WEB APPLICATION SECURITY WITH JAZN—IMPLEMENTING THE SUPERSTITION IN JDEVELOPER

Peter Koletzke, Quovera
Duncan Mills, Oracle Corp.

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)

We will bankrupt ourselves in the vain search for absolute security.

—Dwight David Eisenhower, (1890–1969)

Note: This white paper is adapted from material written by these authors for the book JDeveloper 10g for Forms & PL/SQL Developers, *Oracle Press (2006), ISBN 0072259604.*

Data is the lifeblood of an organization. Decisions are made; customers and clients are served; and careers are made because of data. Therefore, securing data from unauthorized access is an important requirement for any database system.

The Oracle Application Server offers a standard J2EE container (server runtime), *Oracle Containers for J2EE (OC4J),* that executes web application code written in technologies as Java servlets, JavaServer Pages, and JavaServer Faces. OC4J provides a security feature, *JAZN* (Java authorization and authentication), which allows developers to use standard Java security libraries to implement user access and restriction features in their applications.

This white paper briefly examines the security needs of a J2EE web application. It then discusses how JAZN provides solid security services for J2EE web applications, how you can store user credentials in JAZN, and how to tap into existing LDAP providers or XML files. Finally, it provides steps in JDeveloper for setting up JAZN security and for writing security hooks into your application.

Application Security

Application security is an important component of system design. You must carefully plan 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. Security considerations are slightly different in Oracle Forms and Oracle Reports applications than in J2EE web applications.

Security in Oracle Forms and Oracle Reports Applications

Applications developed with Oracle tools such as Oracle Forms and Oracle Reports often managed database security by requiring each user to obtain a database login account and password. This strategy required the database administrator to set up the account and its password and the system administrator to make the TNSNAMES.ORA SQL*Net configuration file available to the user or server (if using web deployed Oracle Forms). However, with this policy of issuing individual database accounts to control access, users could often access the database using tools other than the approved application, unless special policy code was written to block this type of access.

Another common strategy (used by Oracle E-Business Suite) uses a single database login user and requires users to log in using an application account. Privileges to application functions and data are granted through application groups and users assigned to the groups inherit the privileges of the group (in the same way that database users assigned a database role inherit the privileges granted to the role). With this strategy, the user never knows or cares about the database login account and cannot access data using tools other than the application. This strategy also requires access to the network so the application can find the database.

Security in Web Applications

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, unlike Oracle Forms and Oracle Reports applications that require the user to establish a database login session to access the application, web applications handle the database connection automatically in the Model layer code. As with the 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 Oracle Application Server 10*g* provides authentication features that require users to log into an application session. The user accounts on the application server serve 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.

Authentication and Authorization

Logging into a J2EE application using the Oracle Application Server requires two stages—both of which are provided by the OC4J container. These stages follow:

1. 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 user repository can be in the form of an XML file or a directory service (such as an LDAP server). The security service verifies the identity of the user based on entries in the user repository.

2. 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 roles to which the user belongs. 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 J2EE application, the rules are stored in one deployment descriptor file and the roles are mapped in another descriptor file to users and user groups. The logged in user and the user's roles from the authentication stage are given access based on this mapping.

In addition, the application can read the logged-in user's name and role and hide or disable restricted parts of the application appropriately. In the JDeveloper practice at the end of this white paper, you will set up the files to provide user authentication and authorization. You will also define some application code to provide access to pages and items on the page based on the user's role.

JAAS

Java Authentication and Authorization Services (JAAS) provides security services for Java-based applications from a library in the Java JDK. It 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.

JAZN

JAZN, the Oracle JAAS provider, is the facility inside the OC4J server that allows OC4J to use the JAAS library and its standards. Oracle Application Server provides security services through JAZN as well as services such as single sign-on and network encryption. JAZN also provides the ability to authorize based on container security and, thus, supplies the authentication and authorization functions for OC4J. JAZN offers two providers for these services: JAZN-XML and JAZN-LDAP.

JAZN-XML

JAZN-XML provides JAAS services to OC4J using XML files for user information. It offers a fast and easy way to code user information into files rather than using an enterprise-wide service. Therefore, it is well-suited to development work. The user repository resides inside an XML file, *jazn-data.xml*, which stores usernames and encrypted passwords as well as the names of groups to which the users belong. JDeveloper offers a property editor to assist in setting up users and roles; you will use this editor in the practice at the end of this white paper.

JAZN-LDAP

JAZN-LDAP provides JAAS services to OC4J using a Lightweight Directory Access Protocol (LDAP) directory access system. The *Oracle Internet Directory (OID)* component of Oracle Application Server provides LDAP services but JAZN-LDAP can use other LDAP systems as well. JAZN-LDAP supports OracleAS Single Sign-On (a facility for passing user login information between applications). It is intended as a provider for an enterprise-ready production environment.

ALTERNATIVE PROVIDERS

Because JAZN uses JAAS under the covers to implement authentication, it is possible to leverage the *Pluggable Authentication Module (PAM)* mechanism defined by the JAAS standard. The idea behind PAMs is to provide a standard way of plugging in alternative authentication mechanisms using a LoginModule. A *LoginModule* is a JAAS API you can use to interface with existing security systems. This is of particular interest to many Oracle Forms customers who have an existing security infrastructure using a credential store based on database tables and PL/SQL packages. PAMs allow these existing mechanisms to be reused to secure J2EE applications. The Oracle technology network contains resources that discuss how to write a custom LoginModule. Of particular interest is the paper “Declarative J2EE authentication and authorization with JAAS” available at www.oracle.com/technology/products/jdev/howtos/10g/jaassec/index.html.

SWITCHING JAZN PROVIDERS

The practice in this white paper uses the JAZN-XML provider. By default, the OC4J container is configured to use the XML-based user repository, jazn-data.xml, for each application. This user repository is configured through a file, which can be found in the /j2ee/home/config directory within an OC4J installation—application.xml for the application. The application.xml file specifies the type of security provider through the JAZN XML element. Typically the entry would look something like this:

```
<jazn provider="XML" location="jazn-data.xml" default-realm="jazn.com"/>
```

In order to switch to using LDAP to provide user information, this element would change to something like:

```
<jazn provider="LDAP" location="ldap://ldap.tuhra.com:389">
```

Notice how it is relatively simple to switch the security provider without having to change the application. You might modify the code in this way if you were moving from testing the application within a local OC4J instance (or the JDeveloper Embedded OC4J Server) that used an XML-based user repository, to deploying the application into production using an LDAP-based user repository.

Note

Oracle Application Server 10*g*, Release 3 (10.1.3) uses the system-wide JAZN files system-jazn-data.xml and system-application.xml. These files are located in the J2EE_HOME\config directory on the server.

Directory Services

Directory services software, an application server feature, provides a link between the application server and an established user access control list that is external to the application server. This allows an application to use the directory services from the application server to access the external user list. In Oracle Application Server 10*g*, Release 3, any LDAP server can supply these directory services. The examples that follow use LDAP services from Oracle Internet Directory, which can tap into the user list in an existing *Lightweight Directory Access Protocol (LDAP)* system (such as the Windows 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 issues an HTTP request to the application server. This request includes a context root indicating a particular application, TUHRA in this example.

2. The authentication service in the application server reads a configuration file and determines that access to this application must be authenticated by a particular security strategy. It then presents a login page.
3. The user enters an ID and password and submits the page.
4. The authentication service requests OID to verify the user and password.
5. OID looks up the user in the LDAP repository and verifies the password. It indicates to the authentication service that the user and password is valid.
6. The application server authentication service accesses the application and places the user name into the HTTP session state. The application can read the HTTP session at any time to determine the user's name.
7. The application can also request the group or role (in this example, "manager") to which the user belongs from OID at any point, using standard servlet APIs provided for the purpose.
8. The application connects to the database using the application database user account (the same database account for all web users), in this example, APPUSER. It starts a connection session in the database for that user. The user name and password are written into a configuration file that the application can access. This file is not available to any user but (with JDeveloper connection files), the password can be encrypted within the file using a feature called *password indirection*.

