Oracle ADF Faces Cookbook

Amr Gawish

Chapter No. 2
"Getting Started with ADF Faces and JDeveloper"
In this package, you will find:
A Biography of the author of the book
A preview chapter from the book, Chapter NO.2 "Getting Started with ADF Faces and JDeveloper"
A synopsis of the book’s content
Information on where to buy this book

About the Author

Amr Gawish began his career at the age of 18, working as a web designer before entering college. He is very passionate about technology and always tries to push the limits of the technologies he uses.

He completed his bachelor's degree in Math and Computer Sciences from Al-Azhar University in Egypt and is currently pursuing his master's at the University of Liverpool. He is currently employed as an Oracle Fusion Middleware consultant and is certified in Java SE 7, Oracle ADF, WebCenter Portal, and Oracle SOA Suite. He has worked with all these products and the rest of the Oracle middleware stack for more than six years.

He has also gained a fair amount of experience in various other topics such as Gamification, Scala programming, and Akka and is currently working on microcontroller programming with Raspberry Pi and Arduino and robotics.

For More Information:
You can learn more about him by visiting his website, www.amr-gawish.com, or follow him on Twitter (@agawish).

He is currently working with infoMENTUM (www.infomentum.com), which is an Oracle Platinum Partner; a leading company in Oracle Fusion Middleware; and the first company to specialize in WebCenter (both content and portal), Oracle SOA, ADF, and Java in the EMEA region.

Oracle ADF Faces Cookbook

Oracle ADF Faces is the view/controller part of the Oracle ADF end-to-end framework. ADF Faces has more than 150 Ajax-enabled components that help developers to rapidly build applications that are robust, reactive, and easy to use.

In this book, we will learn how to deal with the different features of the ADF Faces framework. In the first part of the book, we will learn how to install JDeveloper and ADF, create a simple Oracle ADF application, present data in different ways using ADF Faces components, and use common ADF components such as inputs, menus, toolbars, and more.

In the second part of the book, we will understand how to create a unified template for ADF Faces applications, apply different skinning techniques, and use different visualization components and graphs.

In the last part of the book, we will learn how to use partial page rendering and different ADF Faces events; create reactive applications using polling, push, and WebSockets; add validation and conversion for different ADF Faces inputs; and create different ADF Faces resources for reuse. Finally, we will discuss some best practices, tips, and advice on how to scale and tune your ADF Faces application.

What This Book Covers

Chapter 1, Building Your ADF Faces Environment from the Ground Up, provides the necessary steps to build your environment and install different software to work with ADF Faces.

Chapter 2, Getting Started with ADF Faces and JDeveloper, focuses on getting you started with a simple ADF application and showing you how to deal with JDeveloper IDE.

Chapter 3, Presenting Data Using ADF Faces, shows different ways to present business service data with highlights over i18n and l10n.

Chapter 4, Using Common ADF Faces Components, describes how to work with the common ADF Faces components.

Chapter 5, Beautifying the Application Layout for Great User Experience, provides different techniques of creating great looking application by creating templates and skins, and by using advanced skinning techniques.

Chapter 6, Enriching User Experience with Visualization Components, explains how to work with different ADF Faces Visualization components to create great dashboards.

For More Information:
Chapter 7, Handling Events and Partial Page Rendering, describes how to use partial page rendering and different ADF events. It also provides an explanation of how to create great reactive applications by understanding how to use polling, push, and work with WebSockets.

Chapter 8, Validating and Converting Inputs, explains how to provide conversion and validation for different input components to insure the validity of the data returned to the service layer.

Chapter 9, Building Your Application for Reuse, shows how to create different resources for reuse, such as task flow templates, declarative components, contextual events, and how to package your application for reuse.

Chapter 10, Scaling Your ADF Faces Application, provides multiple tips and advice of how to scale your application and tune its performance.

For More Information:
Introduction

In today's environment, enterprise applications have become more and more complex; they need the underlying technology to be scalable, distributed, component based, and mission critical.

