ORACLE WLS APPLICATION SECURITY—IMPLEMENTING THE SUPERSTITION IN JDEVELOPER

Peter Koletzke, Quovera

Duncan Mills, Oracle

> *Security is mostly a superstition.*
> *It does not exist in nature,*
> *nor do the children of men as a whole experience it.*
> *Avoiding danger is no safer in the long run*
> *than outright exposure.*
> *Life is either a daring adventure or nothing.*
> —Helen Keller (1880–1968) *Helen Keller*

Data is the lifeblood of an organization. Decisions are made; customers and clients are served; and careers are advanced because of data collected and available to online systems. Securing data and online systems from unauthorized access is a necessary requirement in modern IT projects. Therefore, despite the implementation complexities and Helen Keller to the contrary, we must apply due diligence for implementing the superstition of Absolute Security.

The Oracle WebLogic Server (WLS) offers a standard Java Enterprise Edition runtime service that executes web application code written in technologies such as Java servlets, JavaServer Pages, and JavaServer Faces. Oracle Platform Security Services (OPSS) provides standard Java Platform, Enterprise Edition (Java EE) security features to WLS so that you can ensure the safety of your application systems and data.

This white paper explores how to set up your application to take advantage of this WLS security feature. It briefly examines the security needs of a Java EE web application. It then provides an overview of how the OPSS security features fit into the picture of web application at runtime. The main objective of these brief introductions is to get you started thinking about security needs for your applications and the featured of WLS that can fulfill those needs.

Another main focus of this white paper is to discuss the features in JDeveloper you can use to tap the OPSS facilities in WLS. Techniques you can use in JDeveloper are best explained by demonstration so this white paper devotes the last section to a demonstration in the form of a step-by-step, hands-on practice showing how to use JDeveloper to hook your application up to the OPSS features in WLS. We think that following this practice within JDeveloper will not only inform you about these techniques but give you valuable experience with the required components.

Note

This white paper is focused on WLS security features that you can tap into using JDeveloper techniques. It targets techniques in the user interface realm but you will also want to explore other realms such as the database—for example, Virtual Private Database (VPD, also called “Fine Grained Access Control”)—and the application server—for example, URL security constraints. In addition, you will also want to study any unfamiliar security topic such SQL injection, cross-site scripting, denial-of-service, and URL hacking, so that you can construct test plans that include security attacks of different types.

Application Security

Application security is an important component of system design. You need to plan carefully which users should have access to particular application functions that manage specific data sets. Coding and testing this access is part of a complete system development effort. Most applications need to ensure the following security aspects:

- Only approved users can access the application.
- Users can access only the data they are allowed to access.
- Users cannot perform actions not designed in the system (for example, accessing server files).

Web applications, especially those with Internet audiences, need a publicly-available web server that responds to HTTP requests. The application code and file privileges on this server can be limited to read-only. However, web applications handle the database connection automatically in the Model (database) layer code. As with Oracle's Fusion (and E-Business Suite) applications, a single database user account is used for all users accessing the database. This means that application user accounts must be established outside of the database. The WebLogic server provides authentication features that require users to log into an application session. The user accounts on the application server act as user accounts for a specific application. Application logic then manages specific application privileges to these users. However, this strategy requires application user set up and this may duplicate existing user lists on a network. The solution is to use a Lightweight Directory Access Protocol (LDAP) repository like Oracle Internet Directory (OID) as mentioned later in this paper.

Note

This white paper focusses on techniques and features available with the most currently available version of JDeveloper 11g—11.1.2. Techniques and features may vary somewhat with other versions of JDeveloper 11g.

Authentication and Authorization

The normal way to satisfy the requirement of allowing application access to only approved users is with a login screen where users enter credentials such as a name and password. Logging into a Java EE application accomplishes two objectives—both of which are provided by the runtime process. These stages follow:

AUTHENTICATION

The security service validates the user's credentials based upon a user name and password, or potentially a token based mechanism such as a Secure Socket Layer (SSL) certificate or biometric device such as a finger print scanner. The user name and password approach is the most familiar and common implementation of authentication. The user's name, password, and a definition of the groups to which they belong are stored in a *user repository* (also called an *identity store* or *credentials store*).

The OID component of Oracle WebLogic Server provides LDAP services but you can alternatively use other LDAP systems. As with its predecessor (Oracle Application Server), WLS supports Single Sign-On (SSO)—a facility for passing user login information between applications. It is intended as a provider for an enterprise-level, production environment. The sidebar "Testing Application Security in JDeveloper" describes how the enterprise user repository can be emulated for a local WLS runtime.

Testing Application Security in JDeveloper

It is important to work security into the application as early as possible during development so you can test all possible scenarios while completing a certain application functional area. When you test an application in JDeveloper, you run the integrated WebLogic Server locally and you will normally not tap into the enterprise LDAP server. ADF Securityallows you to set up test accounts and roles (as demonstrated in the hands-on practice later in this white paper) that you can use to try out access to application functions. When the application is deployed to the enterprise server, the test accounts will not be copied but you will need to set up the roles (if they do not exist) needed by the application

AUTHORIZATION

After passing the authentication stage to verify the user, the security service provides access to information about the user to the application. This information may take the form of a list of groups to which the user belongs in the user repository. These roles are then mapped to the *logical roles* within the application. The application's logical roles are used in the definition of rules that allow access for parts of the application. In a Java EE application, the rules and group-to-role mappings are stored in a configuration file. The user, who logged in during the authentication stage, is given access to application functions based on application roles.

In addition, when needed, the application can read the logged-in user's name and role and hide or disable restricted parts of the application appropriately.

Java Authentication and Authorization Services

It is best to rely on prebuilt frameworks when faced with the task of implementing security for an application. This, not only saves you work, but ensures that you are not missing any features required when securing a system. Fortunately, the Java

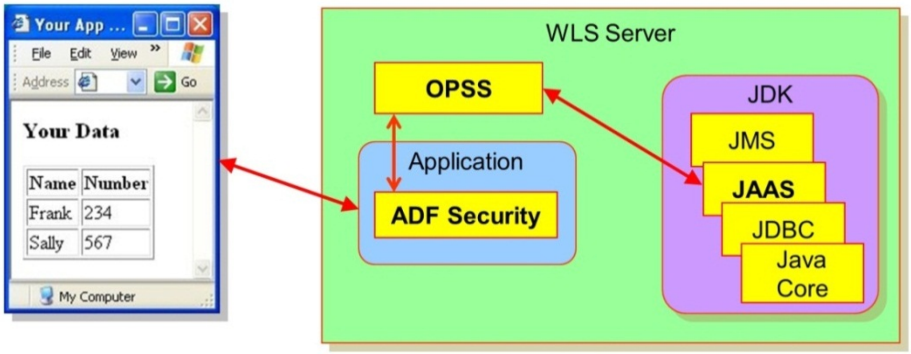
Development Kit (JDK) offers a framework for this purpose: *Java Authentication and Authorization Services (JAAS)*. JAAS offers functionality that you can use by calling its APIs to verify user logins and restrict access to resources. This library also provides an industry standard method for authentication and authorization. The JAAS features are available to application client (desktop) applications as well as web client applications.

Oracle Platform Security Services

The Oracle Plaform Security Services (OPSS) of the WebLogic Server is responsible for providing hooks from the application to JAAS facilities. According to an FAQ for OPSS, the services rendered include, "Security (authentication, authorization, SSO, credential store management, key store management); Audit; Cryptography (encryption and signature); Certificate lookup and validation; User roles; Credential mapping; Role mapping; Java EE policy and role deployment; Java2 and JAAS Policy Provider." (currently at www.oracle.com/technetwork/testcontent/opss-faq-131489.pdf)

ADF Security

Oracle Application Development Framework (ADF) is an architecture supplied by JDeveloper, which offers the developer consistent methods for working with various ADF and non-ADF frameworks (code libraries) like Enterprise JavaBeans (EJBs), ADF Business Components, and ADF Faces. ADF Security is another one of the ADF frameworks. It provides a layer on top of OPSS that makes connecting application components to secure services relatively easy. For example, after setting the application up to use ADF Security, most code that connects pages, page fragments, and elements on the page to security checks is done in a declarative way (the "ADF Way"). The following diagram depicts the relationship between ADF Security, OPSS, and JAAS.



Your Data

Name	Number
Frank	234
Sally	567

Directory Services

In addition to the security services provided by the JDK through OPSS to the application, the application server uses *directory services* software to link between the application server and an established user access control list (user repository) that is external to the application server. Other user (and multiple) repositories can be plugged into this service by use of *login modules*, which provide a standard access to other types of user lists such as Oracle database accounts.

In Oracle WLS, any LDAP server supplies this directory service. The examples that follow use LDAP services from Oracle Internet Directory, which can tap into the user list in an existing LDAP system (such as the Microsoft Active Directory). LDAP repositories are used to validate network users for file and directory access privileges outside of web (and other) applications. The communication path for this strategy is shown in Figure 1.

The process flow for the example of an application server login in this diagram follows:

1. The user sends an HTTP request using a URL including a context root that identifies a specific application (tuhra in this case). The request determines that ADF security is active in the application.
2. The authentication service determines the type of authentication and presents a login page.
3. The user enters a name and password and submits the login page.
4. The authentication service requests OID to verify the user and password.
5. OID verifies the password from the LDAP source and indicates pass or fail to the authentication service.
6. The authentication service accesses the application and places the user name into the HTTP session. The directory services pass this result to the authentication service.
7. During the application session, the application can request the username ("Joe") or group (role, in this example, "manager") to which the user belongs so it can render the pages accordingly.

8. web.xml activates ADF Security for authorization to specific resources like pages and task flows.
9. The application connects to the database using the application database user account (APPUSER) written into a data source on the application server.