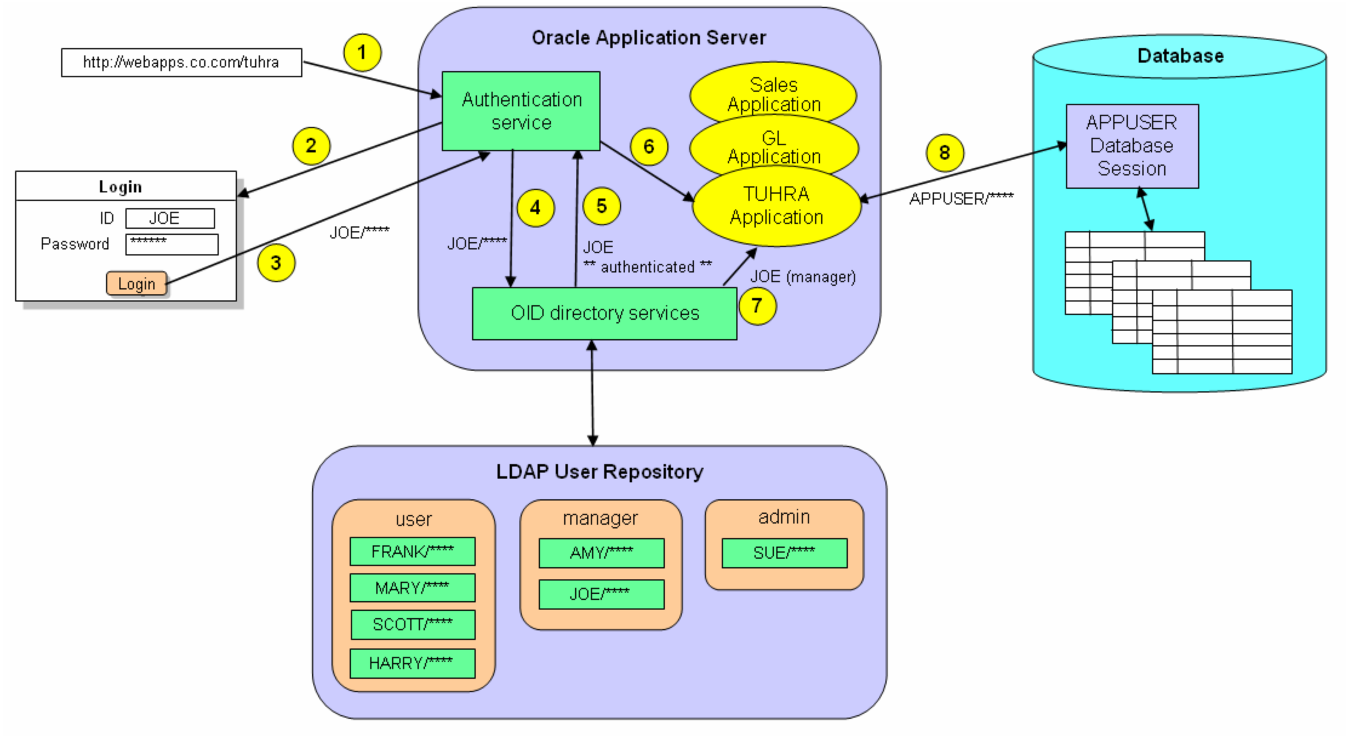


Figure 1. Directory services used for a J2EE web application

Note

Figure 1 represents the LDAP-based user repository. The sample application you develop at the end of this white paper uses the XML-based user repository to demonstrate security within an OC4J server. The documentation for OC4J (for example, the "Oracle Application Server Containers for J2EE Security Guide 10*g* Release 3 (10.1.3)"online book provides further overviews and details of security implementations in OC4J

Levels of Security in a Web Application

Due to the nature of the worldwide web, web applications are often more widely available than traditional Oracle Forms and Oracle Reports applications. Since more people can potentially access an application with only a web browser, an approach that addresses multiple layers is necessary. Based on the strategies discussed earlier, security can be provided at the following levels:

- **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.
- **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.

Matching the Authorized User to the Database

Although the preceding discussion mentions authentication and authorization as being separate from the traditional database login we are accustomed to in Oracle Forms and PL/SQL, some applications will still need to maintain some form of mapping between the authenticated user account on the application server and the database data. For example, the application may need to display the logged in user on the screen or write audit information about the logged in user to audit columns in a table.

In the case of the application you develop at the end of this white paper, we have chosen for convenience to use the EMAIL value in the EMPLOYEES table as the container managed security user ID. In reality, this would make it difficult to change the database user's email address. Any such change would have to be cascaded into the security system outside of the database, an unnecessary complication. In your own system designs you would instead use a separate column or lookup table to map the two different types of identity if that is a feature that you require.

Logging Data Modifications

Although they are not unique to web applications, two techniques familiar to Oracle developers can identify who modified data and what they modified. This information can help you research questions about security breaches or security holes.

DATA JOURNALING

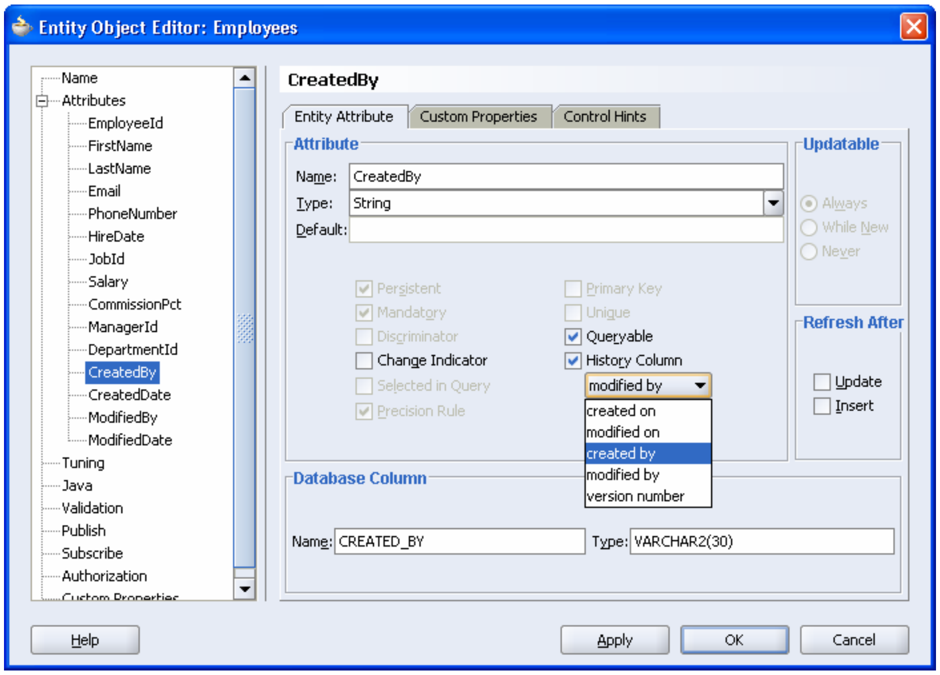
Data journaling (also called "archiving") refers to saving a copy of each record that is modified or deleted. Typically, this copy is placed in a separate journal table that is designed with the same structure as the original table. Row-level BEFORE triggers on UPDATE and DELETE save the entire record in its pre-operation state. They also save information about who performed the action and when the action was performed.

DATA AUDIT COLUMNS

You can add columns to each table to store the name of the user who created the record and the date of creation as well as the name of the user who last modified the data and the date the record was last modified. This information will not provide the same detailed history of changes as the preceding technique but it is simpler to implement.

ADF Business Components contains a declarative mechanism for carrying out this form of auditing: the entity object attribute *History Column* property. Figure 2 shows an entity object attribute called CreatedBy in the Entity Object Editor.

Checking the History Column checkbox enables a dropdown list containing a list of audit operations. Once you assign an audit operation to an entity attribute using this mechanism, ADF Business Components will automatically maintain the audit content columns for you without the need to write database triggers.