Designing and developing such enterprise applications means going through hundreds of requirements; failure to meet any of these requirements can lead to the failure of the whole project. However, Oracle ADF comes to the rescue. Oracle ADF is a Java EE framework that inherits Java EE security, robustness, and scalability.
It also adheres to the rapid application development principles in today's agile world. These principles provide the developer with a rich out-of-the-box functionality to focus and give his/her 100 percent to the business logic.

Oracle ADF is an end-to-end framework that follows the **Model View Controller (MVC)** pattern as illustrated in the following diagram:

Oracle ADF offerings vary between mobile solutions with Oracle ADF Mobile and the Desktop Integration solution, which can be integrated with Microsoft Office Excel with Oracle ADF Desktop Integration. It also provides a free offering through which developers can download, develop, and deploy enterprise applications with Oracle ADF Essentials for free. Additionally, they are all evolved from the core Oracle ADF framework, which is the Java EE framework where they extend beyond MVC and integrate with other Oracle middleware solutions.

To know more about what Oracle ADF has to offer, check its main page on Oracle, [http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html](http://www.oracle.com/technetwork/developer-tools/adf/overview/index.html).

In this book, we will only focus on ADF Faces and ADF Controller of the Oracle ADF framework. To make things easier, we will call them Oracle ADF Faces. Although Oracle ADF Faces exists in both the commercial Oracle ADF and Oracle ADF Essentials (the free version), we will only focus on the commercial part of it in this book since we are discussing the latest version of the Oracle ADF framework.
Oracle ADF Faces is built on top of the **JavaServer Faces (JSF)** framework, which is the standard Java EE framework implementation. Being built on top of the JSF framework makes Oracle ADF a component-based framework just like JSF.

JSF uses XML files called View templates as its view layer and a **FacesServlet** that acts as a controller in which it processes requests, loads the appropriate View template, builds a component tree, processes events, and renders the response (typically in the HTML language) to the client. Check the following diagram for more information about the JSF lifecycle:

The state of UI components and other objects' of scope interest is saved at the end of each request in a process called state saving, which can be saved either on the client side or on the server side (more on this in **Chapter 10, Scaling Your Application**) and restored upon the next creation of that view.

Oracle ADF Faces extends all that we've discussed so far and adds the following to JSF:

- It adds more than 100 additional components that support Ajax
- It enhances the JSF lifecycle by adding ADF-specific phases that add support for task flows, more scope definitions, and the ADF Binding layer

In this chapter, we will take our first steps toward creating our first ADF application; in this application, we will start from scratch and even build the Business Service Layer quickly (this book doesn't focus on the Business Service Layer, but since it's the first step, it'll be included in this chapter).
In order to complete the recipes in this chapter, you need to have the HR schema in your database unlocked and up and running.

If you are a skilled ADF developer, you can skip this chapter.

### Creating an Application Workspace

In this recipe, we will get to know a little bit about the JDeveloper IDE and how we can navigate around it; after this, we will start with the first steps of creating our ADF application by creating the ADF Application Workspace.

### How to do it...

In order to create the ADF Application Workspace, perform the ensuing steps:

1. **Start JDeveloper**; you will find different panes highlighted in the following screenshot:

   ![JDeveloper IDE Screenshot](Image)

   One of the main panes is the **Application** navigator pane (the top-left pane), which displays the application files. If there are no applications (such as the case when you open JDeveloper for the first time), you end up with two links: one for creating a new application and another for opening a pre-existing application.

   The center pane is the main pane, which will be the **Start Page** when you start JDeveloper.

---

**For More Information:**

The bottom-left pane is the **Structure** pane, that gives you the structure of the center pane. If a Java class is in the main pane, the structure pane will contain all members and methods of this class. You can maximize or close any pane in JDeveloper, and if you want to open it again, you can do so from the **Window** menu.

Create a new ADF application by clicking on the **New Application**... link from the **Applications** pane or navigating to **File | New | Application**.

2. Choose **ADF Fusion Web Application** as your application type and click on **OK**. Now the **Create ADF Fusion Web Application** dialog pops up.