> **Note**
>
> The starting point for documentation about WLS security features is the Oracle Fusion Middleware security documentation library currently at docs.oracle.com/cd/E21764_01/im.htm (specifically *Oracle Fusion Middleware Application Security Guide 11g)*

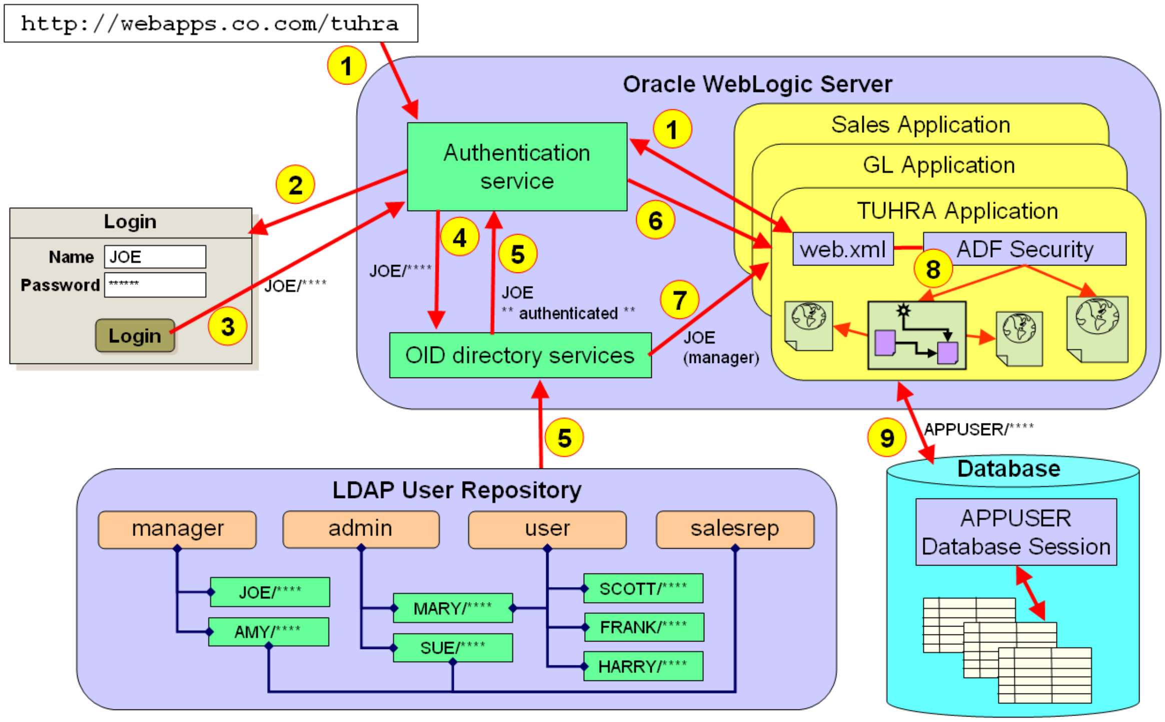


Figure 1. Directory services used for a Java EE web application

Levels of Security in a Web Application

Due to the highly-accessible nature of the World Wide Web, web applications are potentially more widely available than intranet or WAN applications. Therefore, an approach that addresses multiple layers is necessary as follows:

- **Database user** All web application users connect to the database using a single database user account. This application database user account would be different from the application database object owner account. It would be granted access to only the required application objects.
- **Application user account** Just as database grants must be in place so the application user account can access the application owner account's objects, the application needs to set and interrogate privileges when presenting menu options, pages, or components on a page.
- **Application user data access** Access to pages and components can provide security at the table level. However, this level may not be sufficient. Your application may also require restriction to specific rows within a table. You can accomplish this by adding WHERE clause components that read the database user or by using table policies (VPD).
- **Data query restrictions** Query-By-Example (Find mode) screens may allow the user the ability to query data in an unintended way using SQL injection. The ADF Business Components *model-based query criteria* object implements a secure method for user queries that is based on bind variables and is immune to SQL injection.

Groups and Roles

A user repository enrolls users in groups so privileges can be granted by group, not by individual users, who may change job functions over time. Groups reflect the job functions of users, for example, HR Manager or Warehouse Clerk.

The application defines roles, a set of functional privileges to which application areas are granted. For example, a role called Expense Report Approver might be given the ability to approve or reject expense reports. Other roles would not be able to perform that function. The application server would contain a set of these roles (called "groups" on the application server side) and the application would map the application server's role-groups to the application's functional groups. The enrollment of a user in a role-group would be maintained on the application server's directory services (user repository), but the mapping of the user repository roles to application roles would be part of the application.

Security Policy

In addition to mapping application server groups to application roles, the application also defines *security policies*, definitions for privileges. A security policy consists of *principles*—the role or roles who will be allowed to perform the function, *resources*—the application functions to which the policy allows access, and *permissions*—the actions that can be performed by the users in the granted roles. These three components of a security policy are very similar to the definition elements in a database grant as shown in the following table:

Security Policy Component	Database Grant Element
Principal	Database user or role granted the privilege.
Resource	Database object such as a table, view, PL/SQL program unit to which the grant applies.
Permission	The operation allowed to the table, view, or PL/SQL, for example, INSERT or EXECUTE.

Summary

This brief introduction to application security has discussed the basic security services of WLS and how they can be defined to read from user repositories in two formats: XML and LDAP. It explained the basic steps for login: authorization and authentication. It also outlined how various server and client components interact in the security process when a web application is run as well as some of the terminology used when setting up and using security services. With this background along with the techniques demonstrated in the hands-on practice that follows, you will be able to design and program security features into your ADF application so your data and systems can be as safe as possible and, therefore, implement the elusive target of Absolute Security.

> *We will bankrupt ourselves*
> *in the vain search for absolute security.*
>
> —Dwight David Eisenhower*, (1890–1969)*

About the Authors

Duncan Mills is senior architect for Oracle's ADF Product Development group. He has been working with Oracle in a variety of application development and DBA roles since 1988. For the past 16 years he has been working at Oracle in both customer support and product development, spending the last 10 years in product management for the development tools platform. Duncan is a frequent presenter at industry events and has many publications to his credit including coauthorship of the Oracle Press books: *Oracle JDeveloper 11*g *Handbook* and *Oracle JDeveloper 10*g *for Forms and PL/SQL Developers*.

Peter Koletzke is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has 28 years of industry experience. Peter has presented at various Oracle users group conferences more than 300 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice (twice), ODTUG Best Speaker, ECO/SEOUC Oracle Designer Award, ODTUG Volunteer of the Year, and NYOUG Editor's Choice (three times). He is an Oracle Certified Master, Oracle ACE Director, and coauthor (variously with Dr. Paul Dorsey, Avrom Roy-Faderman, and Duncan Mills) of eight Oracle Press development tools books including *Oracle JDeveloper 11g Handbook* (from which some of the material in this white paper is taken).

Hands-On Practice: Implement Application Security in JDeveloper

This hands-on practice builds on the introduction in the body of this white paper and shows how to add security features to an existing application. The steps in this practice assume you have already created an application and are applying security hooks to it. The practice steps refer to the second version of an application —The Ultimate Human Resources Application (TUHRA or TUHRA2)—developed for the authors' book *Oracle JDeveloper 11*g *Handbook*. Although the practice refers to specific features and files in this application, you can use an application of your own and adjust appropriately when a specific page or feature is referenced.

Should you want to follow the steps exactly as written, you can start with the same sample application referred to in this section. Start by pointing your web browser to tuhra2.java.net. Then click Downloads and download the file for Chapter 20, part 5 (C20-V.zip). Unzip it into a new folder and open tuhra2.jws in JDeveloper (you will need to run through the conversion wizard the first time). This sample application will serve as a starting point for the following hands-on practice. You may also download files representing the end of each phase of this hands-on practice from the same website.

Note

This practice was tested with JDeveloper 11*g*, release 11.1.2.0. If you use an earlier version of JDeveloper 11*g*, you may need to adjust steps for editing jazn-data.xml. However, the principles and major steps should remain the same.

As shown in Figure 2, authorization for the TUHRA application is divided into four logical roles (represented by arrowed boxes): non-authenticated users (Public), the general user role (authenticated users without any specific privileges), the manager role, and the admin role. The application contains no hard-coded user names or group names; instead, these logical roles will be mapped to user and group names at runtime using WebLogic internal LDAP.

This hands-on practice sets up security for the sample application, creates a login page, and restricts access to task flows, data, and UI elements based on authorization status. The practice steps through the following phases:

I. Set up security for the application

- Enable ADF security
- Define application roles
- Define a credential store for testing
- Map application roles to credential store roles

II. Implement authorization

- Authorize access to pages
- Create basic management functionality
- Authorize access to task flows
- Authorize access to UI components

III. Implement custom login functionality

- Implement a managed bean to handle login
- Create the login popup
- Create menu options to log in and log out
- Implement a method to handle logout

IV. Access information about the logged-in user

- Create a view object definition to return user information
- Display user information
- Allow a user to edit his/her own records

Tip

Download the sample solution for the end of Chapter 21 from tuhra2.java.net in case you want to compare working code with your code at any point.

