear on Same Page***

  Select the `Employees2` child group under the top-level `Employees` group, and check its ***Same Page?*** checkbox (in the Group Layout category).

  Repeat the same setting for the `Departments2` child group at the same level under `Employees`.

Step 5b: Generate and Run the Application

Click on the ***Run the JHeadstart Generator*** button in the ***Application Structure File Editor*** toolbar to get JHeadstart to regenerate the web tier of the application now that we've completed the custom template setup steps. When generation finishes, run the application again by clicking on the `ViewController` project in the application navigator and press `F11`.

At runtime, to see the effect of the custom template as shown in Figure 81, do the following:

- On the ***Employees*** tab, set the ***Filter By*** quick search field to `LastName`, enter a last-name search criteria of "`part`", and click ***(Go)*** to find ***Karen Partner***.
- Click the ***(Details)*** button to drill-down to the Edit Employees page
- Click on the ***Managed Departments*** or the ***Subordinate*** tabs in the page to switch between the different detail information for the current employee.

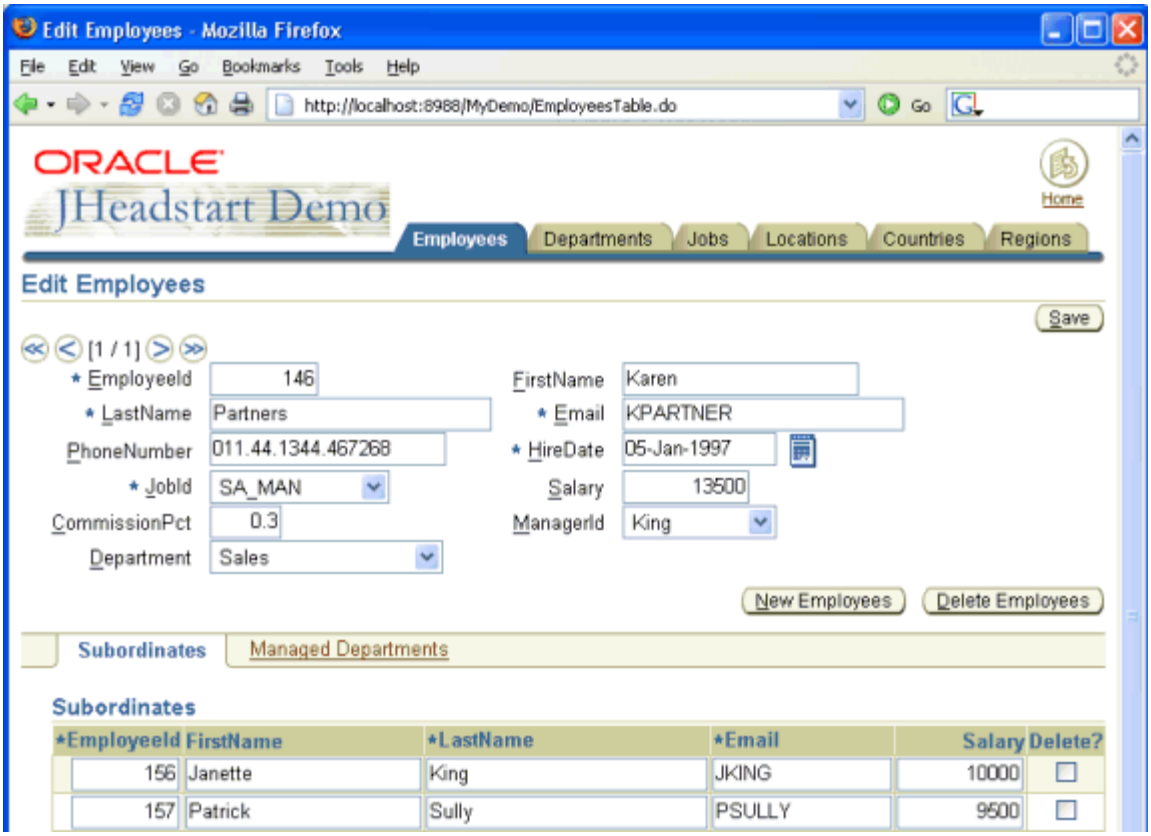


Figure 81: Custom Templates Produce Same-Page Details Tabs

Step 6: Add a Conditionally-Dependent Field

In this step, we will consciously *not* use JHeadstart. This step illustrates how you can enhance your generated application using the visual ADF design time facilities, adding features that you currently cannot generate. As a post-generation feature, we will implement a conditionally dependent field on the page. When we're done, to preserve our customization we'll set a property in the application structure to indicate that the JHeadstart application generator should no longer regenerate this page.

For example, let's assume that an employee's commission percentage can only be entered when the employee is an Account Manager (`JobId` = "`AC_MGR`"). We want the `CommissionPct` field to be disabled for employees with other values for their `JobId` field. So let's get to work on making it happen...

Step 6a: Add Expression to Field's Disabled Property

- ***Open Employees.uix Page in the Visual Designer***

  In your `ViewController` project, expand the ***Web Content*** folder and the `WEB-INF/page` subdirectory. Find `Employees.uix` in the list of pages, and double-click it to open it in the visual designer.

- ***Bind Disabled Property of CommissionPct Field***

  In the visual page designer, select the `CommissionPct` field and find the field's ***disabled*** property in the Property Inspector. Click into the row of the property table for the ***disabled*** property to select it.

  TIP: You can do ***View | Property Inspector*** from the main menu if it's not currently visible.

  Notice the current value of ***disabled*** is `false`. To make the ***disabled*** property's value dynamic — based on evaluating an expression at runtime — click the "***Bind to Data***" toolbar icon as shown in Figure 82.