3. Choose a name for your application; I'll call it **HelloADFFaces**.

4. Choose an application package prefix for your application; for example, if your company is called ACME, and you are making this application commercial and the application name is HR, the prefix should be something like `com.acme.hr`. In this case, the application is `chapter2.helloadf` and it'll be commercial. So, the prefix will be `com` and the company `adffaces`; therefore, it'll be `com.adffaces.chapter2.helloadf`, as shown in following screenshot:

5. Click on **Finish** to end the wizard.

---

For More Information:

How it works...

When you finish the wizard, you will end up with a page in the center pane called **HelloADF Faces Overview**, which has a checklist of what you should do (the plan of your application). The **Applications** pane now has two projects: one called **Model** and the other **ViewController**. These projects have been created for you by the wizard. The **Model** project should hold your Business Service Layer and **ViewController** will hold your pages and all of your controller's logic.

Notice that we clicked on **Finish** instead of **Next**, which is why you didn't witness the creation of the **Model** and **ViewController** projects. In the wizard, you can change their names and technology scopes. There is also another step at the end of the wizard in which you can specify your build tool, either the default JDeveloper build tools, Maven, or Ant. The checklist is very helpful for planning your ADF application, especially if you are starting with ADF development; you should also check the plan to find good resources and references to start your ADF application.

We will put the ADF project together by embracing the JDeveloper ADF plan and following the checklist of steps to be done.

See also

For more information about the JDeveloper user experience, check the following excellent video: [http://download.oracle.com/otn_hosted_doc/jdeveloper/12cdemos/JDevOverview12c/JDevOverview12c.html](http://download.oracle.com/otn_hosted_doc/jdeveloper/12cdemos/JDevOverview12c/JDevOverview12c.html).

Connecting your application to a database

In this application, we want to display employees' data from the HR schema on a web browser and be able to navigate between different records using buttons that help us go to the first, next, previous, and last records; employees' information should be displayed in a form-based view.

In order to retrieve employees' information, we need to establish a connection with the HR database.

In this recipe, we will use JDeveloper to establish a connection between our new ADF application and the HR schema, which comes by default with the Oracle database.
How to do it...

In order to establish a connection with the HR schema, follow the ensuing steps:

1. Expand the **Connect to a Database** checklist item from the checklist and click on the **Create a Database Connection** button that will open the **Create Database Connection** dialog.

   ![Create Database Connection dialog](image)

   You can also create a connection by navigating to **File | New | From Gallery | General | Connections | Database Connection**.

2. Choose a connection name of your choice; I named it **HRConnection** for clarification.

3. Change the username and password of the HR user to match your database's HR schema; make sure you are pointing to the right host.

4. Click on the **Test Connection** button to make sure everything works fine, and if everything is fine, you should see a **Success!** message.

5. Click on **OK** to close the dialog box.

6. Mark this item as done in the checklist as well.

For More Information:

How it works...

When working with applications in ADF, you will need to have connections in order to establish a communication channel with different systems. One of the important connections that will be usually needed is a database connection, which establishes a connection between your application and database.

When you entered the information, you provided the connection with all the information needed in order to connect to a specific user in a specific database instance.

Your connection can be an application-based resource like the one we configured in this recipe; alternatively, it can be inside JDeveloper's resource catalog, which can be helpful if you have multiple applications that use the same database connection.

After creating a database connection, we need to create the Business Service Layer that will use the database connection.

Building Business Service

In this recipe, we are going to build our Business Service Layer, which consists of one entity object based on the EMPLOYEES table, one view object based on the entity object, and one application module to host the latest view object.

In a real application, it's always a good practice to establish a uniform naming convention for your business components, such as suffixes and package names, which can be found by navigating to Tools | Preferences | ADF Business Components | Object Naming and Packages.

This will be the only recipe that talks about Business Service in brief in this book. If you want to know more about building Business Service, check the official documentation at http://docs.oracle.com/middleware/1212/adf/ADFFD/partpage2.htm#ADFFD20093.

How to do it...

In order to build your Business Service, follow the ensuing steps:

1. Right-click on the Model project node from the Applications navigator pane and navigate to New | Business Components from tables...; by doing this, you are able to create all the three objects in one wizard instead of creating everything separately.

You can expand substeps in the Business Service checklist item and create each of them separately if you want; it'll be a good practice for you as well.