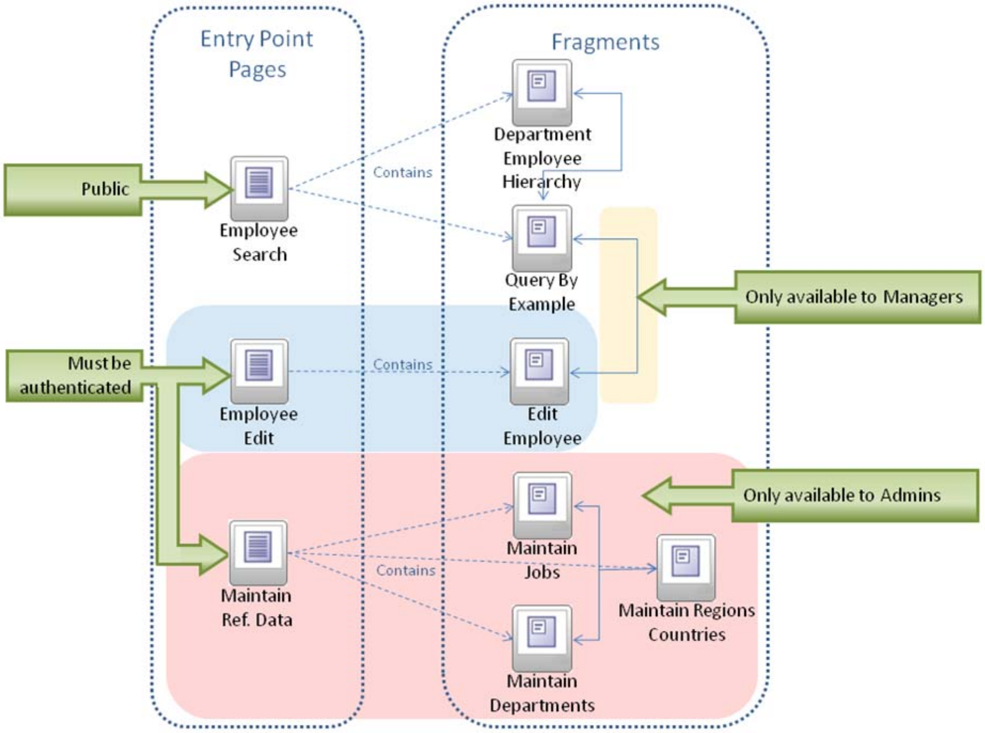


Figure 2. Access in the TUHRA application

Tip

We recommend always implementing authorization for applications in terms of logical roles and using an identity store such as WebLogic internal LDAP or Oracle Internet Directory (OID) for runtime mapping of users to roles.

I. Set Up Security for the Application

Before you can implement the specifics of authorization for the application, you must perform some setup. In this phase, you will enable ADF security and prepare a simple, XML-based identity store for testing purposes.

Enable ADF Security

In this section, you will configure options for security in the TUHRA2 application.

1. With the TUHRA workspace open, select **Application | Secure | Configure ADF Security** to open the Configure ADF Security wizard.

2. On the first page, leave the default selection ("ADF Authentication and Authorization").

3. Click Next. On the "Select authentication type" page, select "Form-Based Authentication" and "Generate Default Pages."

   Additional Information: The various *Authentication Type* options correspond to different ways of authenticating the user:

   - **HTTP Basic Authentication** uses the browser's login functionality to send a user name and password in plain text across HTTP. This is not secure by modern standards but may be required to support certain very old browsers. However, given that ADF Faces only supports the more recent versions of the browsers, this option should never be used for an ADF page.

- **HTTP Digest Authentication** uses the browser's login functionality to send a user name and encrypted password across HTTP. The server does not decrypt the sent password; rather, the server executes the encryption process on the user's real password and compares the results.
- **HTTPS Client Authentication** issues an SSL certificate to the browser and requires it for future authentication.
- **Form-Based Authentication** uses an HTML form, rather than browser functionality, to request user credentials. In a production environment, this type of authentication should only be used when the application is running in HTTPS mode.

For more information about authentication types and their advantages and disadvantages, you can refer to books about application security, for example, *Core Security Patterns* by Christopher Steel, Ramesh Nagappan, and Ray Lai (Prentice Hall, 2005) and *Enterprise Java Security* by Marco Pistoia, Nataraj Nagaratnam, Larry Koved, and Anthony Nadalin (Addison-Wesley, 2004).

4. Click Next. On the "Enable automatic policy grants" page, leave the default (No Automatic Grants) and click Next.

   Additional Information: The other options automatically create a special testing role called "test-all," with access to all pages and task flows in the application. While this can be useful for testing, you will set up your own security roles in the next section, so it is not necessary to create the test-role now.

5. On the "Specify authenticated welcome page" page, select *Redirect Upon Successful Authentication* and set the welcome page to "/employeeSearch.jspx."

   Additional Information: Upon successful authentication, this will send a browser redirect to reroute users to the search page.

6. Click Finish. When the Security Infrastructure Created dialog appears, click OK.

7. Click Save All.

   Additional Information: JDeveloper creates several files to implement security for the application:

   - **login.html** is the default login page created in step 3.
   - **error.html** is the default login error page created in step 3.
   - **jazn-data.xml** will store the security rules or permissions that you will create for the application, as well as providing a simple credential store for testing purposes. This file can be found in the Application Resources section of the Application Navigator, under Descriptors\META-INF.

   JDeveloper also updates the following files, which already exist:

   - **web.xml** Various security settings such as the selected authentication type and the name of the login and error pages have been added to this file. To view the changes, open the web.xml file and click the Security tab of the Overview editor.
   - **weblogic.xml** Recall that you had already defined this file to control some of the WebLogic behavior with respect to handling files for the image upload. If you look at this file now, you will see that a new section called "security-role-assignment" has been created. This section is used to map users (also called "principals") to roles in WebLogic. In this case it is used to map a generic principal called "user" to the WebLogic role "valid-users."

     If you cross-reference with the web.xml file at this point, you will see the security role "valid-users" mentioned there as well; it has been automatically defined for you as a logical-application level role.
   - **jps-config.xml** This file defines the high-level configuration of ADF Security. *JPS* in this context stands for "Java Platform Security." This file controls, among other things, where the credential store used for security is located. This file can be found in the Application Resources section of the Application Navigator, under Descriptors\META-INF.
   - **adf-config.xml** This is the general ADF configuration file (do not confuse it with the unbounded task flow definition file adfc-config.xml). The changes made here just indicate to ADF that security is now enabled and handled by JPS. This file can be found in the Application Resources section of the Application Navigator, under Descriptors\ADF META-INF.

Define Application Roles

In this section, you will define the logical security roles for the application. As noted in the introduction to this section, these roles are distinct from users and groups in your credential store. They are mapped to the store at runtime, allowing for easy change of permissions.

1. In the Application Resources panel of the Navigator, open jazn-data.xml (Descriptors\META-INF node), and select the Overview editor tab if it is not already selected.
2. Select the "Application Roles" sub-tab on the left.
3. Click the Add (green plus "+") button in the header of the "Roles" area and select **Add New Role**. In the fields on the rightl set *Name* as "user."
4. Click Add again and select **Add New Role | Add at Root Level**. Set *Name* as "manager."
5. Repeat step 4 to add a role named "admin."
6. Click Save All.

Define a Credential Store for Testing

In addition to storing logical roles for an application, jazn-data.xml can be used as a simple credential store, including actual users and/or groups. When an application is deployed in a production environment, we do not recommend using jazn-data.xml as a credential store; rather, the roles within your application should be mapped to roles provided by an external credential store (such as an LDAP server) using runtime mappings created in the application's Java EE container. However, we do recommend adding sample users and groups to jazn-data.xml for testing purposes within JDeveloper, and that is what you will do in this section.

1. In the editor for jazn-data.xml, click the Enterprise Roles tab.

   Additional Information: In the top right corner of the jazn-data.xml window, you can see that jazn.com is the default realm, or identity namespace, used by Oracle. Roles created on this panel are not logical application roles, but rather the equivalent of roles or groups in the credential store.

2. Click New Role (green plus "+") in the header of the "Enterprise Roles" box to open a role. Set *Name* as "cs_admin" (for "credential store administrator role").

   Additional Information: You could use identical role names for both the logical and credential store roles; however, you are using different role names here to help you to understand the mapping between the two.

3. Repeat step 2 to create roles named "cs_manager" and "cs_user."
4. Select the "Users" tab.
5. Click New User (green plus "+") and set the following properties for the user:
   Name as "TFOX"
   Credentials as "welcome2tuhra"

   Additional Information: WebLogic requires passwords of at least eight characters including at least one number.

6. Repeat step 5 to create users named "NKOCHHAR" and "CDAVIES," both with the password "welcome2tuhra."
7. Select the TFOX user. Click the Assign Roles (green plus "+") button in the header of the "Assigned Roles" box and select **Assign Enterprise Role**. Check the checkbox next to the cs_admin role and click OK.
8. Repeat step 7 to add NKOCHHAR to the cs_manager role and CDAVIES to the cs_user role.
9. Click Save All.
10. Open the Source tab and examine the code.

    Additional Information: The credential store is inside the `jazn-realm` element. A portion of the code is reproduced next. You may also see a `guid` element in your file, which is a unique global id for the user that WebLogic will automatically generate.

```
<jazn-realm default="jazn.com">
  <realm>
    <name>jazn.com</name>
    <users>
      <user>
        <name>TFOX</name>
        <credentials>{903}LJHErT4qy...WYKpkC0CpY=</credentials>
      </user>
    </users>
    <roles>
      <role>
        <name>cs_admin</name>
        <members>
          <member>
            <type>user</type>
             <name>TFOX</name>
          </member>
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>
```

Users, with their userid and encrypted credentials, are defined in `user` elements; roles are defined in `role` elements, with references to member users and member roles inside `member` elements.

Map Application Roles to Credential Store Roles

Now that you have created both application roles and a set of credential store roles for testing, you need to create a mapping between them. In a production application, the credential store is likely to be in a repository that is separate from the jazn-data.xml file, for example, the WebLogic LDAP repository or OID. However, for testing purposes, the same jazn-data.xml file can be used for storing both your credential store roles and application roles. Recall that you deliberately used different names for these so as to illustrate the mapping that is required.