CreatedBy

Attribute

Name: CreatedBy
Type: String
Default:

[✓] Persistent
[✓] Mandatory
[ ] Discriminator
[ ] Change Indicator
[ ] Selected in Query
[✓] Precision Rule
[ ] Primary Key
[ ] Unique
[✓] Queryable
[✓] History Column

modified by
created on
modified on
created by
modified by
version number

Updatable: [● Always] While New Never

Refresh After: [ ] Update [ ] Insert

Database Column

Name: CREATED_BY Type: VARCHAR2(30)

Figure 2. Defining a history column in the Entity Object Editor

Tip

If you have JAZN security turned on in the application (as described in this paper), the "modified by" and "created by" history column values (shown in Figure 2) are set to the application user name, not the connection user name. If you do not have JAZN security turned on, all history column values will be NULL.

The Problem with USER

The History Columns technique will ensure that the application writes audit information, for example, the logged-in user, to the associated database columns. However, if your table data can be updated outside of the application (for example, by IT staff), you might choose to audit on a more global level using database triggers to capture user information.

The problem you face when implementing these techniques for a web application is in identifying the user. When users log into an application with separate database accounts, the database triggers can use the pseudo-column USER to capture the username. However, when users log in with an application username and connect to the database with a common database username, the pseudo-column USER will be the application database user account. However, you need to record the username set in the application server session. One solution to this problem is to write the application username into a database context. Database triggers and other PL/SQL can then read the application username from the context instead of using the USER pseudo-column.

JDeveloper Practice: Working with JAZN Security

This section provides practice with setting up JAZN security for a sample application. Before beginning to incorporate any security code, you need to carefully design the roles and access privileges for those roles. The sample application you develop in this section consists of a home page, a browse page for employee records, and an edit page for employee records; the edit page can be used for modifying records or creating records. You will also create login and logout pages. Three logical roles—user, manager, and admin—are given the following privileges:

Privilege	user	manager	admin
View any employee record	Y	Y	Y
View salary and commission information	N	N	Y
Edit an employee record	N	Y	Y
Create an employee record	N	N	Y

It is important to understand that these roles are private to the application and do not directly represent roles in the database or any external user repository. The reason for this approach is that it provides us with a security abstraction. You can hook up a security-provider implementation that has its own concept of roles, which may or may not match those within the application. If the roles do not match directly, a mapping can be made between the security provider role name and the application role name once the application is installed into a J2EE runtime container.

In the sample application, we simplify the security implementation by using the XML file security provider built into the OC4J container and by defining identical role names in this provider to eliminate the need for any mapping. All security within the application is then based on these logical roles. Although the identity of the logged in user is available to the program, general tasks such as access control to screens should always be controlled by role.

You use the role information in two ways: first within the user interface to hide or display elements conditionally. For example, a normal user should not see the button that leads to the employee edit screen. This is an effective first level of security. However, one weakness of browser based applications is the browser's location (address) bar. There is nothing to stop a user from manually altering the URL to access a different part of the application directly. Even if you go to great lengths to customize the browser to hide the location bar, this will not prevent any serious security attack. Fortunately container security allows you to use the role information to protect specific URL patterns representing files in the file system. Therefore, as a second level of defense, you can partition the application into various subdirectories under the root and protect sensitive pages accordingly. This practice uses directory-based security to protect the browse and edit pages so they may be accessed only by logged-in users. The edit page will be placed into another directory so that access can be further restricted by role of the logged-in user.

The practice steps that follow create a simple web application using JDeveloper 10.1.3 and security features to it. The work follows these phases:

I. Create a sample application

- Create the application and a connection
- Define the Model project
- Define the page flow
- Create the browse page
- Create the edit page
- Create the home page and test the application

II. Configure container security

- Define the users and passwords
- Set up the user roles
- Browse the security data

III. Define application security settings

- Set up the logical application roles
- Define the protected URLs with security constraints
- Test the constraints

- Set up a login page
- Detect multiple login attempts
- Set up a logout page
- Set up the application entry page
- Switch security on
- Ensure that security is triggered

IV. Add security to the user interface

- Install JSF-Security
- Add an user ID indicator
- Define access for the admin and manager roles

Note

The application files that result from completing these steps are available in Employees_end.zip contained in the Zip file for this white paper. A README.TXT file explains how to install the sample solution application. If you do not have access to the conference proceedings Zip files, you can find this white paper and sample code at www.quovera.com and ourworld.compuserve.com/homepages/Peter_Koletzke.

I. Create a Sample Application

This practice uses JDeveloper 10.1.3, available as a free download from otn.oracle.com (select the studio full install version if you do not have it installed already). You also need access to the sample HR schema installed with Oracle8*i*, 9*i*, or 10*g*. The practice builds an application for the EMPLOYEES table of this schema. If access to this schema is not possible or practical, you can use any schema and any table as an example, although you will need to adjust the instructions accordingly.

Note

Although this phase provides good practice for creating JSF applications in JDeveloper, you can skip this phase and start with the security-specific steps in phase II. You will need to install the sample application for this phase contained in the Employees_start.zip (available from the sources mentioned in the last note. As before, the README.TXT file in that Zip file contains installation instructions.

Create the Application and a Connection

You work with application code in JDeveloper using a container called an *application* (or *application workspace*). The application contains projects, which contain application files. Projects are usually divided into functional areas. For example, a web application, by default, contains a project for the Model code (used to access the database) and another project for the View and Controller code (used to present the user interface and respond to user events). This section sets up the application as well as a connection to the HR schema.

1. Open JDeveloper from a shortcut icon or by double clicking jdeveloper.exe in the JDEV_HOME directory (where you installed JDeveloper.
2. On the Applications node of the navigator, select New Application from the right-click menu. The Create Application dialog will appear. Fill in the following fields:

   Application Name as "Employees"

   Application Package Prefix as "hr"

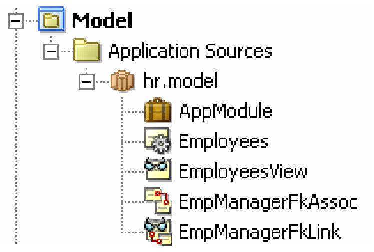
   Application Template as "Web Application [JSF, ADF BC]"

3. Click OK. The application workspace and two projects will appear in the navigator.
4. If you have already created a connection for the HR schema, skip to the next section ("Define the Model Project").
5. Click the Connections Navigator tab (if this tab is not visible, select **View | Connection Navigator** from the menu).
6. On the Database node, select New Database Connection from the right-click menu. Click Next if the Welcome page appears.
7. Enter the *Connection Name* as "HR" and click Next.
8. Enter *Username* as "HR" and *Password* as "HR". Check the *Deploy Password* checkbox. Click Next.
9. Enter the *Host Name*, *JDBC Port* (database port, usually 1521), and *SID* as appropriate to your database, and click Next.
10. Click Test Connection and fix any errors that are reported. Click Finish to complete the connection definition. Click the Applications Navigator tab.

Define the Model Project

This section creates Application Development Framework Business Components (ADF BC) files used to access the database from the web user interface layer. Since the purpose of this practice is to demonstrate security principles, we will not explore the details and features of this layer further.

1. On the Model project, select New from the right-click menu. Select the Business Tier\ADF Business Components category and double click the Business Components from Tables item.
2. The Initialize Business Components Project dialog will appear. Select the HR connection and click OK. Click Next if the Welcome page of the Create Business Components from Tables Wizard appears.
3. Select the EMPLOYEES table and move it to the *Selected* list using the ">" button. Click Next. Select the Employees entity object and move it to the *Selected* list to define the EmployeesView view object. Click Next
4. Click Next to bypass the Read-Only View Objects page. On the Application Module page click Finish. The ADF BC files will be created and additional nodes for the Employee table will appear under Model\Application Sources\hr.model node of the navigator as shown here:



5. Click Save All (in the JDeveloper toolbar).

Define the Page Flow

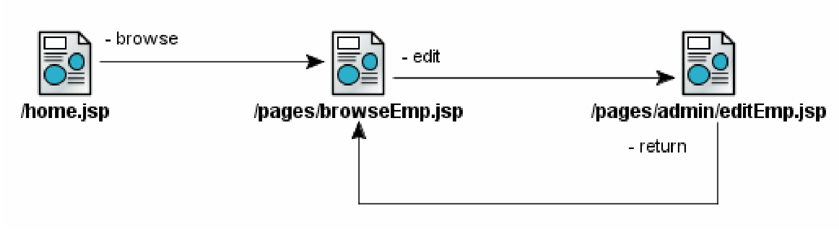
This section sets up navigation between pages in the application. You define the home page in a non-protected directory so non-logged-in users can access it. The browse page will be defined in a "/pages" directory that will be available only to logged-in users. The edit page will be placed in a "/pages/admin" directory so that it is available only to logged-in users who are members of the admin and manager roles.