For More Information:

2. If this is the first time you’re dealing with business components in this application, a dialog will pop up asking you to initialize the business components project. It also asks about database connection. If you only create one database connection, it should appear preselected; if not, you should select it from the drop-down list or create a new one using the green plus icon; once this is done, click on OK to proceed. The **Create Business Components from Tables** dialog should pop up.

3. Then, you define **Entity Objects**; click on the **Query** button to query all tables in the HR schema, select the **EMPLOYEES** table, and shuttle it into the selected panel.

4. Make sure that **Entity Objects** is created in the right Java package, which should exist in `com.adffaces.chapter2.helloadf.model.entities` based on the best practices of ADF BC. Also, it should not be mixed with application modules and view objects, so the result should be **Employees**.

5. Click on **Next**.

   ![Create Business Components from Tables](image)

6. Create **Entity-based View Objects** that are traditionally known as updatable view objects; you should shuttle the Employees table from the left to the right panel.

7. Make sure that the package name follows the best practice by adding the `views` suffix; the package name will be `com.adffaces.chapter2.helloadf.model.views`. Also, make sure you follow the best practice by having a suffix for your view objects instances by adding the `View` suffix, which in this case should be `EmployeesView`.

8. Click on **Next**.

9. Skip **Query-based View Objects** as we don't have any of these in this simple application; then click on **Next**.

10. Create an **Application Module** and naming it `HrAppModule`, which follows the best practice of adding the `AppModule` suffix at the end of the name and appending `services` to the package name. So, in this case, it should be `com.adffaces.chapter2.helloadf.model.services`.

For More Information:

11. Click on Finish to close the dialog box. You should end up with the following structure in your Applications pane if you have followed the best practices in the ADF guide:

```
com.affafaces.chapter2
    helloadf
        model
            entities
                association
                    Employees
                        EmployeesPK
                            Assoc
                Employees
                services
                    AppModule
        views
            links
                Employees
                    EmployeesPK
                        Link
                EmployeesView
        AppModule
```

12. Place a tick on the third item of the checklist to indicate it is done.
How it works...

In order to build a proper Business Service, you need to create ADF business components from your database tables, which will require you to create the following:

- **Entity object(s)**: These are like one-to-one mappings with database tables so the structure of the entity bean should match the database table structure along with the added value of validations.

- **View object(s)**: These are like `select` statements. It's the data that users can see and is based on the Entity object that makes this view object support the Update, Add, and Delete row operations; alternatively, it can be based on a SQL statement, which means it'll be read-only.

- **Application module(s)**: The application module is the window of your Business Service to the user interface. You can expose view objects or other application modules inside of it, and it'll be available to the ADF Binding layer, where it can be dragged to the user interface inside JSF pages, using the drag-and-drop feature.

After you create your entity object, view object, and application module, your Business Service is in place, and with your `HrAppModule`, you automatically created your first Data Control.

A Data Control can be created for Java objects, web services, and more; however, it is created automatically with every application module you create inside your Business Service Layer.

You can see the created Data Control by expanding **Data Control** from the **Applications** pane; you will find the `HrAppModuleDataControl`, and by expanding it, you will find `EmployeesView1` underneath it, which represents your view object.

If you want to make sure that everything is correct with your Business Service, you can start the Oracle ADF Model Tester by right-clicking on the `HrModule` application module and then clicking on **Run**.

Now you are ready to define your application flow and Finish Step 4 in the Application Overview checklist.

---

For More Information:  
Defining the page flow

When we talk about defining the page flow of our application, we talk about it in terms of how pages interact with each other, designing what is the right sequence of pages that the user has to navigate through; also, designing how work units are arranged in pages, and how they can be modularized and communicate with each other as well.

In this recipe, we move from the Business Service layer to the Controller layer. In order to work with page flows, we need to first understand what ADF Task Flow is.

ADF Task Flows provide a modularized approach to define the control flow in an ADF application. So, instead of representing an application as a single large page, you can break it up into a collection of reusable task flows.

Each task flow contains a portion of the application's navigational graph and can be considered as a logical business unit of work.