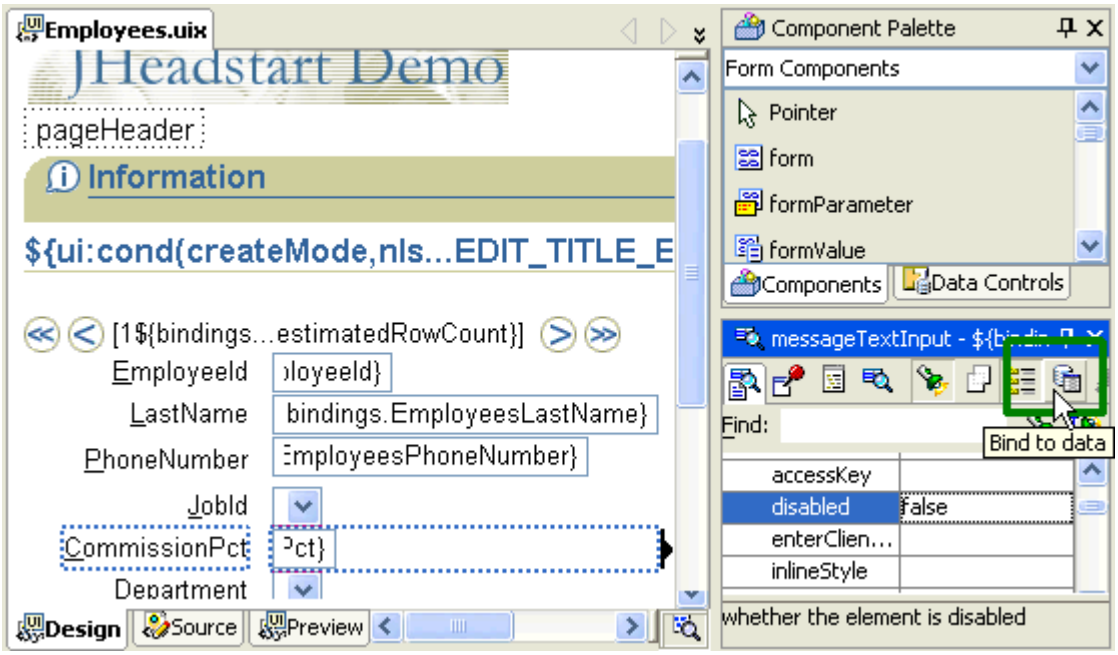


Figure 82: Binding Disabled Property of CommissionPct Field

When you do this, a small "database-can" icon appears next to the name of the ***disabled*** property in the inspector to indicate that its value is bound to an expression. Additionally, the value of the ***disabled*** property will change from a fixed `false` setting to an equivalent expression of `${'false'}`.

- ***Enter Expression Determining When Commission is Disabled***

1. Click into the area showing the ***disabled*** property's value of `${'false'}` and notice that a ***(...)*** button appears.

2. Click this ***(...)*** button and a helpful binding picker dialog appears to assist in formulating the expression.

3. As shown in Figure 83, expand the ***bindings*** node to find the element representing the `EmployeesJobId` data item, and select it.

   This will update the ***Expression*** field to reflect the expression for the current employee's `JobId` value of `${bindings.EmployeesJobId}`.

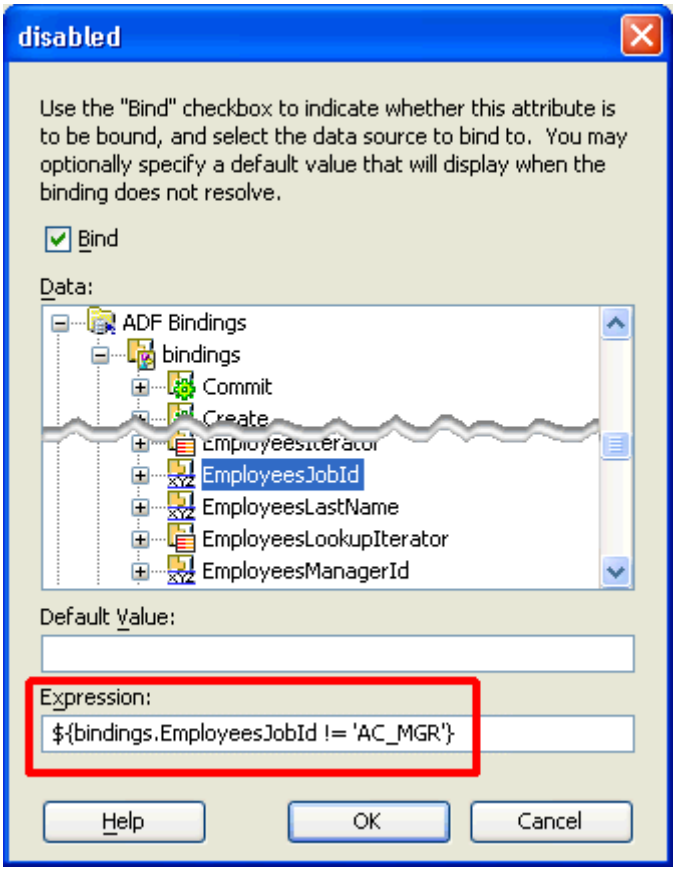


Figure 83: Using the Binding Picker to Help Create Expression

4. Edit the ***Expression*** to add the not-equals comparison to the literal value `'AC_MGR'` as shown in Figure 83.

   This is an expression that returns a boolean `true` when the value of the current employee's `JobId` does not equal `AC_MGR`. Since this is the expression for the value of the ***disabled*** property, that means at runtime we'll effectively have ***disabled*** = true when the `JobId` value is different from `AC_MGR`, and ***disabled*** = false (in other words, *enabled*) when `JobId` equals `AC_MGR`.

5. Click ***(OK)*** to save the expression

Finally, do a ***File | Save All*** in the JDeveloper main menu, and we're ready to try out the changes.

If your application is still running on the JDeveloper embedded container, then you can just continue to work with it to verify the new functionality we've just added. If it's not running, just click on your ViewController project and press `F11` to get it running again.

Try these steps:

- Click on the ***Employees*** top-level tab
- Click on the ***(Advanced Search)*** button
- Select "Accounting" from the ***Department*** dropdown list
- Click ***(Find)*** to perform the search
- Click ***(Details)*** to drill-down to show employee details
- Try scrolling back and forth between employees with ***(<)*** and ***(>)*** buttons

Notice that on an employee like Shelly Higgins, who's `JobId` is `AC_MGR`, the `CommissionPct` field is enabled. On employees who have a different `JobId`, the `CommissionPct` is disabled. Excellent!

However, we're still not completely done.

If you update an employee's `JobId` to be `AC_MGR` when it currently was *not* that value, the `CommissionPct` field does not correctly enable immediately as we (and the end user!) might expect. If we navigate away to a different employee and back to the one we changed, the `CommissionPct` field is correctly enabled, but it would nice to have it get enabled instantly, when appropriate. So let's make it so!

Step 6b: Make Field React to Live Data Changes

The ADF UIX technology that the JHeadstart Application Generator uses by default for your web tier pages cleverly combines Asynchronous JavaScript, XML, and Dynamic HTML to deliver a much more interactive web client interface for your business applications. In ADF UIX, the feature is known as partial page rendering because it allows selective parts of a page to be re-*rendered* to reflect server-side updates to data, without having to refresh and redraw the entire page. This combination of web technologies for delivering more interactive clients is gaining momentum in 2005 under the newly-coined acronym AJAX [10]. ADF UIX has supported it for several years already, and luckily, leveraging its power requires just a few declarative property changes in your page.

NOTE: JHeadstart also supports generating a JSP-based web tier instead of ADF UIX, but the JSP pages don't support partial page rendering or any of the more advanced UI components and features that ADF UIX offers.

We'll leverage the ADF UIX partial page rendering technology to cause the `CommisionPct` field to enable or disable dynamically in response to changes made in the current page to the JobId dropdown list. Here's what you need to do...

Go back to the visual designer for the `Employees.uix` page and select the `JobId` dropdown field. In the property inspector, select the ***PrimaryClientAction*** property, then click the ***(...)*** button to launch the ***primaryClientAction*** editor dialog.

NOTE: Developers familiar with our Oracle Forms tool might recognize the `primaryClientAction` event for a dropdown list in ADF UIX to be similar to a `WHEN-LIST-CHANGED` trigger in Forms.

Fill out the dialog as shown in Figure 84. The name of the ActionEvent is arbitrary, so we just pick a name that makes sense like `whenJobChanged`. We configure the action to submit the form, but not to validate the form on submission. Finally, we enable the firing of partial page rendering targets and select the id in the list corresponding to the `CommissionPct` field.