1. On the ViewController node of the navigator, select Open JSF Navigation from the right-click menu. The JSF Navigation Diagram tool will open. You will draw page and navigation cases on this diagram to define the page flow of this application.
2. Drag a JSF Page from the Component Palette and drop it on the page. (If the Component Palette is not visible, select it from the View menu.) The name will be editable as soon as you drop the page symbol. Change this name to "/pages/browseEmp" and press ENTER. (JDeveloper will fill in the ".jsp" extension automatically.) If the name is not editable, click the name to open a field editor.

Tip

The top menu bar View menu contains selections for all main windows in JDeveloper. Select windows from this menu if you do not see them displayed in the JDeveloper IDE.

3. Drag and drop another JSF Page to the right of the first page and name it "/pages/admin/editEmp". Press ENTER.
4. Drag and drop a JSF Page to the left of the first page and name it "/home.jsp".
5. Click the JSF Navigation Case component in the Component Palette. Click "/pages/browseEmp.jsp" then click "/pages/admin/editEmp.jsp" to draw the navigation line between the two pages.
6. Click the "success" label to open its field editor and rename it to "edit". Click ENTER to exit the field edit process.
7. Draw another JSF Navigation Case from editEmp.jsp to browseEmp.jsp. Change its name to "return". This defines the flow from the edit page back to the browse page.
8. Draw another JSF Navigation Case from home.jsp to browseEmp.jsp and name it "browse". The diagram should appear something like this:



Create the Browse Page

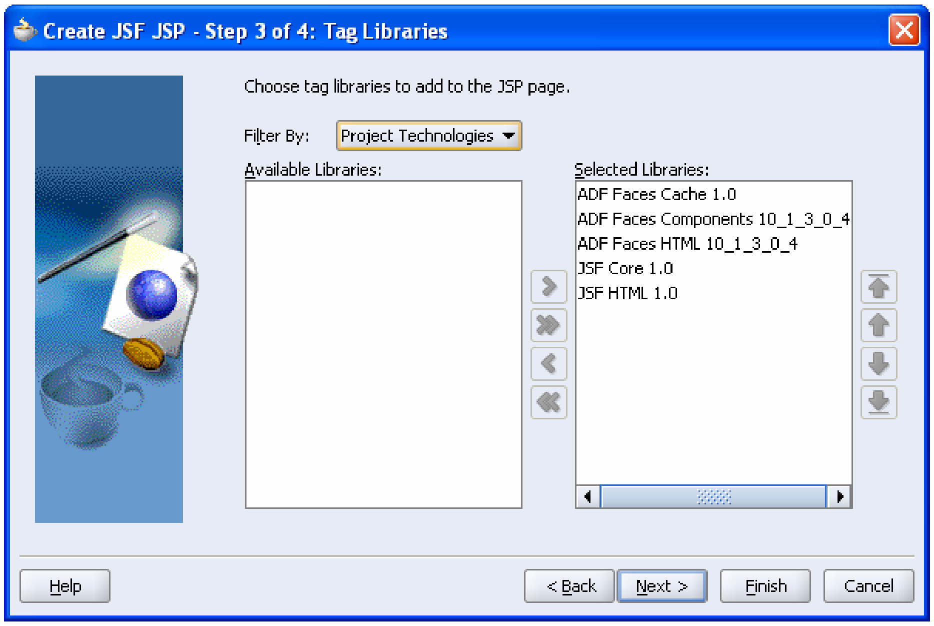
Now you can create user interface files for the web pages. You will create a separate file for each JSF page you added to the diagram. You will also create a login page and a logout page. All but the login page will use JavaServer Faces (JSF) components to display HTML controls in the browser. Specifically, you will use the ADF Faces component set—a tag library developed by Oracle and released to the Jakarta open source project in January 2006. The ADF Faces components offer a lot of prebuilt functionality that makes developing web applications easier. In this section, you will create the browse page.

1. Double click "/pages/browseEmp.jsp" to start the Create JSF JSP Wizard. Click Next if the Welcome page appears. Click Next on the JSP File page.
2. Be sure "Do Not Automatically Expose UI Components in a Managed Bean" is selected and click Next.
3. Be sure the libraries shown in Figure 3 are defined in the *Selected Libraries* area (move them after selecting "All Technologies" if they are not defined.
4. Click Next and change the Title to "Browse Employees" (for the browser window title). Click Finish to create the JSP file and open it in the editor.
5. Select the Component Palette tab. On the ADF Faces Core page of the Component Palette, drop a PanelPage component onto the page. A number of boilerplate components will be rendered as shown in Figure 4.

   Additional Information: The *af:panelPage* component (referenced by its initial lowercase tag name including the tag library prefix "af") provides layout areas (called *facets*) in predefined locations on the page. You can add components to the facets as needed and they will always be displayed in their assigned location at runtime.
6. Click "Title 1" and change the *Title* property to "Browse Employees" in the Property Inspector.

Note

After setting any property in JDeveloper, press ENTER to register the change.



Choose tag libraries to add to the JSP page.

Filter By: Project Technologies

Available Libraries:

Selected Libraries:
ADF Faces Cache 1.0
ADF Faces Components 10_1_3_0_4
ADF Faces HTML 10_1_3_0_4
JSF Core 1.0
JSF HTML 1.0

Figure 3. Library selection in JSF JSP Wizard

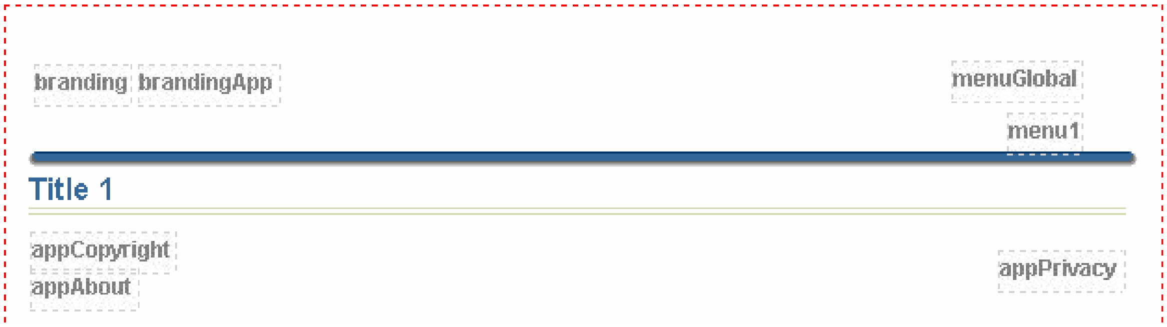


Figure 4. A PanelPage component in the visual editor

7. Click the Data Controls tab of the Component Palette area and expand AppModuleDataControl. Drag EmployeesView1 to the page over the panelPage component, and select **Tables | ADF Read-only Table** from the context menu.

8. In the Edit Table Columns dialog, select DepartmentId and click Delete. Also delete Email, PhoneNumber, Salary, CommissionPct, and ManagerId. Check the *Enable selection* and *Enable sorting* checkboxes and click OK. A representation of the table display will appear on the page.

   Additional Information: The text inside some of the cells uses *Expression Language (EL)* to retrieves information from the Model project. JSF technology uses EL extensively to dynamically supply property values.

9. In the visual editor, select the Submit button. In the Property Inspector, change the *Text* property to “Edit” and select “edit” from the pulldown in the *Action* property. The latter setting (the name you gave the JSF navigation case on the diagram) will cause the navigation from the browse page to the edit page when the user clicks the Edit button.

10. In the Structure window, ensure that the af:tableSelectOne tag is visible. This component contains action items such as the Edit button.

11. From the Data Control Palette, expand the EmployeesView1\Operations node and drag and drop the Create operation on top of the af:tableSelectOne node in the Structure window. This node represents adding a blank record to the ADF BC cache so the edit page will appear blank.