Each task flow contains one or more nodes that are called activities. An activity node can represent a simple logical operation such as displaying a page or a page fragment, executing an application's logic, or calling another task flow.

The transitions between the activities are called control flow cases and can represent navigations between different pages.

There are two types of task flows:

- **Unbounded task flows**: These are task flows without a specific start point that can contain one or more activities, control flow rules, and managed bean definitions. The application can interact with any activity inside of unbounded task flows without having to take a specific path. Usually, it represents the application's main navigational page model such as the adfc-config Task Flow.

- **Bounded task flows**: Unlike the unbounded task flow, a bounded task flow has a single entry point (activity) and zero or more exit points. It has its own private activities, control flows, and Managed Bean definitions. Bounded task flow allows the reuse of parameters, transaction management, re-entry, and routing; it also allows us to save a state and more. Unlike unbounded task flows, it can be rendered within the ADF region inside a JSF Page.

You can have an overview of task flow and task flow-oriented architecture by watching this great video: http://www.youtube.com/watch?v=TajCHL7Hw5M

In this recipe, we will create one bounded task flow. We won't create any unbounded task flows, but we will be using the adfc-config task flow later.

For More Information:
How to do it...

In order to create the bounded task flow, follow the ensuing steps:

1. Expand the **Design Application Flow** checklist item and click on the **Create a Task Flow** button. A pop-up window will pop up asking which project you wish to create this task flow for; choose ViewController and click on **OK**. Another dialog window will pop up asking you to create your task flow.

2. Name the task flow file `retrieve-employees-information`. The name should be meaningful. It'll be better if you arrange your task flows into different physical directories under the **WEB-INF** directory based on their functionalities; however, for my simple application, I'll retain the default setup.

3. Make sure that the **Create as Bounded Task Flow** checkbox is checked.

4. Uncheck the **Create with Page Fragments** option and then click on **OK**.

You can create your task flow with page fragments but then this task flow can't be executed on its own and needs to be part of a JSF page inside the ADF region.

For More Information:
Once you create your task flow, you should be directed to it; if this doesn't happen, you can easily locate it by navigating to Applications Navigation pane | ViewController | Web Content | Page Flows.

When you open the task flow, you will be directed to the Diagram view asking you to place components from the Components pallet on the right-hand side.

1. Expand the Components category under Components palette if not expanded; locate the View activity and drag-and-drop it inside the task flow diagram view.

2. Name the newly added View activity employees. You can see that there is a green circle around it, which indicates default activity; you will also see a purple triangle over it, which means that this page has not been created yet.

3. Save everything using Ctrl + S or clicking on the double disk icon on the toolbar.

4. Open the HelloADFFaces Overview tab and check the Design Application Flow checklist item as done.

If you close the Overview window, you can re-open it again by navigating to Application menu | Show overview.

How it works...

In this recipe, we created a simple bounded task flow that has one activity, which represents a View; a View means a page in our case, which will be physically created in the next recipe.

For More Information:
Creating a View page

In this simple recipe, we will create the actual JSF page behind the Employees View activity inside the retrieve-employees-information task flow that was created in the previous recipe.

How to do it...

In order to create the JSF page behind the View activity, follow the ensuing steps:

1. Open the retrieve-employees-information task flow and then double-click on the employees View activity. A dialog should pop up asking for the filename and the page template for this page and/or Managed Beans for it. For this simple application, we won’t create page templates as we will use Oracle Three Column Layout.

2. Leave the default filename as is.

3. Choose the Oracle Three-Column layout under Page Layout tab | Reference ADF Page Template. In Chapter 5, Beautifying Application Layout for Great User Experience, we will learn how to create our own rich page template.

4. Close the pop-up window by clicking on the OK button.

5. Open the HelloADFFaces Overview tab and check the Design Pages checklist item as done.

For More Information:
The page should be opened automatically, but if it isn't, you can open employees.jsf by navigating to Applications Pane | ViewController | Web Content. You can see that the Oracle three-column template is loaded, and you are able to see the main structure of the page.