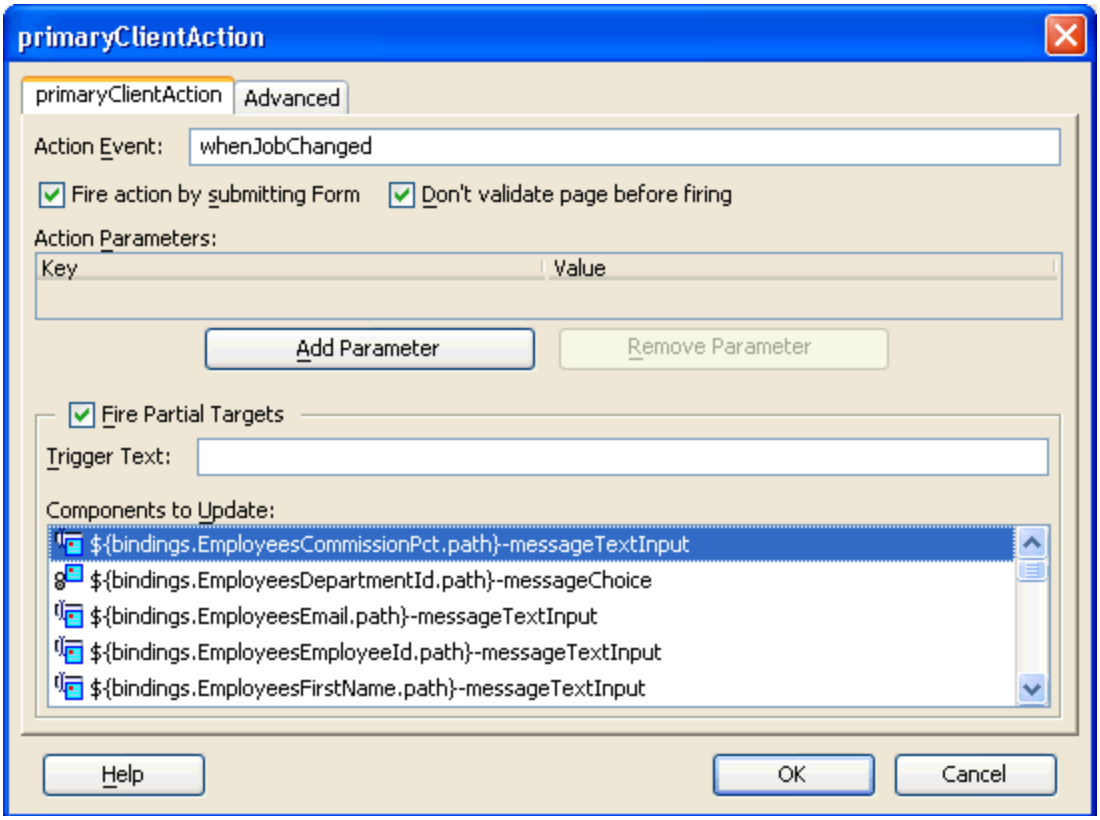


Figure 84: Configuring JobId Dropdown List to Update CommissionPct

Partial targets are components on the page that will be refreshed without needing to refresh the entire page. In this case whenver the `JobId` is changed, we're setting up the action to update the `CommissionPct` field in reaction to that change as well.

Save the page, then go back to the still-running application to see the effect of our last change. First, browse to a different employee to activate the new version of the `Employees.uix` page we just saved. Once you've done that, you'll now notice that changing an employee's `JobId` to `AC_MGR` immediately enables the `CommissionPct` field.

Step 6d: Protect the Customized Page From Regeneration

To preserve this post-generation change that we've made manually to the `Employees.uix` page using JDeveloper's ADF design-time tools, select the `Employees` group in the ***JHeadstart Application Structure File Editor*** and uncheck the ***Generate View (UIX or JSP)*** checkbox. You'll find the property in the ***Generation Settings*** category.

Step 7: Apply Custom Skins to Change Look and Feel

You have undoubtedly noticed that the ADF web application we've built in this tutorial has a consistent look and feel on all the pages. This consistency is an important ingredient in delivering the attractive and easy-to-use experience that all application developer want for their end users. You might have also noticed that the pages we've built resemble those in the self-service web applications of the Oracle e-Business Suite. By default, ADF UIX pages are configured to have this Oracle Browser Look and Feel [11]...and with good reason. Over 2000 developers inside Oracle who implement and enhance the e-Business Suite use the ADF Business Components and ADF UIX technologies to build these self-service web applications, the same key technologies we're using in this tutorial.

But if it's not your style to have your application look like the Oracle e-Business Suite, no worries. The ADF UIX technology supports changing the look and feel of all your pages in a consistent way by applying a skin [12]. As we'll see, applying a skin does not require any changes to the application pages that you've built. It's just a configuration setting.

Step 7a: Changing Between Supplied Skins

ADF UIX comes with two built-in skins, `blaf` and `minimal`. The `blaf` skin is the default, and its name comes from the acronym for the Oracle "Browser Look and Feel" we discussed above. The `minimal` skin provides an alternative look that is simpler and that tries to minimize the use of downloaded images.

To change the look and feel of our tutorial application...

- Find the `uix-config.xml` file under the `WEB-INF` directory of your `ViewController` project's ***Web Content*** folder.
- Double-click the `uix-config.xml` file in the application navigator to view it in the code editor.
- Find the `<look-and-feel>` tag in this XML configuration file and notice that it looks like this:

```
<configurations xmlns="http://xmlns.oracle.com/uix/config">
  :
  <default-configuration>
    :
    <look-and-feel>blaf</look-and-feel>
    <!--
      <look-and-feel>minimal</look-and-feel>
    -->
  </default-configuration>
</configurations>
```

- To change to use the `minimal` look and feel, comment-out the `blaf` entry, and uncomment the `minimal` entry so that your `uix-config.xml` file looks like this:

```
<configurations xmlns="http://xmlns.oracle.com/uix/config">
  :
  <default-configuration>
    :
    <!--
      <look-and-feel>blaf</look-and-feel>
    -->
    <look-and-feel>minimal</look-and-feel>
  </default-configuration>
</configurations>
```

- Then save the file

The first thing we notice is that, as shown in Figure 85, the JDeveloper visual designer for the `Employees.uix` page changes to show the page in the minimal look and feel. This underscores how the visual page designer really does offer a true "What You See Is What You Get" (WYSIWYG) environment.