12. Select ADF Command Button from the context menu. A button labeled “Create” will appear in the editor and will be selected. In the Property Inspector, change *Text* to “New” and *Action* to “Edit” . The Structure window should contain the excerpt shown in Figure 5

13. Click Save All.

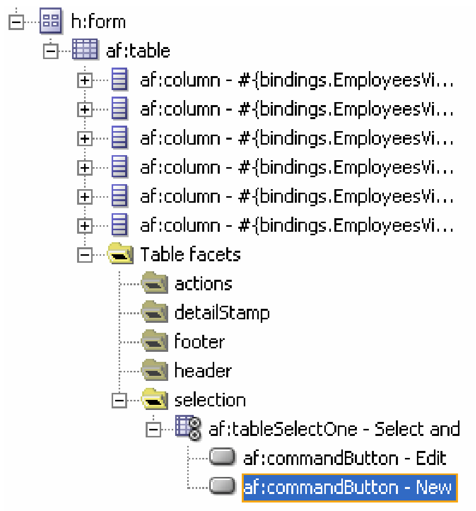
Create the Edit Page

This section creates the employee edit page.

1. Click the faces-config.xml tab in the editor window to redisplay the navigation diagram. Double click the /pages/admin/editEmp.jsp symbol in the diagram to start the JSF JSP Wizard. Click Next if the Welcome page appears.

2. Click Next until you reach the HTML Options page. Fill in Title as “Edit Employee” and click Finish. The editEmp.jsp file will be created and will open in the visual editor.

3. As with the browse page, drop a PanelPage component onto the page, and change its title to Edit Employee.



```
h:form
  af:table
    af:column - #{bindings.EmployeesVi...
    af:column - #{bindings.EmployeesVi...
    af:column - #{bindings.EmployeesVi...
    af:column - #{bindings.EmployeesVi...
    af:column - #{bindings.EmployeesVi...
    af:column - #{bindings.EmployeesVi...
    Table facets
      actions
      detailStamp
      footer
      header
      selection
        af:tableSelectOne - Select and
          af:commandButton - Edit
          af:commandButton - New
```

Figure 5 Structure window for browseEmp.jsp

4. From the Data Control Palette, drop EmployeesView1 onto the page over the panelPage component. Select **Forms | ADF Form** from the context menu.

5. In the Edit Form Fields dialog, delete ManagerId, DepartmentId, and PhoneNumber (non-required columns). Check *Include Submit Button* and click OK. A form including fields and prompts will appear on the page.

6. Select the Submit button. Change its *Text* property to “Return”, and select “return” in its *Action* property pulldown.

Create the Home Page and Test the Application

This section creates the home page. This page is available to non-logged-in users so you would decorate it with elements that identify the application. The design you develop here is very stripped down because it will be used only to demonstrate security principles. Although you will define security later in this practice, you will test the application, although the security at the end of this section.

1. From the faces-config.xml diagram, double click /home.jsp and click Next in the wizard until you reach the HTML Options page. Change *Title* to “HR Application Home” and click Finish.

2. In the home.jsp visual editor, drag and drop a PanelPage component from the ADF Faces Core page of the Component Palette. Change its *Title* property to “HR Application Home”.

3. Drag a CommandButton from the Component Palette to the editor and drop it when the af:panelPage component is highlighted with a surrounding box.

4. Change the *Text* property of this button to "Login" and the *Action* to "browse". Clicking this button at runtime will navigate to the browse page but we will define security settings so if the user is not logged in, the login page will appear instead of the browse page.
5. Click Save All. You can now test run these pages. With the home.jsp page active in the editor, click the Run (green arrow) icon in the top toolbar.
6. The Embedded OC4J Server will start and the home page will be displayed in your web browser.
7. Click Login and the browse page will appear. The default action for ADF BC is to query records so you will see one page of records displayed.
8. Scroll through the record set (using the Next and Previous links and the record pulldown) to try out record navigation. Click the sort buttons at the top of the columns to test sorting.

   Additional Information: Sorting was enabled by the option you set in the field selection dialog that appeared when you dropped the data control onto the page. Record navigation and sorting are native features of the ADF Faces table component.
9. Select a radio button to the left of one of the employee records and click Edit. The edit page will display with the details of the selected employee record.
10. Change the first name of that record and click Return. The browse page will load again and the name change will be reflected in the display. You have not added Commit or Rollback features to the application so these changes are only cached in the ADF BC model layer and will not affect the database.
11. Click New. The edit page will load again but all fields will be blank (the result of the Create operation in the ADF BC code).
12. Fill out the fields with the required indicator ("*" before the prompt). You can select a HireDate value using the calendar date picker by clicking the icon to the right of the date field.
13. Click Return when all required fields are filled in. The new record will be displayed and selected in the browse page.
14. Close the browser and stop the Embedded OC4J Server by clicking the red square (Terminate) icon in the Log window at the bottom of the IDE screen.

WHAT DID YOU JUST DO?

You just created three pages that demonstrate browse, sort, modify, and add operations for the EMPLOYEES table records in addition to displaying a home page. This application is sufficient for the purposes of demonstrating security principles.

You might want to take this application to the point where records would be saved to the database. To do this, you would drop Commit and Rollback operations (each as an ADF Command Button) from the Operations node in the Data Control Palette (the top-level node under AppModuleDataControl, not under EmployeesView1) into the af:tableSelectOne node of the browser page.

You can add delete capabilities to this page by dropping a Delete operation as a button from the EmployeesView1\Operations node to the same af:tableSelectOne. When the user selects a record and clicks Delete, the record will be deleted from the ADF BC cache. Committing that change would delete the record from the database.

II. Configure Container Security

Now that you have a basic application to test, you can practice setting up security. Although you can set up application security before container security, it is best to set up the container security first. That way, the user repository will be ready so you can test the application right after you code security hooks into the application. In this phase, you set up the OC4J container with a user repository that will be used to both authenticate users and provide the list of roles granted to them.

Define Users and Passwords

Each user that can log into to system needs to be defined in the jazn-data.xml file. This section defines users and passwords.

1. Be sure the embedded server is stopped (indicated by no red square icon in the Log window).

2. With the cursor selecting a file in the workspace, select **Tools | Embedded OC4J Server Preferences**. The Embedded OC4J Server Preferences for Employees dialog will appear. This dialog allows you to control many aspects of the internal OC4J server within JDeveloper, including the security parameters you need to set.

3. Expand the Current Workspace\Authentication\Realms nodes in the preference tree. There should already be a realm within this node called jazn.com. If not, click *New*, and in the Create Realm dialog specify the *Name* as "jazn.com". Click OK.

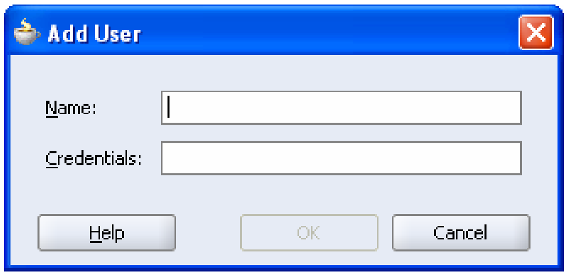
   Additional Information: A *realm*, also called a *protection domain*, is just a namespace within the jazn-data.xml. An application can specify that authentication and authorization should take place for users in a specified realm. The default realm is called jazn.com.

4. Expand jazn.com. Two nodes will appear beneath jazn.com in the navigator—Users and Roles.

5. Select the Users node and click Add.

   Additional Information: JDeveloper stores some credential information of its own within the jazn.com realm. These will display as existing users in the dialog with names like "DataBase_User_5-S5QS0zCT7P7WqAyij6RN4NL8FtF2ZF". These entries can be ignored and should be left unchanged.

6. The Add User dialog shown next will appear. We are using the email address from the EMPLOYEES table as usernames, so enter the *Name* as "SKING" and, for *Credentials* (the password), enter "emp**"**. Click OK.



Caution

JAZN usernames and passwords are case sensitive.

7. Repeat steps 5 and 6 for CDAVIES, AHUNOLD, and SKUMAR. Use the same password for simplicity. Leave the Embedded OC4J Preferences for Employees dialog open.