**How it works...**

When you create a page, you have two options:

- Creating a `.jsf` page (recommended since 11g Release 2) that uses Facelets
- Creating a `.jspx` page, which is given for backward compatibility with JDeveloper release 1 so that you can have a seamless migration from 11gR1 to 12c

You are also asked to choose a page template, which is a great way to have a page structure that can be applied to your pages, without having to worry about maintaining it for each page separately. You can also have a single point of change that can be applied to all the pages that use this template. JDeveloper and ADF come with the Oracle three column template that can be used.

While creating this page, we reached a status where we can run and display what we've done; however, the page is currently empty and we want to have the employees' data, so first let's place some components on the page before running it.

**Adding ADF Faces components to JSF pages**

Now, for the magic to happen from the Business Service Layer to the view layer, in order to place the data, you can select the data that you want to insert from the Data Control and drag-and-drop it onto our page and choose the format that this page should have. So let's do that.

What we want is to have a form that has all employees' information inside our page.

**How to do it...**

Let's see how to add ADF Faces components to JSF pages:

1. Expand `Hr AppModuleDataControl` under Data Controls and drag `Employees1` into the white square inside of the `employees.jsf` page. Remember this is the view object that contains all employees' information, and it's based on the `Employees Entity` object.

For More Information:  
2. Choose **ADF Form**... when the drag-and-drop dialog appears as shown in the following screenshot:

Now, a new dialog pops up asking you what you want to display in your form.

3. Check the **Row Navigation** checkbox.

This will place all navigation buttons such as **First**, **Next**, **Previous**, and **Last** under the form.

4. Click on **OK** to close the dialog.

5. Open the **HelloADFFaces Overview** tab and check the **Add Common Components** checklist item as done.

---

For More Information:

How it works...

You can notice that, inside the form's input, you have something like #{bindings.FirstName.inputValue}. This Expression Language (EL) expression represents the binding that gets created automatically once you have dragged-and-dropped the view object onto the page, which is a proof of the ADF Binding layer in action.

You can see these bindings by navigating to the **Bindings and Executables** tab, which will open the Page Data Binding Definition of the employees.jsf page, which looks like the following screenshot:

You can see that FirstName is under the **Bindings** section, and FirstName is pointing to something called Employees1Iterator under the **Executables** section. The **Executables** section is pointing to Employees1, which is the child of HrAppModuleDataControl under the **Data Control** section.

The **Bindings and Executables** sections are very important to understand since the whole ADF Binding mechanism is based on them. The **Executables** section represents activities that get executed when the page is ready to be viewed. So, Employees1Iterator is getting executed to get all the data from the Employees1 view object, and the data will be available inside the Employees1Iterator object. That's why, when you click on it, you will get to edit some properties like how often this iterator should get refreshed, should the result get cached, and more features such as these.

For More Information:

As for the **Bindings** section, they are bound to those results data, and to items that can be referenced by components in your JSF page such as input Texts and Buttons. Also if you check the last four bindings, you will find them corresponding to four buttons in the page. These are actual actions that manipulate the data inside the iterator, which by definition manipulates the data in the view object as well. You can also see all these actions from the Data Controls pallet directly by expanding the view object, and you will find a folder called Operations; expanding it will reveal all the operations you can have for this view object, which covers the main **CRUD (Create Retrieve Update Delete)** operations plus a few other operations. You also have two operations that are not related to a specific view object but related to the whole Data Control, and these are Commit and Rollback (To commit to database or roll-back changes).

The Page Definition is an XML file that is linked to the employees.jsf page. This link is the one that makes **Bindings** available to the employees.jsf page's EL as part of the ADF Faces lifecycle. The link between the page and Page Definition happens in a specific file called DataBindings.cpx, which is also an XML file that has all pages and their corresponding Page Definitions.

To learn more about ADF Model and ADF Binding layer, check the official documentation at [http://docs.oracle.com/middleware/1212/adf/ADFFD/bcdcpal.htm#BABHJJHA](http://docs.oracle.com/middleware/1212/adf/ADFFD/bcdcpal.htm#BABHJJHA).

Now that you have finished creating the form inside the page, it is time to see everything in action and run your application.

---

For More Information:

Running your first ADF Faces application

In this recipe, we will run our application in multiple scenarios by trying the worst scenario first and enhancing it till we reach the best scenario.