1. Return to the Overview editor tab in jazn-data.xml.
2. Click the Application Roles tab.
3. Select the "user" role in the *Roles* list.
4. In the Mappings tab (in the lower right), click Add User or Role (green plus "+") and select **Add Enterprise Role**. Select the checkbox in the heading of the first column to select all three roles and click OK.

   Additional Information: Any user with one of these roles in the credential store will automatically have the "user" application role.
5. Repeat step 4 to make both cs_manager and cs_admin member roles of manager.
6. Repeat step 4 to make cs_admin a member role of admin.
7. Click Save All.

What Did You Just Do?

You set up the jazn-data.xml file to hold the following application metadata:

- **Application roles** Collections of application privileges
- **A sample credential store** Users and roles for testing purposes
- **Mappings** Mappings between application roles and roles in the credential store

II. Implement Authorization

When you first enable ADF security on an application, no users have any permissions on the application at all. In this phase, you will re-open access to various parts of the application based on user roles.

Authorize Access to Pages

In this section, you will open the search page to all users, including those who are not yet logged in, and will also open the edit page to all logged-in users.

1. Terminate the integrated WebLogic server, if it is running.

2. Run the employeeSearch.jspx page.
3. When the default login page appears, enter the following information and click Submit:
   Username as “NKOCHHAR”
   Password as “welcome2tuhra”

   Additional Information: An HTTP 500 error is displayed:

   ```
   oracle.adf.controller.security.AuthorizationException: ADFC-0619:
   Authorization check failed: '/employeeSearch.jspx' 'VIEW'.
   ```

   This is because you have not yet granted permission on the page to the public or to any group of which NKOCHHAR is a member.
4. Close the browser and stop the application.

> **Tip**
>
> When developing and testing security, it is always a good idea to close the browser after every run. This is because, if the browser is not entirely shut down (including all windows and any other applications that share session cookies with the browser), it may still regard the session as authenticated. In addition, if you have changed the credential store or permissions, you will need to re-deploy the application (by stopping and restarting it in JDeveloper) to copy the new values across. You rarely need to restart the integrated server, however.

5. Re-open the Overview tab of jazn-data.xml, if it is not already open.
6. Select the “Resource Grants” tab on the left and select “Web Page” under Resource type.
   Additional Information: You can declaratively authorize access to application resources in two different ways: by authorizing access to a particular ADF page definition file (on this tab), or by authorizing access to a bounded task flow (on the “Task Flows” resource type). Authorizing access to a particular ADF page definition file allows access to whatever resource—a page, page fragment, method call activity, or template—uses that page definition.

   In general, the best practice is to authorize access to task flows rather than page definition files—this allows the same data-bound resource to be used in multiple task flows, with different authorization in each. However, this is not an option for resources embedded directly in the application’s unbounded task flow, such as employeeSearch.jspx, so you must define the authorization at the page level in this case.

   You will authorize resource access using the task flow method in the next section.
7. Select the employeeSearch page definition.
8. Click the “Add Grantee” (green plus “+”) button in the *Granted To* column and select **Add Application Role**.
9. In the Select Application Roles dialog, select anonymous-role as shown in Figure 3, and click OK.

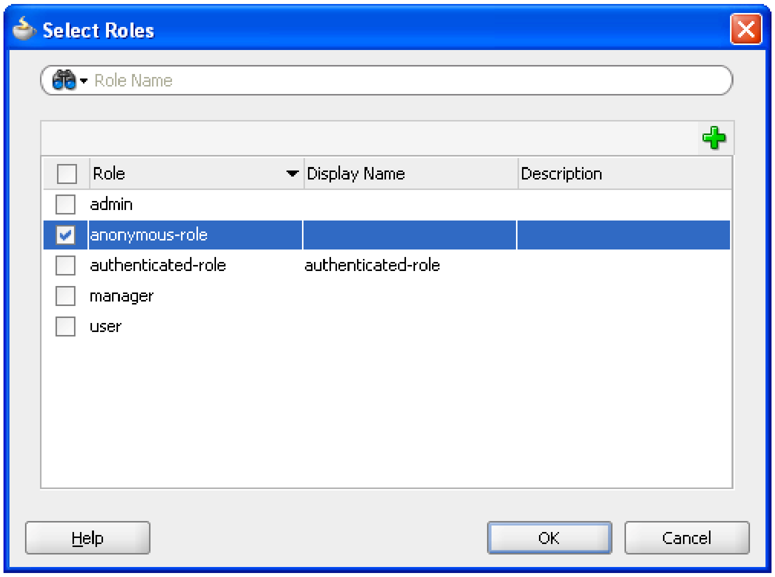
   Additional Information: In addition to the three application roles you defined in the phase “Set Up Security for the Application,” you will see two additional roles, anonymous-role and authenticated-role, as shown in Figure 3.

   These roles exist in any ADF Security-enabled application. Resources granted to anonymous-role can be accessed by any user, even a user who is not logged in. Resources granted to authenticated-role can be accessed by any logged-in user.

   Also, note that the right-most pane, as shown in Figure 4, contains a list entitled “actions”; for more information, see the sidebar “Authorized Actions.”
10. Repeat steps 7–9 to grant access to employeeEdit page def to the “user” application role.
11. Click Save All.
12. Run employeeSearch.jspx.

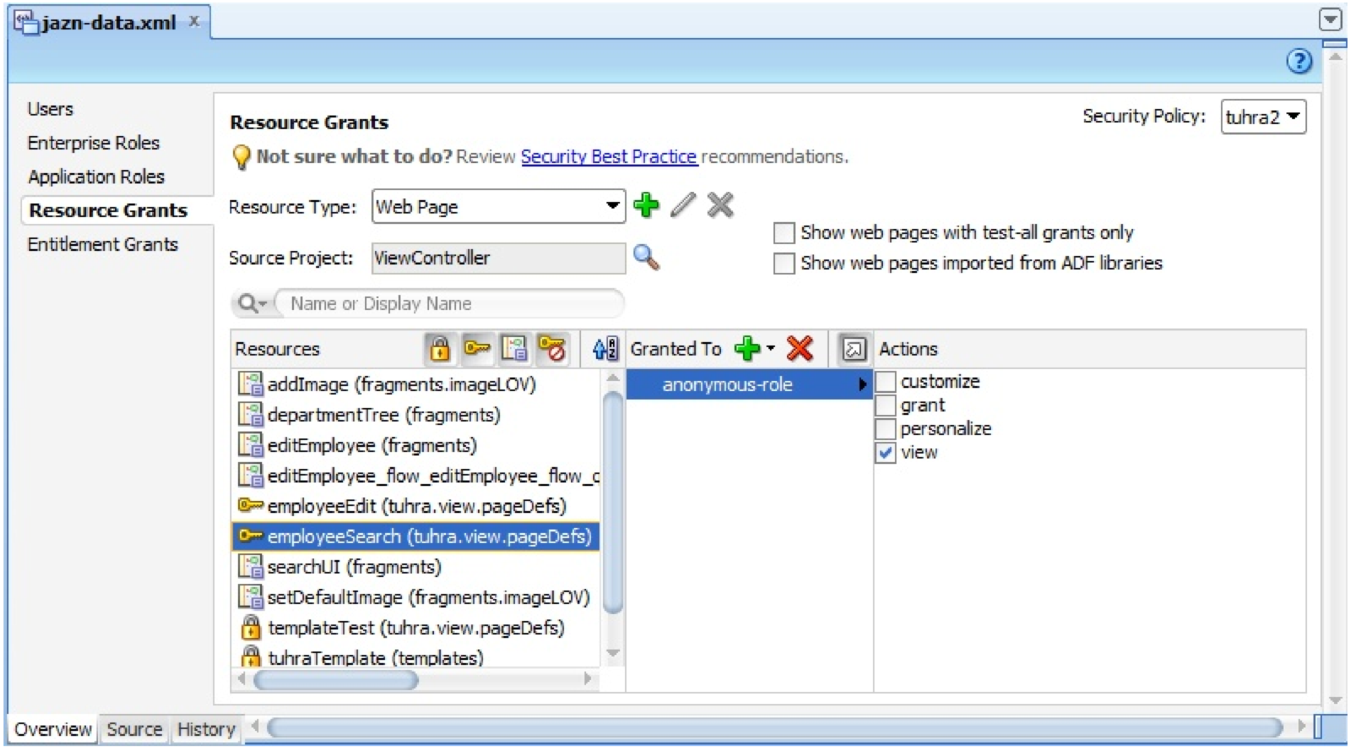
Additional Information: When the browser opens, you will see only the template, the panelSplitter, and the images. However, you will not see the expected fragment in the right-hand pane, because you have not yet authorized access to the bounded task flow searchEmployees-flow that should be running there. You will authorize access to searchEmployees-flow in the section “Authorize Access to Task Flows.”

13. Close the browser and stop the application.



	Role	Display Name	Description
□	admin		
☑	anonymous-role		
□	authenticated-role	authenticated-role	
□	manager		
□	user		

Figure 3. Selecting the anonymous-role for the employeeSearch page



Security Policy: tuhra2

Resource Type: Web Page

Source Project: ViewController

□ Show web pages with test-all grants only

□ Show web pages imported from ADF libraries

Resources:

- addImage (fragments.imageLOV)
- departmentTree (fragments)
- editEmployee (fragments)
- editEmployee_flow_editEmployee_flow_c
- employeeEdit (tuhra.view.pageDefs)
- employeeSearch (tuhra.view.pageDefs)
- searchUI (fragments)
- setDefaultImage (fragments.imageLOV)
- templateTest (tuhra.view.pageDefs)
- tuhraTemplate (templates)

Granted To: anonymous-role

Actions:

□ customize
□ grant
□ personalize
☑ view

Figure 4. The anonymous-role defined for the employeeSearch page

Authorized Actions

When you specify that a particular application role is authorized to access a page definition or Task Flow, you can specify up to four actions that role is authorized to perform on the page or the contents of the Task Flow:

- **Customize** Allows the role to perform runtime customizations to the page, changing its contents for all viewers, as a portal administrator might do to a portal.
- **Grant** Allows the role to create authorizations for other roles to access the page or Task Flow at runtime.
- **Personalize** Allows the role to perform runtime personalization of the page, changing its contents for the logged-in user only, as a portal user might do to a user-customizable portal.
- **View** Selected by default, allows the role to access the page or Task Flow activities as designed by the developer. Note that this does not imply that the object is somehow read-only. The user can interact with the data on the page to make updates if the application allows that.