Set up the User Roles

Now that the user entries are created, you can create the three roles and assign users to those roles:

1. Select the Roles node under the jazn.com realm and click Add.

2. In the Create Role dialog fill in *Name* as "admin**".** As before, case sensitivity is important. Click OK.

3. Repeat step 2 for roles named "user" and "manager".

4. Select the admin role in the *Roles* list, click the Member Users tab, and using the shuttle control, assign SKING to the admin role. In the same way, assign CDAVIES to the user role and AHUNOLD to the manager role. Do not assign SKUMAR to any role.

   Additional Information: Multiple users can belong to a role and a user can belong to zero, one, or more roles, but this example is kept simple to demonstrate security principles.

Note

You will notice that you can also associate member roles within a role, so you could make admin a member role for manager. In that situation, any user assigned to admin would automatically have privileges assigned to the manager role as well.

5. Click OK to dismiss the OC4J preferences dialog.

Browse the Security Data

Now that you have defined users and roles, you can take a look at the jazn-data.xml file that stores this information. As mentioned, the normal location for the jazn-data.xml file is the j2ee/home/config directory under your OC4J installation. However, this can be overridden if required. In fact, when you define security information for an application workspace through the JDeveloper editor, you are overriding the files in the J2EE home configuration. JDeveloper needs each application workspace to have its own version of the file, so rather than updating the central copy in j2ee\home, a workspace specific file, prefixed with the workspace name, is created in the workspace root folder and used when running the application from JDeveloper's Embedded OC4J Server

1. Select the application workspace node (Employees) in the navigator and select **File | Open.** You should see the Employees.jws workspace file in this directory.
2. Select the file Employees-jazn-data.xml and click Open. Identify the realm, user, and role information you entered in the editor.

   Additional Information: Depending on your JDeveloper version, the passwords may be encrypted or may be shown prefixed with an exclamation point "!" inside the credentials element of the user element. The "!" character signals the server to encrypt them the first time that the application is run. If the credentials properties are already encrypted, you will see a (903) prefix to the encrypted string.
3. In the same way, open the Employees-oc4j-app.xml file. You will see that the jazn element identifies the Employees-jazn-data.xml file and defines the provider as XML. If you wanted to use a different source of credentials, you would change these attributes as described earlier.
4. Close the editor windows for these two files. (Click the "X" icon in the editor's tab for each file.)

WHAT DID YOU JUST DO?

Using the interfaces provided by the JDeveloper IDE, you configured security for the internal J2EE container, the Embedded OC4J Server, which JDeveloper uses to run J2EE code. You added a set of valid users and passwords as well as roles for the users. These roles match the names of the logical roles that you will be using within the application so no further mapping will be required. When you test security, later in this practice, you will test these credentials. In addition, you also examined the JAZN file and OC4J file that points to it. This demonstrates how the security is configured for a workspace and how the code that defines users and roles appears.

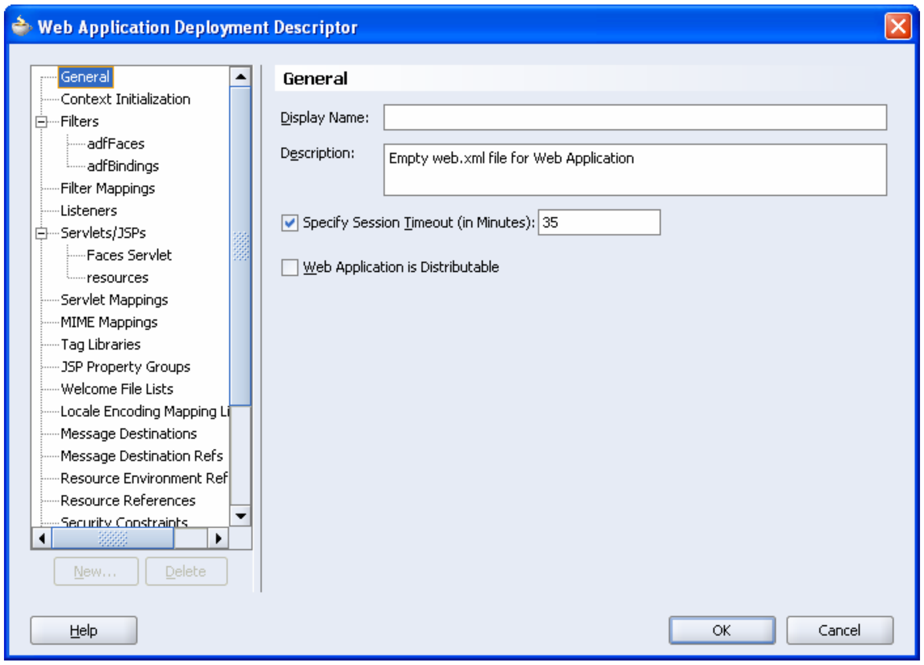
III. Define Application Security Settings

In this phase of the practice, you will add security to the application. All work in this phase will take place in the ViewController project's web.xml file.

Set up the Logical Application Roles

Remember that we need three logical roles for the application: admin, manager and user. Use these steps to create these roles.

1. On the web.xml node in ViewController\Web Content\WEB-INF, select **Properties** from the right-click menu. The Web Application Deployment Descriptor dialog will appear as shown in Figure 6.
2. Select the Security Roles node in the navigator and click Add.
3. In the Create Security Role dialog, enter the *Security Role Name* as "admin" (pay attention to case sensitivity) and *Description* as "Administrative users". Click OK.
4. Repeat step 2 and 3 for the manager (Management users) and user (Employees and other limited-access users) roles. Click OK to dismiss the descriptor dialog.
5. Click Save All.



General

Display Name:

Description: Empty web.xml file for Web Application

[✓] Specify Session Timeout (in Minutes): 35

[ ] Web Application is Distributable

Figure 6. Web Application Deployment Descriptor dialog

Define the Protected URLs with Security Constraints

The security model of the application divides the pages into several sets depending upon the user groups to which each page is available:

- **Non-logged in users** The initial home page is available to users who are not logged into the application.
- **Logged in users** The browse page is available only to logged-in users.
- **Managers and administrators** The edit page is available only to managers and administrators.

The access restriction to pages is accomplished by partitioning different functional areas of the application into different physical directories under the web root. You have already placed the pages in various physical directories. The next step is to apply permissions for specific roles to the relevant URL patterns that specify each directory. In this section, you will apply these protections.

1. Open the editor for the web.xml again.
2. In the navigator, select the Security Constraints node and click New. The new constraint will be added and the dialog will now look something like the screenshot in Figure 7.
3. Click Add and in the Create Web Resource Collection dialog, enter the *Web Resource Name* as "AdminZone**"**. Click OK.
4. Click the Authorization tab at the top of the dialog and check the admin and manager roles as shown in Figure 8.
5. Click the Web Resources tab and select AdminZone in the *Web Resource Collections* area. Click Add next to the lower set of tabs.
6. In the Create URL Pattern dialog, enter the *URL Pattern* as "faces/pages/admin/*". Click OK.

   Additional Information: This setting protects the AdminZone pages so that when the application requests the container to display a page in this location—for example, the "/pages/admin/editEmp.jsp" page— the container will check if the user is already authenticated. If the user is not yet authenticated, he or she will be challenged to authenticate with a login screen before proceeding. If the user is already authenticated, the container will check the user's roles against the authorized roles for this resource—admin and manager in this case. It will allow the user access to the page only if the user is a member of one of those roles.

Notice that the URL pattern we use here does not exactly mimic the file system directory location of the page file under the context root of the application. For example, the actual location of the edit page is ../pages/admin/editEmp.jsp. The extra "faces/" prefix is required because all page requests are going through the JSF servlet, which is triggered by the use of this faces prefix. This prefix is specified web.xml. Click the Servlet Mappings node in the navigator to see this mapping. Return to the Constraint node.

As you would expect, the trailing "*" is a wildcard matching any page in that directory. You could add the exact URL of each page to protect them individually. However, partitioning the application into directories and using wildcard mappings is simpler.