How to do it...

To run your first ADF Faces application, perform the following steps:

1. Locate your task flow `retrieve-employees-information` and right-click on it.
2. Click on Run or select it and click on Run (the green arrow icon) on the toolbar, and the page will start appearing.

If this is the first time you run an application in JDeveloper, a pop-up window will ask you to instantiate your first Weblogic instance and will ask you about the password for your weblogic user and port numbers.

If you did everything correctly, you should see a form that you can navigate through with Next, Previous, First, and Last.

For More Information:
Getting Started with ADF Faces and JDeveloper

However, if you check the URL, it's quite ugly. It's not quite what you have in mind, so how can it be enhanced?

This method is not recommended and is only used for debugging and testing Page-based task flows, but it's worth mentioning.

The second scenario is where you run the page directly. So let's try that and see if there are any changes.

1. Right-click on the employees.jsf page and run it. You will probably get a warning message from JDeveloper telling you that running the page itself is not recommended, which is true, but ignore it for the moment and click on Yes; let's examine the end result.

The URL is much better now; it doesn't show all the adf.task-flow parameters in the URL that expose a bit of sensitive information about our task flow.

But still, the warning from JDeveloper means that this way is also not recommended, so what is the best way to run your application?

Well, if you remember, we said that the adfc-config task flow and usually any unbounded task flows are generally responsible for the application's navigation model, so why don't we use it? In order to do so, all we have to do is expose the employees.jsf page as a View activity in that task flow.

2. Open the adfc-config task flow.

3. Drag-and-drop the employees.jsf page from the Application Pane into the body of the task flow to have it look like the following screenshot:

For More Information:
4. As you can see, the activity's name is employees without the .jsf part, and this is what is going to be shown in the URL.

By doing this, we added the employees.jsf page as a View activity inside the adfc-config task flow, which as you know can act as an entry point since unbounded task flow has unlimited entry points.

5. Right-click on adfc-config and run it; you will now see an even more enhanced version of the URL. You will see something like the following:

http://127.0.0.1:7101/ViewController/faces/employees

This is good, but ViewController isn't really the name of our application. So, in order to make the name reflect your application, let's rename it to what JDeveloper thinks is the name of our application.

6. Right-click on the ViewController project and click on Project Properties... at the end of the list; select Java EE Application.

7. Change Java EE Web Application Name and Java EE Web Context Root to something like helloadf and then run again; now, you can see your application with a nicely formatted URL like the following: http://127.0.0.1:7101/helloadf/faces/employees

You can also run the page directly from the ViewController project. It'll ask you for a target to run; choose the adfc-config. It'll also ask you for the target inside adfc-config; choose employees.

How it works...

In this recipe, we tried three different methods to run our ADF application; however, it is preferable to always use the third method; adfc-config is the heart of your ADF application and contains all of the project's navigation information. From the task flow, you can notice that there are no green circles behind the page, which means that if you have any other activity, then the new activity can also be called directly and it doesn't have to follow any order.

For More Information:
If you run your application using any one of the scenarios and this is the first time you run an application in JDeveloper's lifetime, it'll create the weblogic domain first with no managed servers and only one admin server. It'll be created under the JDEV_USER_HOME environment variable that we created. After the domain is created successfully, the Admin Server starts, and after it does, JDeveloper starts packaging the application into a WAR (Web Archive) file. You can see this deployment profile by right-clicking and navigating to ViewController | Properties | Deployment; you will find an entry that represents the WAR packaging of this project. This project is also dependent on the Model project, which means that packaging will include all the stuff created in the Model Project. After the WAR file is created successfully, JDeveloper starts deploying the WAR file into the Admin Server. If the operation is successful, it will start to access the URL of the application inside the embedded Weblogic Server to show the target.

When you view the page, you can see that each input that has bindings represents a data field from the Employees table. When you start clicking on the buttons, you can see that the row of focus is changing to the next, previous, first, or last row, and you can validate this data against the database.

For More Information:

Where to buy this book

You can buy Oracle ADF Faces Cookbook from the Packt Publishing website:

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.