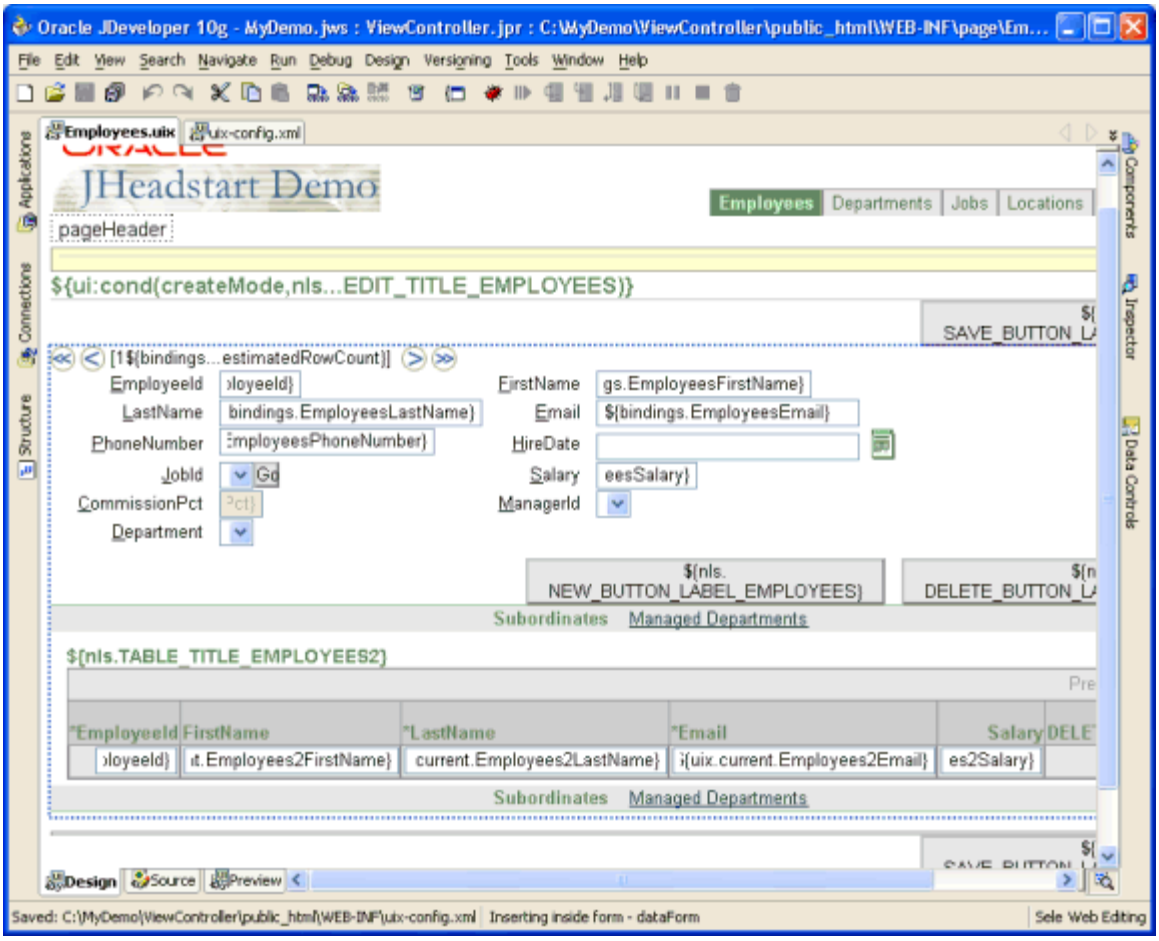


Figure 85: Visual Designer Updates to Reflect Custom Skin

By refreshing your browser, or rerunning the application if you don't still have it running, as shown in Figure 86 you can experience the effect of changing to the minimal skin at runtime as well. All of our application's functionality works the same as it did before, but the look and feel has been consistently modified with one edit to a configuration file.

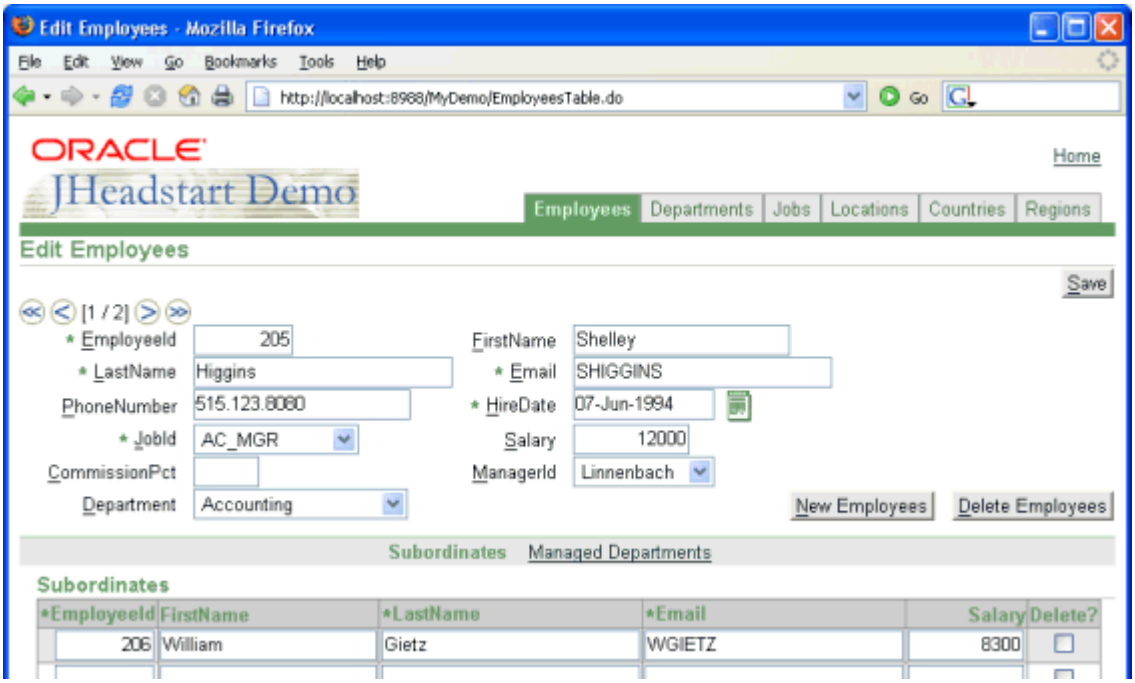


Figure 86: Tutorial Application Running with "Minimal" Skin

Step 7b: Exploring Custom Skins

If neither of the two built-in skins suits your fancy, you can also create your own custom ADF UIX skins. By visiting the Oracle Community Weblogs [13] website, as shown in Figure 87 you can experiment with dynamically changing between six different skins. Our `blaf` and `minimal` are in the list as expected, but then there are four custom skins to try experiment with as well. They are designed to "clone" the look and feel of four popular online destinations you might be familiar with. For more technical information on creating your own ADF UIX skins, see Jonas Jacobi's How to Create a Custom Skin for ADF UIX [14] article on OTN.