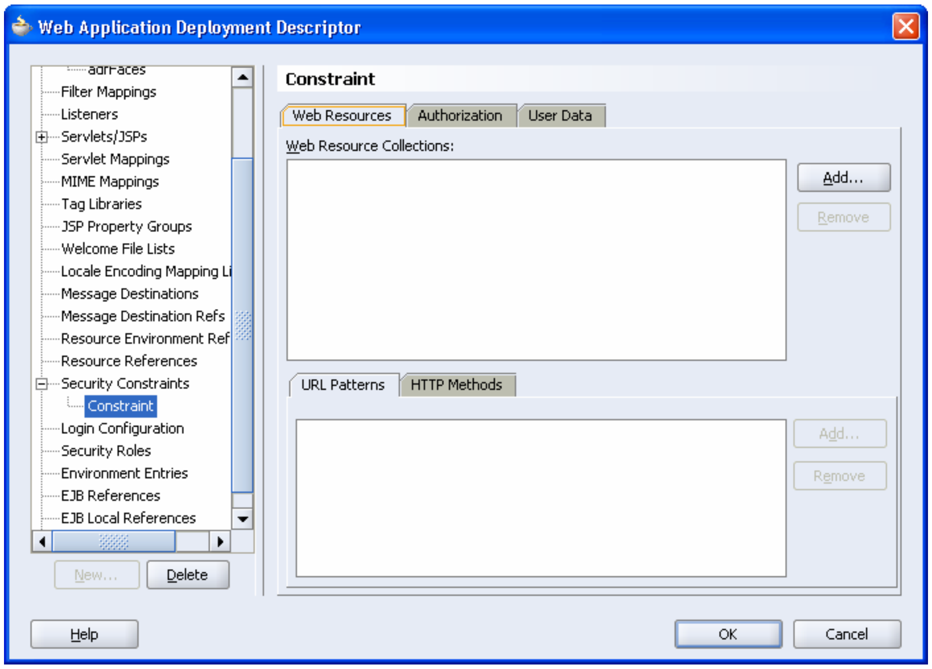
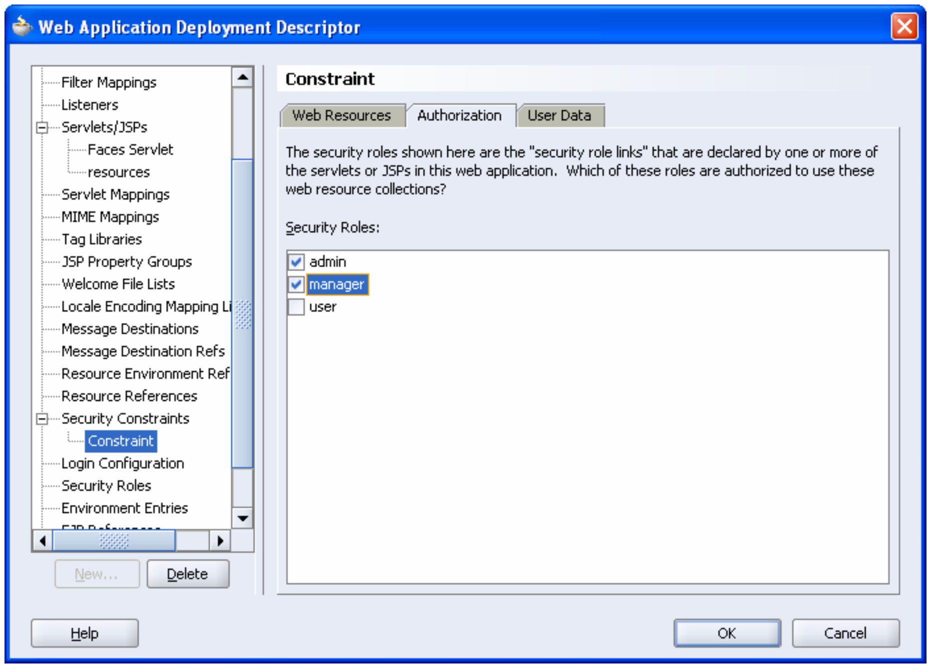


Figure 7. Constraint page of the Web Application Deployment Descriptor dialog



The security roles shown here are the "security role links" that are declared by one or more of the servlets or JSPs in this web application. Which of these roles are authorized to use these web resource collections?

Security Roles:

- ✓ admin
- ✓ manager
- ☐ user

Figure 8. Authorization tab of the Constraint page

7. Repeat steps 2–6 to create another security constraint called “UserZone” which is available to all three roles and protects the URL pattern “faces/pages/*”.

   Additional Information: J2EE container security supports a flat security model and has no provision for creating a hierarchical security rules. This is why we define separate security constraint elements for the application. If you had specific pages that only the administrator would view, you would create another constraint that only authorized the admin role. A separate URL pattern would restrict those pages to only the admin users.

8. Click OK to dismiss the dialog. Click Save All.

CONSTRAINT ORDER IS IMPORTANT

When applying security, always define the most restricted access first, in this case, the AdminZone constraint. If a user is a member of several roles, she or he will be authenticated at the most powerful level. The container evaluates the security constraints in the order that they are defined in the web.xml; using the wrong ordering could result in a user being authenticated at a lower privilege level than which they are entitled. For example, in this application, we defined AdminZone above UserZone. To verify the generated XML, you may open the web.xml file and browse for the security-constraint tags.

The web.xml editor has no provision for re-ordering the constraints. If you make a mistake in the ordering drag and drop the nodes in the Structure window or cut and paste source code to change the order.

Test the Constraints

Although you have not fully defined security yet, you can test the constraints.

1. In the navigator, on the pages/browseEmp.jsp file, select Run from the right-click menu. The server will start and the browser will attempt to connect to that page.
2. A default login dialog will appear because you defined the security constraint for the “faces/pages/*” URL pattern which contains the browseEmp.jsp page. Enter the *User name* as “SKING” and *Password* as “emp2” (an incorrect password). Click OK.
3. The login dialog will reappear because you used an incorrect password. Enter the correct password (“emp”) and click OK.
4. The browse page will appear. Notice that the end of the URL is “/faces/pages/browseEmp.jsp”. This designates the restricted directory. Change this ending to “faces/pages/admin/editEmp.jsp” and press ENTER to load the edit page.
5. Close the browser. Run browseEmp.jsp again. Log in as SKUMAR (password of “emp”). You will receive a security error because SKUMAR is not enrolled in any group that has access to that page.
6. Close the browser and stop the server.

Set Up a Login Page

When using container-managed security, you can use several approaches to authenticate the user. The two most common approaches are: basic authentication where you allow the browser to generate a default popup login dialog (as demonstrated in the preceding section); and a custom login form, which is coded as part of the application. This section creates a custom login page so that you could (as a follow up to this practice) add layout elements to fit in with the rest of the application.

For the login page, you will be using a normal JSP page, not a JSF JSP of the type that you have been using so far. The normal JSP file is required for two reasons. First, the container calls the login page and, in the process, bypasses the various servlet mechanisms that an ADF Faces page needs to work correctly. Second, JSF cannot be configured to call the special URL that the container uses to accept authentication requests.

1. On the ViewController project node, select New from the right-click menu. Select the Web Tier\JSP category and double click the JSP item. The Create JSP Wizard dialog will appear. Click Next if the Welcome page appears.
2. In the Step 1 page, set the name to **“**login.jsp” and add “\security” to the *Directory Name* so it reads “APP_HOME\ViewController\public_html\security”. Substitute for “APP_HOME” the exact drive and directory you specified when creating the application.
3. Click Next. Click Next on the Step 2 page. On the Step 3 Libraries page, select “All Libraries” in the *Filter By* pulldown, and move the JSTL Core 1.1 library to the *Selected Libraries* list.

Additional Information: The *JSP Standard Tag Library (JSTL)* is a set of tags that supplies iterative and conditional control as well as and other functionality. It is defined by the JSP specification, and is considered easier to read and easier to use than native JSP tags such as scriptlets and expressions. You will be using JSTL expressions to detect if the user is on the first or subsequent login attempts. JSTL tags allow you to write simple procedural code within the body of your JSP page. You will write JSTL code to conditionally display a message about invalid login credentials if the user has more than one unsuccessful login attempt.

4. Click Next. In the Step 4 HTML Options page of the wizard, set the *Title* to "Login". Click Finish. The page will be created and will display in the visual editor.

5. You can now lay out the page. From the HTML Forms page of the Component Palette page, drop a Form tag onto the page. The Insert Form dialog will appear.