Runtime customization, grants, and personalization are a feature exposed by applications using Oracle WebCenter capabilities and beyond the scope of this white paper.

Create Basic Management Functionality

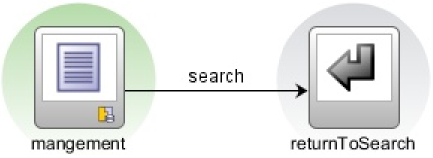
In this section, you will create a page, management.jspx. A complete implementation of the management.jspx page is included in the sample application on the samplecode website. However, because the purpose of this practice is to master security techniques not develop JSF code, you will leave the page blank. You are creating this page so that, in the following section, you can see a technique for authorizing access to top-level pages other than by creating authorizations for their page definitions.

1. Create a bounded task flow (in the ViewController project, **File | New**, JSF/Facelets, ADF Task Flow), management-flow.xml, deselecting *Create with Page Fragments*.

   Additional Information: Unlike the other bounded task flows in the application, which contain fragments and are used inside regions, management-flow.xml will contain only a single, top-level page and will be accessed from the unbounded task flow via a control flow case.

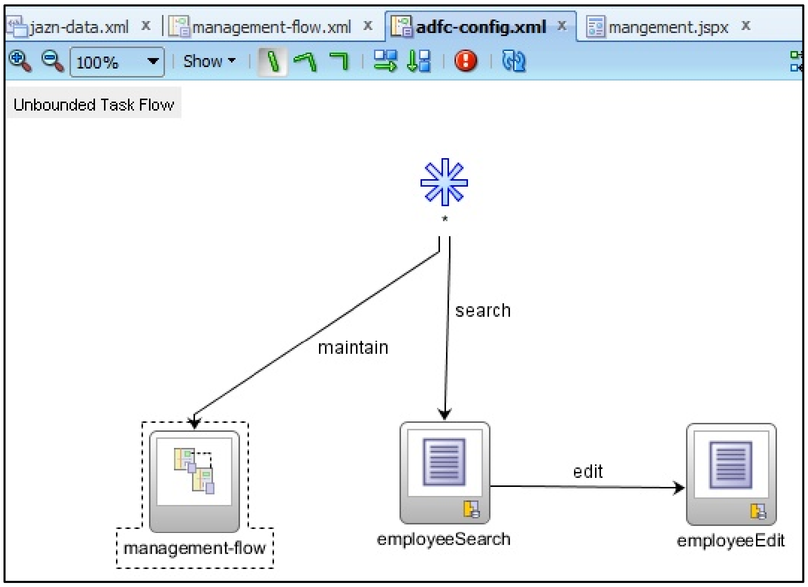
2. Drag a view activity from the Component Palette onto the flow.
3. Name the activity “management.”
4. Double-click the activity to open the Create JSF Page dialog.
5. Select “JSP/XML.” Base the page on the tuhraTemplate template and click OK.
6. Return to the task flow and drag a task flow return from the Component Palette to the diagram, and name it “returnToSearch.”
7. Create a control flow case named “search” from the management view activity to the task flow return activity.
8. Select the returnToSearch activity and set its *Outcome: Name* property to “search.”

   Additional Information: When the task flow returns, it will follow the wildcard control flow case in the unbounded task flow to return to employeeSearch.jspx. The task flow should appear something like the following illustration:



9. Click Save All.
10. Open the application’s unbounded task flow, adfc-config.xml (under Web Content\WEB-INF).
11. Drag management-flow from the Application Navigator onto the task flow diagram to create a task flow call activity.

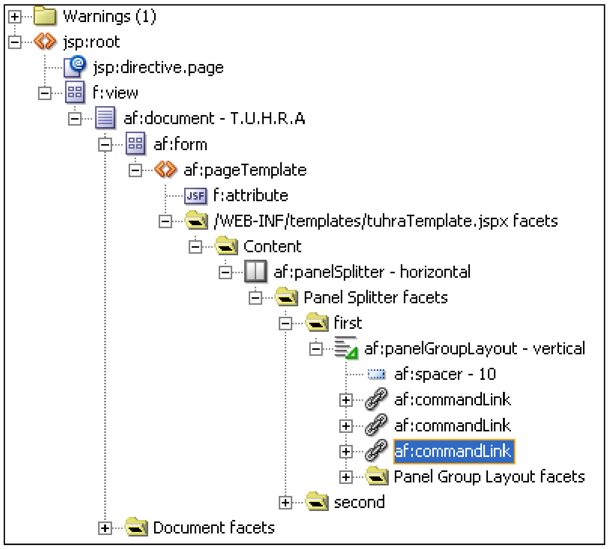
12. Create a control flow case from the wildcard control flow rule to the task flow call activity, named “maintain.” The task flow diagram should appear as shown here:



13. Open employeeSearch.jspx in the editor.

14. Select the second command link *(View By Department)* in the “first” facet of the `af:panelSplitter`.

15. Using the Structure window, copy the link, and paste a copy immediately after it. The Structure window should look like the following:



```
Warnings (1)
jsp:root
  jsp:directive.page
  f:view
    af:document - T.U.H.R.A
      af:form
        af:pageTemplate
          f:attribute
          /WEB-INF/templates/tuhraTemplate.jspx facets
            Content
              af:panelSplitter - horizontal
                Panel Splitter facets
                  first
                    af:panelGroupLayout - vertical
                      af:spacer - 10
                      af:commandLink
                      af:commandLink
                      af:commandLink
                      Panel Group Layout facets
                  second
      Document facets
```

16. Set the following properties on the new command link:
    Id as “maintLink”
    Action as “maintain”
    PartialSubmit as “false”

17. Expand the command link and `af:panelGroupLayout` inside it and select `af:image`.

18. Set the following properties on the image:
    Source as “images/maintain.png”
    ShortDesc as “Maintenance”

19. Select the `af:outputLabel`.

20. Set the following properties on `af:outputLabel`:
    For as “maintLink”
    Value as “Maintenance”

21. Delete the `af:setPropertyListener`.

22. Click Save All.

Authorize Access to Task Flows

Although you have granted access to the application’s top-level pages, you also need to allow access to the task flows that run in regions of those pages. In addition, you need to allow access to the maintenance page you created in the previous section, which you can do by allowing access to its bounded task flow. In this section, you will implement authorization for the bounded task flows.

1. In the jazn-data.xml file’s editor Overview tab, select the Resource Grants tab.

2. Select *Resource Type* as “Task Flow” and select searchEmployees-flow.

3. Click the “Add Grantee” (green plus “+”) button in the *Granted To* column and select **Add Application Role**.

4. In the Select Application Roles dialog, select anonymous-role and click OK.

5. Repeat steps 3–4 to authorize anonymous-role to access the task flow departmentTree-flow.

6. Repeat steps 3–4 to authorize the “user” role to access editEmployee-flow and imageLOV-flow.

7. Repeat steps 3–4 to authorize the “admin” role to access management-flow.

   Additional Information: Instead of putting the management view activity directly in the application’s unbounded task flow, in the preceding section you created a bounded task flow to hold it. This allows you to authorize access to everything inside the flow in one operation, rather than having to control access to each activity within the flow individually.

8. Click Save All.

9. Run employeeSearch.jspx.

   Additional Information: This time, the expected UI appears, because anonymous users are authorized to access both the page and the task flows running in the “second” facet of the panelSplitter.

10. Perform a search, select an employee in the results table, and select **Edit Employee** from the right click menu on the table.

    Additional Information: The default login page appears. This is because, although employeeSearch access was granted to anonymous-role, employeeEdit access is restricted to the user role.

11. Log in with the credentials “CDAVIES”/“welcome2tuhra.”

    Additional Information: The Edit Employee page appears, because CDAVIES has the “user” role, and that role has been authorized for the Edit Employee page and all of its internal task flows.

12. Click the “Return to Employee Search” link to return to the search page.

13. Click the Maintenance link.

    Additional Information: A page containing a 401 – Unauthorized error will appear. This is because CDAVIES does not have the “admin” application role, which is required for access to the management task flow.

14. Close the browser and stop the application.

15. Repeat steps 9–13, this time logging in using the credentials “TFOX”/“welcome2tuhra.”

    Additional Information: This time, when you click the Maintenance link, you will see the (empty except for the template components) Maintenance page. This is because TFOX has the required “admin” application role.

16. Close the browser and stop the application.

Authorize Access to UI Components

When you attempted to access the Maintenance page when logged in as CDAVIES, you received an unfriendly error page. It would be better if the Maintenance link did not appear at all for users who did not have the required application role. In this section, you will hide that link from non-administrative users. You will also hide the command components that link to the Edit Employee page from anyone who is not a manager.

The reason you are hiding those components, instead of restricting access to the Edit Employee page to managers, is that users with only the "user" role will, eventually, need to use the Edit Employee page to edit their own record. You will implement this in the section "Allow a User to Edit His/Her Own Records." What an ordinary user will not be able to do is edit arbitrary records using command components on search results.

1. Re-open the employeeSearch.jspx page in an editor.
2. Select the maintLink command link.
3. Set the *Rendered* property of the link to "#{securityContext.taskflowViewable['/WEB-INF/management-flow.xml#management-flow']}."

   > **Note**
   >
   > You'll need to use the Expression Builder to set the property to a value other than "true," "false," or "default."