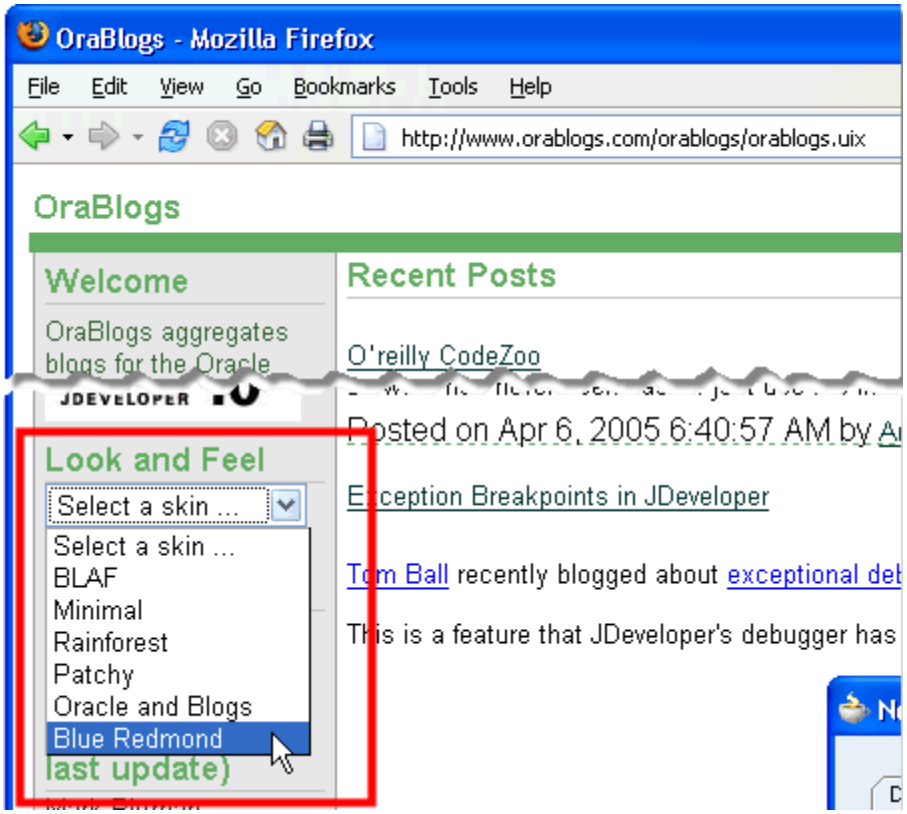


Figure 87: Online Example of Using Other Custom ADF UIX Skins

Conclusion

In this end-to-end tutorial we've experienced first-hand how Oracle JHeadstart 10g turbo-charges developer productivity for Oracle ADF-based web applications. Using a JDeveloper wizard to generate the back-end ADF Business Components that handle database interaction, and using the declarative JHeadstart Application Generator to iteratively generate the entire web front end, we built an attractive, consistent, interactive, and skinnable web application with browse, search, insert, update, and delete functionality against six related database tables from the Oracle HR sample schema. The application featured single- and multi-row editing, page-by-page scrolling, master/detail handling, dropdown lists, a pop-up LOV, a shuttle picker, and a tree control. We saw an example of how the JHeadstart Application Generator can be customized using a powerful templating mechanism to locally (or globally) change the basic structure of the pages it generates. We also saw that since JHeadstart is generating standard ADF artifacts and metadata, we can use the standard tools that JDeveloper provides to customize the generated pages where needed.

NOTE: For customers working today with Oracle Designer [15] — accustomed to the productivity they gain from model-driven, repository-based generation of complete applications — JHeadstart offers the one-two productivity "punch" that combines an enhanced generation of ADF Business Components from a Designer module component, as well as a fully-working UIX or JSP web application generated on top of them. This means that you can generate both the back-end and the front-end of your web application off the Designer repository using JHeadstart. For more information, see the JHeadstart Developer's Guide [2].

We claimed at the outset that no Java code was required to build the demo, and in fact none of the steps we followed above

required our writing any Java code. But you must be thinking, "Surely JHeadstart or JDeveloper itself must have generated *some* Java code to make all of that functionality work, right?" You might be surprised.

Figure 88 shows the result of opening a command shell, changing directories to the `C:\MyDemo` root directory where we've been working on the tutorial, and doing a recursive directory search for any `*.java` files. The executive summary: `File Not Found`. While a search for `*.uix` files, `*.properties`, or `*.xml` files will produce a number of hits, no Java code was needed to build this demo. All of the base functionality to support the features we have employed lives in the base ADF framework library components and some standard framework extension libraries that JHeadstart provides to complement its declarative application generation.

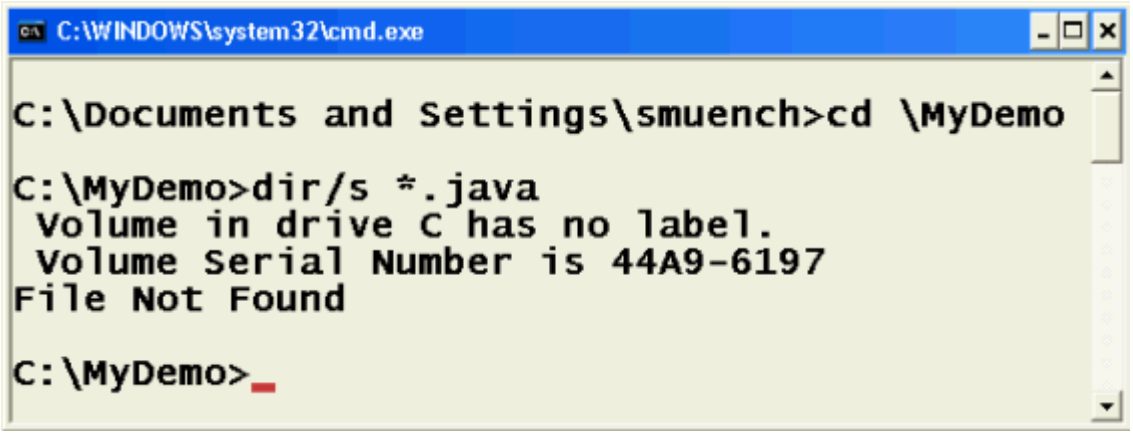


Figure 88: No Written or Generated Java Code Required For This Tutorial

This does not imply that building your *own* business applications with Oracle ADF and JHeadstart 10g Application Generator won't require any Java code. They undoubtedly will. However, what we're illustrating here in the tutorial is that Java code is not required for a huge amount of the basic functionality every business application needs. The code you'll end up writing will be code that is specific to your business application functionality, and not to making the basics of screen management, data management, or business rules enforcement work correctly.

Another interesting point to note in closing is that while we've been testing the tutorial application on the embedded Oracle Containers for J2EE (OC4J) server inside JDeveloper, the same ADF-based web application can run inside any J2EE application server. As shown in Figure 89, in addition to support Oracle's J2EE server, JDeveloper offers automated installation of the ADF Runtime libraries on a number of other popular J2EE servers as well. For servers not included in this automated list, JDeveloper can prepare a standard J2EE EAR file to be deployed using the deployment tool provided by that application server.

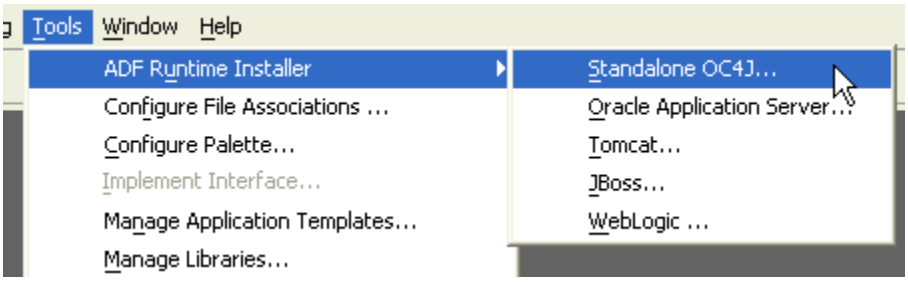


Figure 89: ADF Runtime Installer Automated Installation Options

So, in hoping that this tutorial has been both educational and eye-opening for you, we close with a few links to where you can find additional information, samples, and tutorials about JDeveloper 10g, Oracle ADF, and Oracle JHeadstart.