   Additional Information: Here, securityContext is a JSF managed bean that ADF Security introduces at runtime to keep track of security information. Its *taskflowViewable* property is a `Map` from task flows to Boolean values, containing a value of `true` for task flows that the current authenticated user (or anonymous-role, for non-logged-in users) is allowed to access. The actual value that you test for in the taskflowViewable map is a compound value created from the name and location of the XML file containing the task flow— "/WEB-INF/management-flow.xml," followed by the name of the task flow in that file— " management-flow." These values are separated by "#."

   You could use the much simpler `#{securityContext.userInRole['admin']}`, which evaluates to true if the current user is in the "admin" application role, but that effectively creates a duplicate reference to the "admin" role: The management-flow task flow is restricted to members of that role, and the role name would also be hard-coded in the page. If you were later to decide to change authorization for management-flow, you would have to change both of these references. By using the more complex EL expression, you will cause ADF to render the link for anyone authorized to access its destination, whatever role or roles that may translate into in the future.
4. Run employeeSearch.jspx.

   Additional Information: The Maintenance link is not displayed, since the user is not yet authenticated, and management-flow is not viewable by anonymous-role.
5. Edit an employee's record (to force a login), log in as CDAVIES, and return to the search page.

   Additional Information: The Maintenance link is still not displayed, since CDAVIES does not have the "admin" application role, which is required to view management-flow.
6. Close the browser.
7. Repeat steps 4–5, this time logging in as TFOX.

   Additional Information: This time, the Maintenance link is displayed.
8. Close the browser and stop the application.
9. Open the searchUI.jsff page fragment in a visual editor (under Web Content\fragments).
10. In the Structure window, expand `af:panelStretchLayout` and its center facet to locate the `af:panelCollection` that surrounds the results table. In the menus facet of `af:panelCollection`, expand the Actions menu and select the Edit Employee command menu item.
11. Set the *Rendered* property on the item to "#{securityContext.userInRole['manager']}."

    Additional Information: This uses the alternative way of checking who the user is as we mentioned earlier—in this case, checking against an absolute role name rather than against a permission to do something. Best practice is generally to use the taskflowViewable approach, but here this is not possible, because members of only the “user” role can access the control flow; they just cannot access it from this menu.

12. Expand the `af:table` inside the `af:panelCollection` and locate `af:popup` inside the table contextMenu facet.
13. Select the Edit Employee menu item nested under the popup’s menu.
14. Set the *Rendered* property on the item to “#{securityContext.userInRole['manager']}.”
15. Click Save All and run employeeSearch.jspx. Notice now that all of the edit links in the table context menu and the table’s Actions menu have disappeared, and there is no way to make them appear. You will fix this problem in the following phase.
16. Close the browser and stop the application.

What Did You Just Do?

You implemented authorization for the application, exposing the search page to the public, restricting access to the Edit Employee page to authenticated users with the “user” role, and restricting access to the Management page to authenticated users with the “admin” role.

Then, you hid the UI components that link to the Edit Employee and Management pages from users not authorized to access those pages. Unfortunately, this has the effect of rendering the pages completely inaccessible. By default, ADF only displays a login page when a non-authenticated user tries to access a resource requiring authentication. Because you hid the links to these resources from non-authenticated users, there is now no way to trigger the display of the login page, so no way for the user to authenticate. You will fix this problem in the following phase by allowing any unauthenticated user to request authentication at any time.

III. Implement Custom Login Functionality

The default login form functionality provided by ADF is not appropriate for the TUHRA application. First, the default form does not have a look and feel consistent with the rest of the application. Also, and even more important, the default functionality does not provide any way for a user to trigger an authentication request, since all links to resources requiring authentication are hidden from users who are not already authenticated. In this phase, you will create a login popup that is better integrated into the application’s appearance and can be accessed by any unauthenticated user on demand.

Tip

You can copy code shown in this phase from the sample solution for Chapter 21 by downloading it from the website mentioned at the start of this hands-on practice.

Implement a Managed Bean to Handle Login

In this section, you will create a managed bean that will handle login attempts in a manner consistent with application behavior. For example, it will provide an in-context message to a user upon failed login, rather than relying on browser behavior to handle authentication errors.

1. In the ViewController project, create a Java class, `tuhra.controller.LoginHandler`. (**File | New**, General\Java, Class)
2. Create two private fields:

```
private String _username;
private String _password;
```

3. Generate or create public accessors for both fields. (Select **Generate Accessors** from the right-click menu in the editor.)
4. Add the following method to the class (without the line numbers):

```
public String performLogin() {
  byte[] pw = _password.getBytes();
  FacesContext ctx = FacesContext.getCurrentInstance();
  HttpServletRequest request =
    (HttpServletRequest)ctx.getExternalContext().getRequest();
  CallbackHandler handler = new SimpleCallbackHandler(_username, pw);
  try {
```

```
08:     Subject mySubject = Authentication.login(handler);
09:     ServletAuthentication.runAs(mySubject, request);
10:     String loginUrl = "/adfAuthentication?success_url=/faces" +
11:       ctx.getViewRoot().getViewId();
12:     HttpServletResponse response =
13:       (HttpServletResponse)ctx.getExternalContext().getResponse();
14:     sendForward(request, response, loginUrl);
15:   } catch (FailedLoginException fle) {
16:     FacesMessage msg = new FacesMessage(FacesMessage.SEVERITY_ERROR,
17:                                         "Incorrect Username or Password",
18:                                         "An incorrect Username or Password" +
19:                                         " was specified");
20:     ctx.addMessage(null, msg);
21:   } catch (LoginException le) {
22:     reportUnexpectedLoginError("LoginException", le);
23:   }
24:   return null;
25: }
```

Additional Information: This is the method that will be called as an action when a user attempts to log in. The method performs the following tasks:

- **Lines 04–05** get an object encapsulating the HTTP request from FacesContext.
- **Line 06** creates a CallbackHandler, which is an object that retrieves information for security operations. A SimpleCallbackHandler allows security operations to retrieve the user name and password that were passed to its constructor; other CallbackHandler implementations can obtain the user name and password from another source.
- **Line 08** creates a Subject, which is an object that encapsulates credentials, from the information provided by CallbackHandler.
- **Line 09** attempts to log in the user issuing the request using the credentials that are encapsulated by Subject.
- **Lines 10–11** construct a URL to which to forward the user. The getViewId() method returns the path, from the context root, to the currently rendered page. So, if the line executes when the user is on the employeeSearch.jspx page, forwardURL will be set to “/adfAuthentication?success_url=/faces/employeesSearch.jspx.”
- **Lines 12–13** retrieve an object that encapsulates the HTTP response from FacesContext.
- **Line 14** calls a method, sendForward(), which you will implement later in this section to forward the user to the URL specified in Lines 11–12.
- **Lines 15–20** handle a FailedLoginException, which is the exception thrown when the credentials supplied are incorrect. The lines handle the exception by adding a new message to FacesContext.
- **Lines 21–22** handles a LoginException, which can be thrown by many different problems with a login, including not only incorrect credentials but also attempts to log in to a locked account or uses of an expired password. FailedLoginException is a subclass of LoginException, but since a FailedLoginException will be caught by Line 12, these lines will only be executed when there are login problems other than incorrect credentials. reportUnexpectedLoginError() is a method that you will implement later in this section to deal with miscellaneous problems with the login process.
- **Line 24** returns null so that the ADF controller will not attempt to follow a control flow case.

5. Import the following classes:

```
javax.faces.context.FacesContext
javax.servlet.http.HttpServletRequest
javax.security.auth.callback.CallbackHandler
weblogic.security.SimpleCallbackHandler
javax.security.auth.Subject
weblogic.security.services.Authentication
weblogic.servlet.security.ServletAuthentication
javax.servlet.http.HttpServletResponse
javax.security.auth.login.FailedLoginException
javax.faces.application.FacesMessage
javax.security.auth.login.LoginException
```

6. Implement the `sendForward()` method as follows (without line numbers):

```
01: private void sendForward(HttpServletRequest request,
02:                          HttpServletResponse response,
03:                          String loginUrl) {
04:   FacesContext ctx = FacesContext.getCurrentInstance();
05:   RequestDispatcher dispatcher = request
06:     .getRequestDispatcher(loginUrl);
07:   try {
08:     dispatcher.forward(request, response);
09:   } catch (ServletException se) {
10:     reportUnexpectedLoginError("ServletException", se);
11:   } catch (IOException ie) {
12:    reportUnexpectedLoginError("IOException", ie);
13:   }
14:   ctx.responseComplete();
15: }
```

Additional Information: This method performs the following actions:

- **Line 05–06** create a `RequestDispatcher`, which forwards a response to a particular URI.
- **Line 08** uses the `RequestDispatcher` to forward the current HTTP response to the URL specified by loginUrl.
- **Lines 09–10** handles a `ServletException`, which is thrown when the resource at the `RequestDispatcher`'s target URI throws an exception.
- **Lines 11–12 handle** an `IOException`, which is thrown when the request cannot be read or the response cannot be written to.
- **Line 14** marks the HTTP response as complete so that the browser can finish rendering it.

7. Import the following classes:

```
javax.servlet.RequestDispatcher
javax.servlet.ServletException
java.io.IOException
```

8. Implement the `reportUnexpectedLoginError()` method as follows:

```
private void reportUnexpectedLoginError(String errType, Exception e){
  FacesMessage msg =
    new FacesMessage(FacesMessage.SEVERITY_ERROR, "Unexpected Error During Login",
                     "Unexpected Error during Login (" + errType +
                     "), please consult logs for detail");
  FacesContext.getCurrentInstance().addMessage(null, msg);
  e.printStackTrace();
}
```

Additional Information: This method adds a summary error message to the `FacesContext`, and then prints the full stack trace of the exception to the console.