- JHeadstart Product Center on OTN [7]
- JDeveloper Product Center on OTN [16]
- Recommended Reading List on Oracle ADF [17]
- Oracle Community Weblogs (OraBlogs) [13]

If you have any questions on JHeadstart that are not answered with the above links, you can post them on the OTN JHeadstart Discussion Forum [18]. In addition, if you have any questions on Oracle ADF in general, you can post them to the OTN JDeveloper Discussion Forum [19] on OTN.

Related Documents

1. Building J2EE Applications with Oracle JHeadstart for ADF (PDF Format) [http://otn.oracle.com/products/jdev/tips/muench/jhs-step-by-step.pdf]
2. JHeadstart Developer's Guide [http://download.oracle.com/consulting/jhsdevguide.pdf]
3. ADF Business Components Design Pattern Catalog [http://www.oracle.com/technology/products/jdev/tips/muench/designpatterns/index.html]
4. JHeadstart Frequently Asked Questions [http://www.oracle.com/technology/consulting/9iservices/files/JHS_FAQ_EXT1012.html]
5. Tutorial Files for Offline Viewing and Database Setup [http://otn.oracle.com/products/jdev/tips/muench/jhs-step-by-step.zip]
6. JDeveloper Downloads [http://www.oracle.com/technology/software/products/jdev/index.html]
7. JHeadstart 10g Product Center [http://www.oracle.com/technology/consulting/9iServices/JHeadstart.html]
8. Installing the Sample Schemas and Establishing a Database Connection [http://www.oracle.com/technology/obe/obe_as_1012/j2ee/common/obeconnection.htm]
9. Oracle Magazine DEVELOPER: Frameworks [http://www.oracle.com/technology/products/jdev/tips/muench/oramag/index.html]
10. Asynchronous JavaScript and XML (AJAX) [http://en.wikipedia.org/wiki/AJAX]
11. Oracle Browser Look and Feel [http://www.oracle.com/technology/tech/blaf/index.html]
12. Wikipedia Definition of 'Skin' [http://en.wikipedia.org/wiki/Skin_(computing)]
13. Oracle Community Weblogs [http://www.orablogs.com]
14. How to Create a Custom Skin for ADF UIX [http://www.oracle.com/technology/products/jdev/howtos/10g/adf_uix_laf_ht/index.html]
15. Oracle Designer Product Center on OTN [http://www.oracle.com/technology/products/designer/index.html]
16. JDeveloper Product Center on OTN [http://www.oracle.com/technology/products/jdev/index.html]
17. Recommended Reading List on Oracle ADF [http://www.oracle.com/technology/products/jdev/tips/muench/requiredreading/index.html]
18. OTN JHeadstart Discussion Forum [http://forums.oracle.com/forums/forum.jsp?forum=38]
19. OTN JDeveloper Discussion Forum [http://forums.oracle.com/forums/forum.jsp?forum=83]

Appendix 1: Adding a Validation Rule for Extra Credit

In this extra-credit step of the tutorial, we'll add some declarative validation logic to one of our business domain objects. This will illustrate how business logic and user interface are cleanly separated in an Oracle ADF application, as well as show how the errors are displayed by default to the end-user.

Using Oracle ADF our business validation logic is nicely encapsulated inside the entity objects that represent our business domain objects. You can capture a number of aspects of business domain validation declaratively. Here we'll illustrate two simple examples of enforcing that an attribute is mandatory, and that its value lie within a certain numerical range.

- ***Make an Employee's Salary Mandatory***

  In your `Model` project, find the `Employees` entity object and double-click it to edit it. In the ***Entity Object Editor***, as shown in Figure 90 expand the ***Attributes*** node in the tree at the left and select the ***Salary*** attribute. Then, in the panel on the right, check the ***Mandatory*** checkbox.

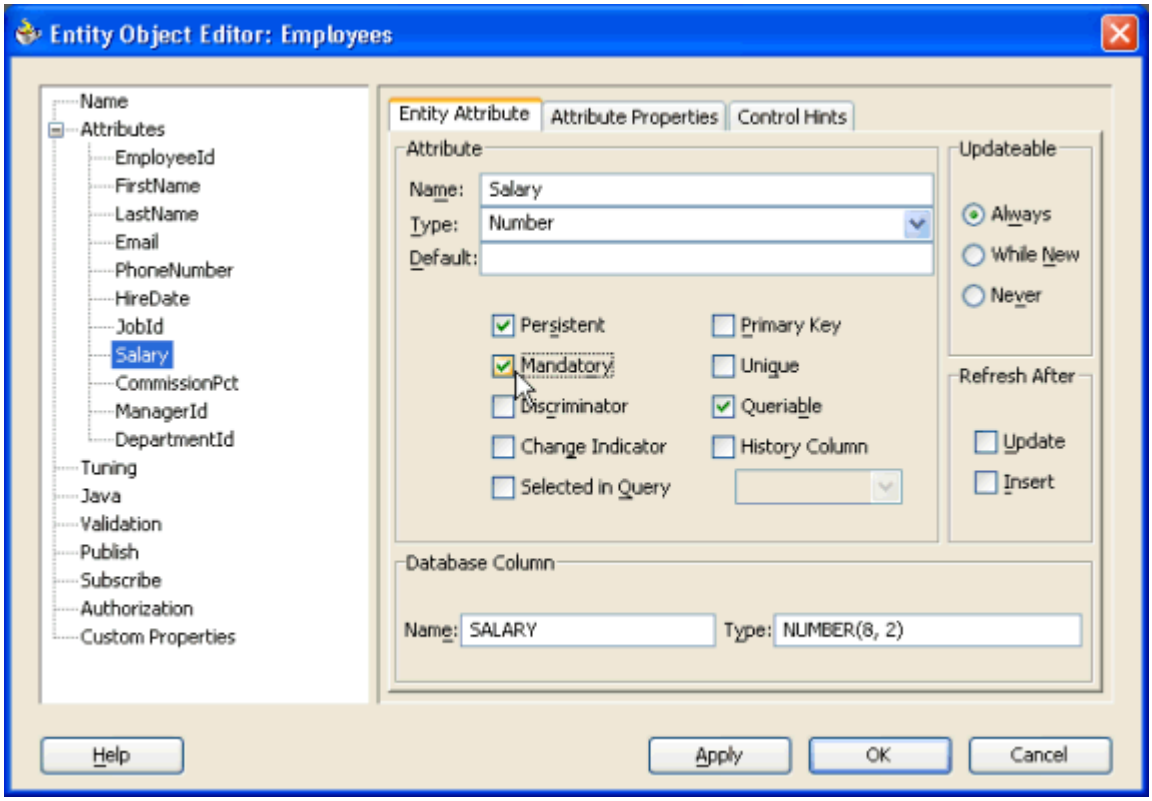


Figure 90: Making the Salary Attribute Mandatory

- ***Validate that Salary is Between 1 and 25000***

  While still in the ***Entity Object Editor*** on the `Employees` entity, select the ***Validation*** panel. As shown in Figure 91, select the Salary attribute and click the ***(New...)*** button to add a new validation rule.

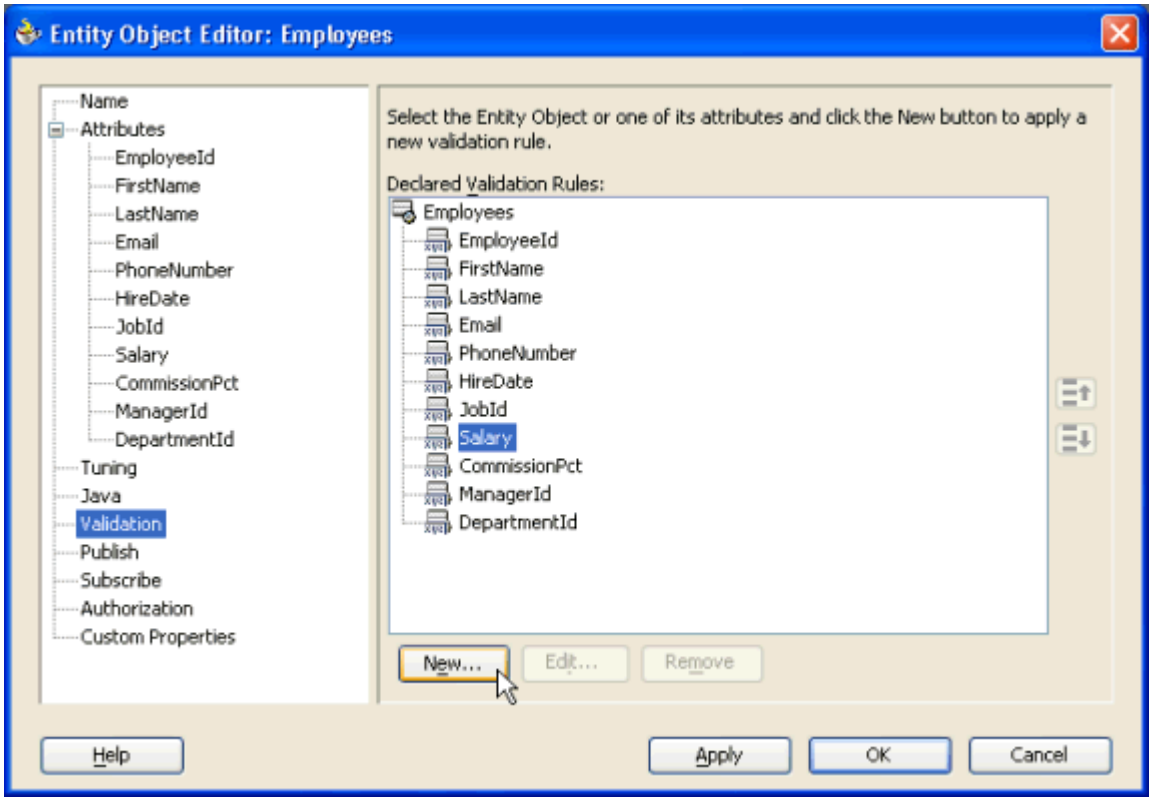


Figure 91: Adding a New Attribute Validation Rule

In the ***Add Validation Rule for: Salary*** dialog, as shown in Figure 92, select `RangeValidator` from the Rules dropdown. Enter a ***Minimum Value*** of `1` and a ***Maximum Value*** of `25000`. Finally, type in an error message like ***Salary must be between 1 and 25000***, then click ***(OK)*** to add the rule.

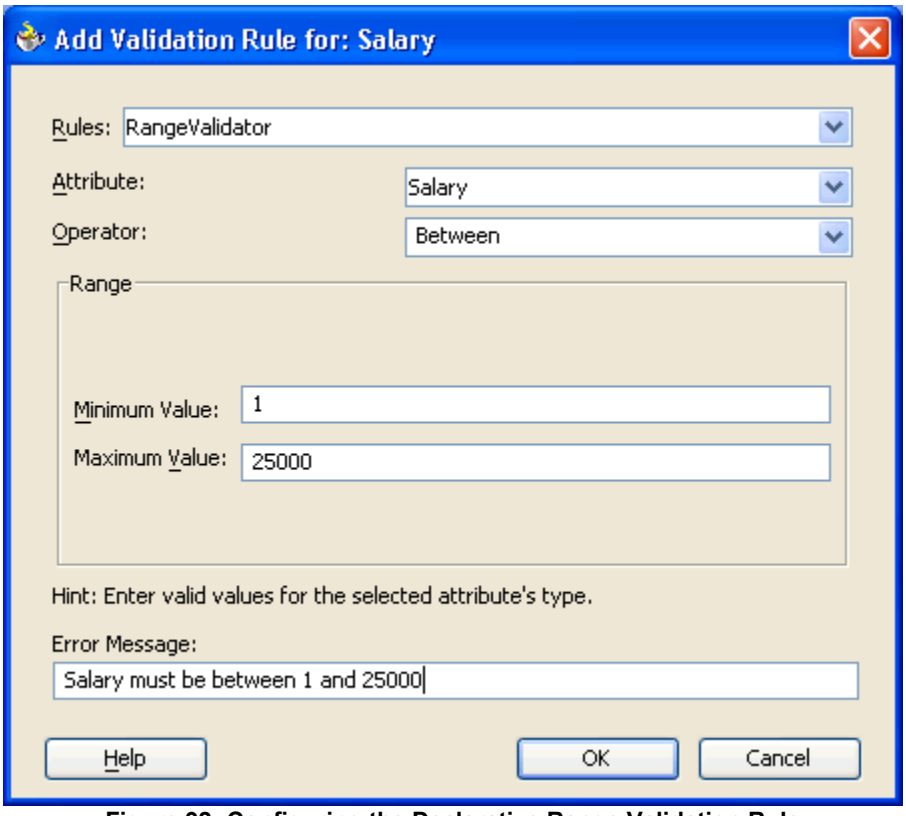


Figure 92: Configuring the Declarative Range Validation Rule

As shown in Figure 93, any validation rules you've added to attributes or at the entity object level appear in the tree display on the validation panel.