9. Select **Make** from the right-click menu in the editor to compile the file. (Ignore any deprecated method messages.) Open adfc-config.xml and select the Overview tab.

10. On the Managed Beans tab, add a managed bean with the following properties:
    Name as "login"
    Class as "tuhra.controller.LoginHandler"
    Scope as "request"

11. Click Save All.

Create the Login Popup

In this section, you will create the custom login UI. This is not a separate page, but rather a popup that appears from within tuhraTemplate.

1. Open tuhraTemplate.jspx in a visual editor. (Look under Web Content\WEB-INF\templates.)

2. In the visual editor, select the Help menu defined in the menu bar for the page, and in the Structure window, select the `af:group` component containing the menu bar.

3. Drag a Popup into the group (Layout panel, Secondary Windows section).

4. Set the Popup's *Id* to "pt_loginPopup."

   Additional Information: By convention, any object IDs assigned within a page template definition should be prefixed with "pt_." This reduces the possibility of experiencing a duplicate component name in a page that uses that template.

5. Drop a Panel Window into the Popup, and set the following properties on the Panel Window:
   Modal as "true"
   Title as "Log In to TUHRA"
   TitleIconSource as "/images/key.png"

6. Drop a Panel Form Layout into the window.

7. Drop an Input Text item into the Panel Form Layout, and set the following properties:
   Label as "Username"
   Value as "#{login.username}"

8. Drop another Input Text item after the first (but not into the footer facet), and set the following properties:
   Label as "Password"
   Value as "#{login.password}"
   Secret as "true"

9. Drop a Panel Group Layout into the footer facet of the form, and set the following properties:
   Layout as "horizontal"
   Halign as "end"

10. Drop a Spacer into the separator facet of the Panel Group Layout, and set its *Width* to 4.

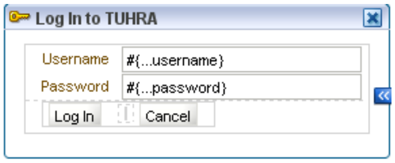
Tip

Use the Structure window, not the visual editor, to find the separator facet.

11. Drop a Button into the Panel Group Layout, and set the following properties:
    Text as "Log In"
    Action as "#{login.performLogin}"

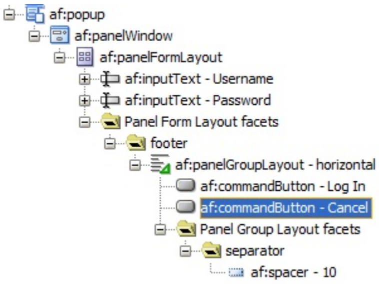
12. Drop another Button into the Panel Group Layout, and set the following properties:
    Text as "Cancel"
    Immediate as "true"

    Additional Information: The popup should look like the illustration on the right:



Username #{...username}
Password #{...password}

The structure window should look like the following:



```
af:popup
  af:panelWindow
    af:panelFormLayout
      af:inputText - Username
      af:inputText - Password
      Panel Form Layout facets
        footer
          af:panelGroupLayout - horizontal
            af:commandButton - Log In
            af:commandButton - Cancel
            Panel Group Layout facets
              separator
                af:spacer - 10
```

Create Menu Options to Log In and Log Out

In this section, you will create menu options to open the popup dialog and to log the user out.

1. In the Structure window, expand the `af:menuBar` that is a sibling of the popup.

2. Select **Insert inside af:menu - Action | Group** from the right-click menu on the Action menu component.

3. Still in the Structure window, drag the group so that it comes before the Save command menu item.

4. Drag a Menu Item into the `af:group`, and set the following properties on it:
   Text as "Log In"
   Rendered as "#{!securityContext.authenticated}"

   Additional Information: The *authenticated* property of the securityContext expression evaluates to true if the current user is authenticated; the Log In option will be rendered only if the user is not authenticated.

5. Drop a Show Popup Behavior into the Log In command menu item and set the following properties for it:
   PopupId as "pt_loginPopup"
   TriggerType as "action"

6. Drag another Menu Item into the group, and set the following properties on it:
   Text as "Log Out"
   Rendered as "#{securityContext.authenticated}"

7. Click Save All.

8. Run employeeSearch.jspx.

9. Expand the Action menu.

   Additional Information: The **Log In** menu item is displayed, because the user is not yet logged in. The **Log Out** menu item is not displayed.

10. Select the **Log In** option to open the "Log In to TUHRA" popup.

11. Enter "TFOX"/"**welcome2tuhra**" and click Log In.

    Additional Information: The Maintenance link is now displayed, because TFOX has access to it.

12. Select **Action | Log Out** from the menu bar inside the TUHRA application.

    Additional Information: The Log Out menu item is displayed (and the Log In menu item is not) because the user is logged in. However, selecting the Log Out menu item does nothing. You will implement logout in the following section.

13. Close the browser and stop the application and the server.

Implement a Method to Handle Logout

In this section, you will implement a method to log the user out, and edit the Log Out menu item to call it.

1. Open the `LoginHandler` class in an editor.

2. Add the following method to the class:

```
public String performLogout() {
  FacesContext ctx = FacesContext.getCurrentInstance();
  HttpServletRequest request =
    (HttpServletRequest)ctx.getExternalContext().getRequest();
  HttpServletResponse response =
    (HttpServletResponse)ctx.getExternalContext().getResponse();
  String logoutUrl =
    "/adfAuthentication?logout=true&end_url=/faces/employeeSearch";
  sendForward(request, response, logoutUrl);
  return null;
}
```

   Additional Information: This method forwards the user to the URL specified in the `logoutUrl` variable. The adfAuthentication resource will log the user out (because of the logout parameter) and forward the user to /faces/employeeSearch (because of the end_url parameter).

3. Re-open tuhraTemplate.jspx.

4. Find and select the **Log Out** menu item.

5. Set the *Action* of the item to "#{login.performLogout}."

6. Click Save All.

7. Run employeeSearch.jspx, and retry the login and logout functionality using TFOX as the user.

   Additional Information: The Log Out menu item now logs the user out.

8. Close the browser and stop the application.

What Did You Just Do?

You created a managed bean with methods to log the user in and out. You also created a popup containing a login screen, which calls the login method, and menu items to bring up the popup for non-authenticated users and log authenticated users out.

IV. Access Information about the Logged-In User

The application already accesses certain information about the logged-in user—namely, whether the user has a particular role and whether the user is authorized to access a particular resource. In this phase, you will implement functionality that relies on other information about the user: displaying information about the user and allowing the user (even if not a manager) to edit his/her own records.

Create a View Object Definition to Return User Information

In this section, you will create a view object definition that retrieves information about the logged-in user, and add an instance of it to the data model.

1. In the Model project, in the `tuhra.model.queries.main` package, open the Create View Object wizard. (Select New View Object from the right-click menu on the package node.)
2. On the Name page, name the view object definition “UserInfo.”
3. Click Next. On the Entity Objects page, add a usage of the entity object definition Employees and deselect the *Updatable* checkbox.
4. Click Next. On the Attributes page, select the following attributes:
   Email
   EmployeeId
   FirstName
   LastName
5. Click Next. Click Next. On the Query page, add the following WHERE clause:

   ```
   Employees.EMAIL = :AuthenticatedUserId
   ```

6. Click Next. On the Bind Variables page, add a bind variable with the following properties and click Finish:
   Name as “AuthenticatedUserId”
   Type as “String”
   Value Type as “Expression”
   Value as “viewObject.DBTransaction.session.userPrincipalName”

   Additional Information: The value you specified is a Groovy expression, which calls the following methods on the view object and returns the result:

   ```
   getDBTransaction().getSession().getUserPrincipalName()
   ```

   This is the login ID (email) of the logged-in user.

7. Click Finish. In the editor for the UserInfo view object definition, in the General tab, expand the Tuning panel, and select “At Most One Row.”

   Additional Information: This ensures that no more than one row will be fetched from the database, but also makes it clear that you as a developer only ever expect a single row to match.

8. Add an instance of this view object definition, also called “UserInfo,” to TuhraService’s data model. (Double click TuhraService under the tuhra.model.services node in the Application Navigator.)
9. Click Save All.

Display User Information

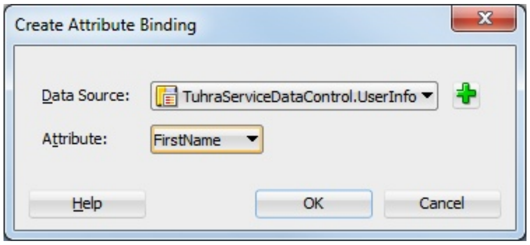
In this section, you will add information about the logged-in user to the application pages.

1. In the Application Navigator, refresh the Data Controls panel and then open tuhraTemplate.jspx (under ViewController\Web Content).

2. Select the Bindings tab.

3. Create attributeValues bindings for the FirstName and LastName attributes of UserInfo.

   Additional Information: For *DataSource*, select “UserInfo” (by clicking the green plus “+”) and select *Attribute* as “FirstName” as shown next; repeat for the LastName.

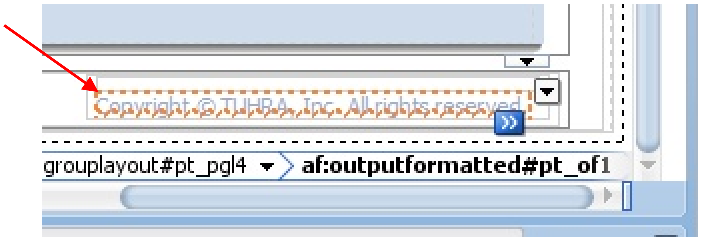


Data Source: TuhraServiceDataControl.UserInfo

Attribute: FirstName

4. Return to the Design tab.

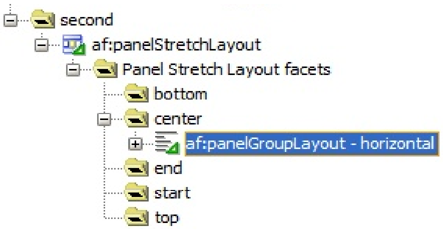
5. Select the copyright notice in the footer of the page as shown next, then in the Structure window, select the horizontal Panel Group Layout that contains the vertical Panel Group Layout above that text.