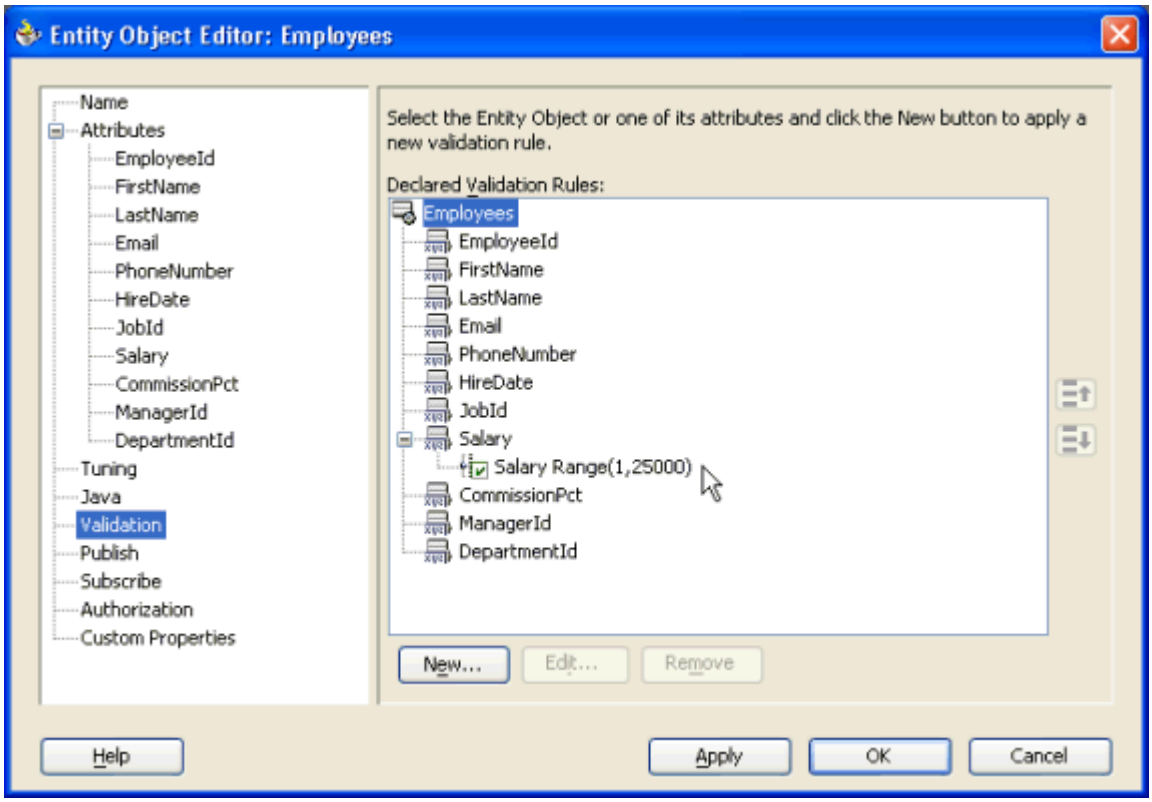


Figure 93: All Rules Appear in Validation Panel

At this point, let's re-run the application to see the effect of having added two simple validation rules. As we've done before, select the `ViewController` project in the Application Navigator and press `F11` to run.

When the initial ***Employees*** tab appears, enter the number `100` in the quick-search ***Filter By*** field and click ***(Go)*** to find the employee with `EmployeeId` 100. Click the ***(Details)*** button to edit the data for employee "Steven King".

First try blanking out the existing value of ***Salary*** and pressing the ***(Save)*** button. Since ADF UIX automatically offers complementary client-side mandatory attribute enforcement, you'll see an error box as shown in Figure 94.

Figure 94: ADF UIX Can Enforce Some Validations on Client

Next, try to enter a ***Salary*** value of `25001` and click ***(Save)*** again. Since this value is outside of the valid range, as shown in Figure 95 we see that the end-user is shown the custom error message we provided when we added the range validation rule on the `Employee` entity object's `Salary` attribute.

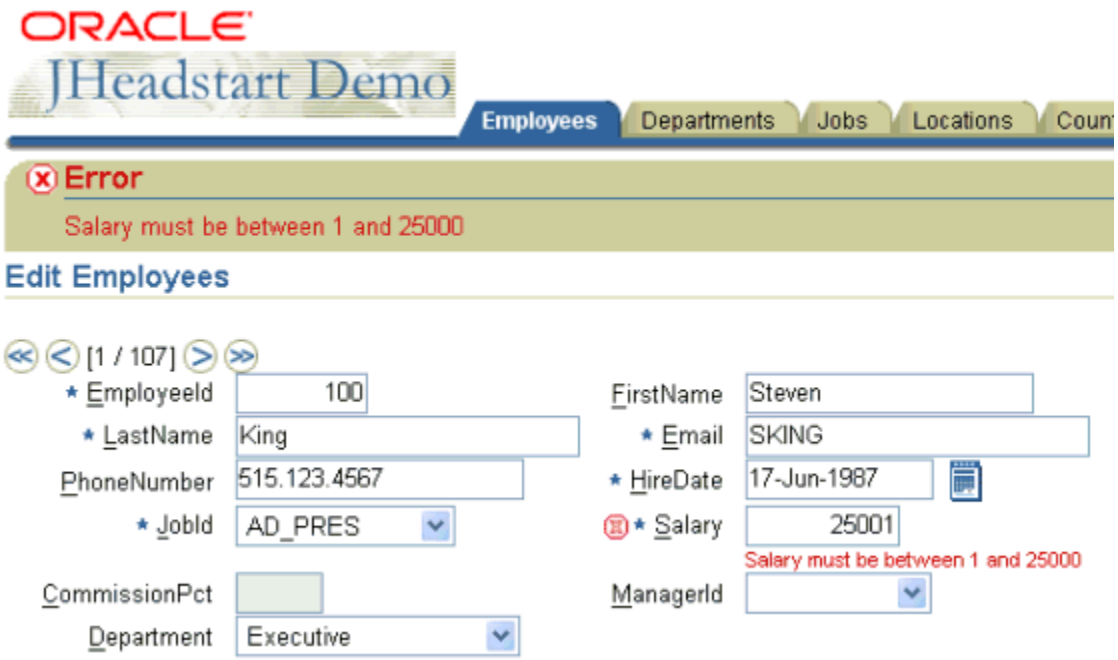


Figure 95: Validation Errors Presented Automatically to User

Under the covers, what makes this work is the automatic coordination that's occuring between the front-end ADF UIX components and the view objects in the back-end ADF application module. Those view objects, in turn, coordinate with your underlying busing domain layer — realized as a set of ADF entity objects — which encapsulate the business validation. Any errors raised by the business layer automatically "bubble up" back to the user interface and are presented to the user. Again — as we've seen throughout this tutorial — all of this infrastructure is provided for you by the Oracle ADF framework so you don't have to write the "application plumbing" code.

If you repeat the `dir /s *.java` test we performed above in the `C:\MyDemo` directory, you'll notice that adding the declarative range-validation rule above for the `Employees` entity resulted in the creation of a Java message bundle file for the `Employees` entity in order to capture the custom error message. If you look in the file, you'll see that there's no real code in there; a message bundle is basically a Java class that contains translatable strings like the error message we entered above:

```
sMessageStrings = {
  {"Salary_Rule_0", "Salary must be between 1 and 25000"}
};
```

The key point to take home is that — whether your prompts and error messages are in `*.properties` files or Java message bundles — applications you build with Oracle ADF and Oracle JHeadstart 10g for ADF are setup out of the box to use the best-practice techniques to make building even multi-lingual applications easy!