Copyright © TUHRA, Inc. All rights reserved
grouplayout#pt_pgl4
af:outputformatted#pt_of1

6. Cut (CTRL-X, do not Delete) the horizontal Panel Group Layout. This action will place it and its child components into the clipboard.

7. Drop a Panel Stretch Layout on top of the second facet where the node you just cut was previously located.

8. Expand the Panel Stretch Layout facets node and paste (CTRL-V) from the clipboard into the center facet. The Structure window should appear as follows:



```
second
  af:panelStretchLayout
    Panel Stretch Layout facets
      bottom
      center
        af:panelGroupLayout - horizontal
      end
      start
      top
```

   Additional Information: You cannot use the **Surround With** menu option to add a Panel Stretch Layout only its facets can contain components. So you need to place the Panel Stretch Layout, then add components to its facets.

Tip

Remember that Undo (CTRL-Z) can restore previous states of the code.

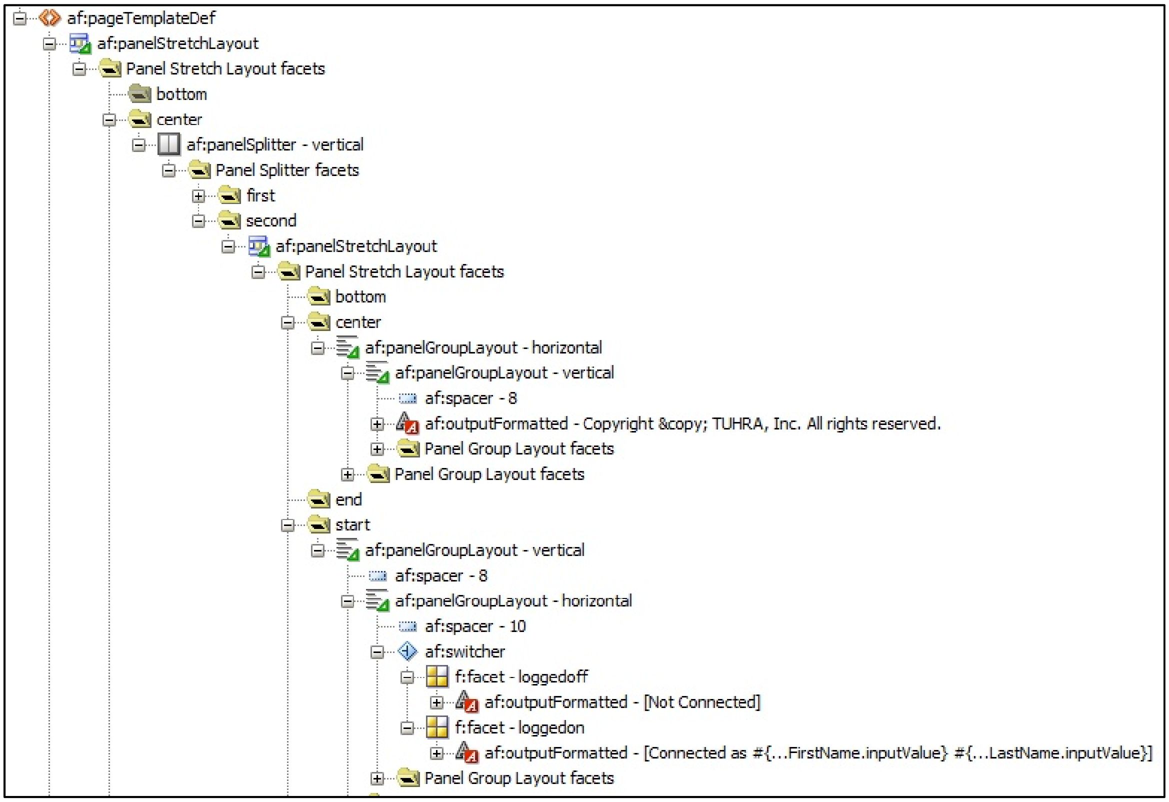
9. In the Property Inspector for the new Panel Stretch Layout, set *StartWidth* as “200px,” *EndWidth* as “100px,” *TopHeight* as “0px,” and *BottomHeight* as “0px.”

10. Into the start facet of the new Panel Stretch layout, drop a Panel Group Layout, and set the *Layout* property to “vertical.”

11. Drop a Spacer into the Panel Group Layout, and set the following properties:
    Height as 8
    Width as 200

12. Drop another Panel Group Layout after the new spacer, still inside the vertical `af:panelGroupLayout` component, and set its *Layout* property to “horizontal.”
13. Drop a Spacer inside this Panel Group Layout, and set its *Width* to “4.”
14. Drop a Switcher (Operations panel) after the Spacer, and set the following properties on it:
    FacetName as “#{securityContext.authenticated?'loggedon':'loggedoff'}”
    DefaultFacet as “loggedoff”

    Additional Information: Depending on whether the user is authenticated, the switcher will display one of two facets, “loggedon” or “loggedoff.” If the call returns null, the switcher will display its default facet.
15. Select Insert inside **af:switcher | Facet** from the switcher’s right-click menu.
16. Name the facet “loggedoff.”
17. Drop an Output Text (Formatted) inside the facet, and set the following properties:
    Value as “[Not Connected]”
    StyleUsage as “pageStamp”
18. Repeat steps 14–16 to create a facet named “loggedon,” with an Output Formatted with the *Value* “[Connected as #{bindings.FirstName.inputValue} #{bindings.LastName.inputValue}].” The Structure window at this point will look like Figure 5 (under `jsp:root`).
19. Click Save All.
20. Run employeeSearch.jspx.
    Additional Information: The text “[Not Connected]” should appear in the page footer, along with the copyright notice.
21. Log in as CDAVIES.
    Additional Information: The text “[Connected as Curtis Davies]” should appear in page footer.
22. Close the browser and stop the application.



```
af:pageTemplateDef
  af:panelStretchLayout
    Panel Stretch Layout facets
      bottom
      center
        af:panelSplitter - vertical
          Panel Splitter facets
            first
            second
              af:panelStretchLayout
                Panel Stretch Layout facets
                  bottom
                  center
                    af:panelGroupLayout - horizontal
                      af:panelGroupLayout - vertical
                        af:spacer - 8
                        af:outputFormatted - Copyright &copy; TUHRA, Inc. All rights reserved.
                        Panel Group Layout facets
                      Panel Group Layout facets
                  end
                  start
                    af:panelGroupLayout - vertical
                      af:spacer - 8
                      af:panelGroupLayout - horizontal
                        af:spacer - 10
                        af:switcher
                          f:facet - loggedoff
                            af:outputFormatted - [Not Connected]
                          f:facet - loggedon
                            af:outputFormatted - [Connected as #{...FirstName.inputValue} #{...LastName.inputValue}]
                        Panel Group Layout facets
```

Figure 5. Structure window (Design tab) after adding user information

Allow a User to Edit His/Her Own Records

In this section, you will add a menu item to the Action menu that will let users (even non-managers) edit their own information.

1. In tuhraTemplate.jspx, select the Action menu where you previously created the Log In and Log Out options.

2. Add a Menu Item to the `af:group` component inside the Action menu.

3. Set the following properties on the menu item:
   Text as "Edit My Information"
   Action as "edit"
   Disabled as "#{!securityContext.authenticated}"

   Additional Information: The "edit" control flow case points to editEmployee.jspx. By setting *Disabled* to `#{!securityContext.authenticated}`, rather than setting *Rendered* to `#{securityContext.authenticated}`, you gray out the item for non-authenticated users rather than hiding it entirely.

4. Select the Bindings tab.

5. Add an attributeValues binding for the EmployeeId attribute of UserInfo.

6. Reselect the Design tab.

7. Drop a Set Property Listener (Operations panel) into the Edit My Information menu item and set the following properties:
   From as "#{bindings.EmployeeId.inputValue.value}"
   To as "#{uiState.editEmployeeId}"
   Type as "action"

   Additional Information: This sets the editEmployeeId that is used as a parameter to the editEmployee-flow task flow, to be the employee ID of the currently logged in user. That way, a user will always edit his or her own record if he or she open the Edit Employee page via this menu item (rather than from a search result as a manager).

8. Click OK. Click Save All.

9. Run employeeSearch.jspx.

10. When the browser opens, expand the Action menu.

    Additional Information: The Edit My Information menu item appears disabled because you are not yet logged in.

11. Select **Action | Log In** and log in as CDAVIES.

12. Select **Action | Edit My Information**. The Edit Employee page opens to CDAVIES' record.

13. Close the browser and stop the server.

What Did You Just Do?

You added functionality to the application to display the logged in user's name and to allow the user (regardless of role) to access the Edit Employee page for his or her own record. The application now incorporates connection points to the WLS security services.

What Could You Do Next?

The Edit Employee page allows the user to edit too much information about him- or herself, including salary, hire date, and manager. You could set the *readOnly* attribute on these fields so that they can be edited only by a manager.

More ambitiously, you could change the way the link from search results to the edit page is implemented. Rather than allowing any manager to edit any employee, you could change the functionality such that a user can edit an employee from a search result only if one of the following is true:

- The user has the "admin" application role.
- The user is in the employee's management chain.

The latter would most likely involve creating a service method to determine if one employee is in another's management chain and passing the logged-in and selected employees' IDs as parameters to the method.