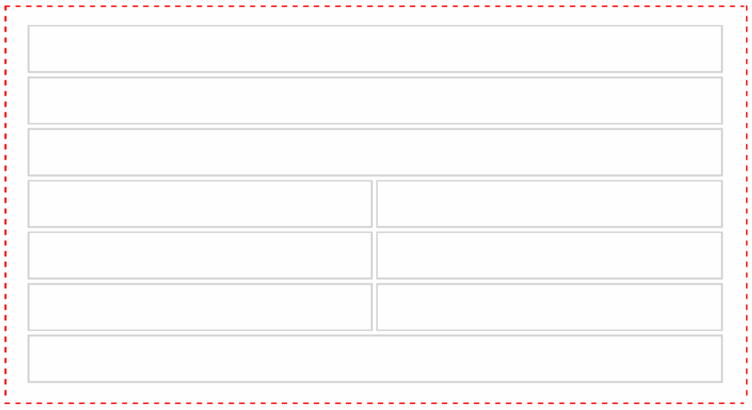
6. In this dialog, enter the *Action* as "j_security_check" and select the *Method* as "post". Click OK. A form box will appear on the page.

   Additional Information: The *j_security_check* action is a magic action that every J2EE container is required to implement for developers to use. This will hook into whatever authentication mechanism is currently selected for the container. As you will see, some special field names are also required to complement this action.

7. From the HTML Common page of the Component Palette, drop a Table inside the Form tag area that has just been created. Set *Rows* as "7", *Columns* as"2", *Width* as "100" percent, *Header* as "None", and *Border Size* as "0". Click OK.

8. You will be using this table to lay out the various parts of the login screen such as the title, a button, and input fields. Start this process by group selecting the two cells in the top row of the table, then choose **Table** | **Merge Cells** from the right-click menu. This will combine the cells into a single two-column cell, which we will use to hold the title.

9. Repeat the cell-merging process for the second, third, and last rows of the table. The end result in the visual editor should look like this.



10. Click inside the top row and enter the text "Log in to the HR Application". Select "Heading 2" from the formatting pulldown in the editor toolbar.

11. Click inside the second row and click the Horizontal Rule component in the Component Palette. A line will appear in the editor. Add another horizontal rule to the last row of the table.

12. In the left-hand cell of the fourth row, enter "* User ID"; in the fifth row cell left-hand cell, enter "* Password".

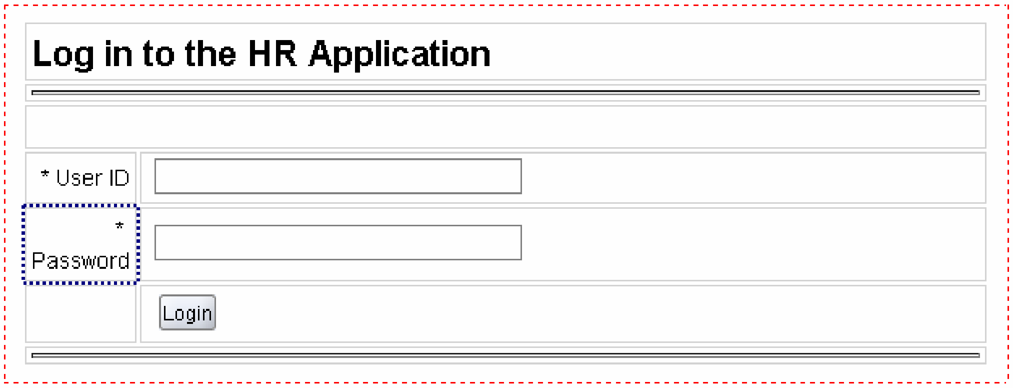
13. From the HTML Forms page of the Component Palette, drag a Text Field into the right-hand cell of the fourth row. In the Insert Text Field dialog, enter the *Name* as "j_username". Click OK.

14. Add a Password Field to the right-hand cell of the fifth row. Enter the field's *Name* as "j_password" and click OK.

    Additional Information: These two field names are also magic values that the container expects when a j_security_check form is submitted. The Password Field will hide the characters the user types.

15. Drop a Submit Button into the right-hand cell of the sixth row. Set its *Name* property to “login” and its *Value* property to “Login”. This will be used to submit the form to the container so it can validate the credentials. Click OK.
16. Click in the User ID prompt cell. In the Structure window, select the td node above the User ID prompt. Set its *Width* property to “10%”. This will resize the column for all three rows. Also set its *Align* property to “right” so the prompt is right aligned. (This will more closely emulate the layout of an ADF Faces field with its prompt.) Repeat the alignment setting for the password prompt’s td tag.

    Additional Information: These settings modify the default behavior so the prompts are right-aligned and stay close to the fields when the browser window is widened or narrowed. The login form should look something like this:


Log in to the HR Application

* User ID	
* Password	
	Login

17. Click Save All. Although the security functionality is not complete and navigation is not yet defined, you can run this page to check its appearance.
18. Close the browser and stop the server when you are finished.

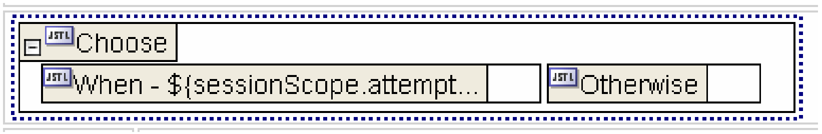
Detect Multiple Login Attempts

As a final polish to the login page you will add some JSTL code to detect if the user has visited the login page more than once; this would indicate that a login attempt failed and that an error message should be displayed.

1. Switch to the JSTL 1.1 Core page of the Component Palette and drag a Choose tag into the empty third row of the table on the login screen. The Insert Choose dialog will appear.

    Additional Information: The JSTL `choose` tag allows you to embed a conditional test in the page. Each test condition starts with a `when` tag. It provides similar functionality to an IF…THEN…ELSIF…ELSE…END IF structure in PL/SQL.

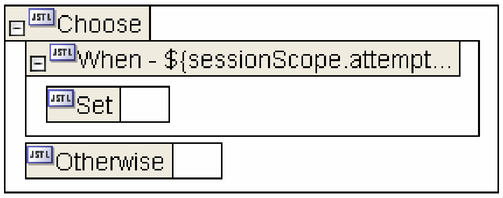
2. Click New to add a `when` condition and enter the *Value* as “${sessionScope.attemptsAtLogin == null}”. This statement checks if a session variable, attemptsAtLogin, does not exist.
3. Check the *Add c:otherwise* box and click OK. The Choose tag will appear in the visual editor as shown next. (This tag will not be displayed at runtime.)



Additional information: JSTL supports both `if` statements and `choose` statements. The `if` statement only allows testing of a single condition and does not provide an else option. In this situation, a single outcome is not sufficient, so `choose` is best.

4. Click the Source tab to view this code. The sessionScope area contains values you set throughout the session. If this is the first time that the login screen has been displayed, the attemptsAtLogin variable will be not be assigned (and therefore will have a null value). You will want to initialize it so that next time around (if the login fails) the `c:otherwise` statement will be executed.

5. You can set session variables from JSTL using the Set tag. In the Design tab, drag a Set tag from the Component Palette and drop it into the empty box to the right of the When tag. This will nest the Set tag within the When tag and the choose tag will appear as follows:



6. With the Set tag selected, set the *Var* property as “attemptsAtLogin”, *Scope* as “session” (using the pulldown), and *Value* to “0” (zero).
7. If the user returns to the screen for a second time, the `attemptsAtLogin` session variable will have been set to 0, so the `c:otherwise` statement will be triggered. In this tag you will print out an error message and maintain a count of how many attempts the user has actually made. Click inside the Otherwise tag and type the error message “Error: Invalid User ID or Password.”.
8. Select the text and, with the visual editor tools, set the font color to red. Select the word “Error” and click the “B” button to boldface the word.
9. Drag a Set tag from the Component Palette drop it after the error message in the Otherwise tag. In the Property Inspector, set the Set tag’s *Var* as “attemptsAtLogin”, and the *Scope* to “session” as before; this time, set the *Value* as this:

   ```
   ${sessionScope.attemptsAtLogin + 1}
   ```

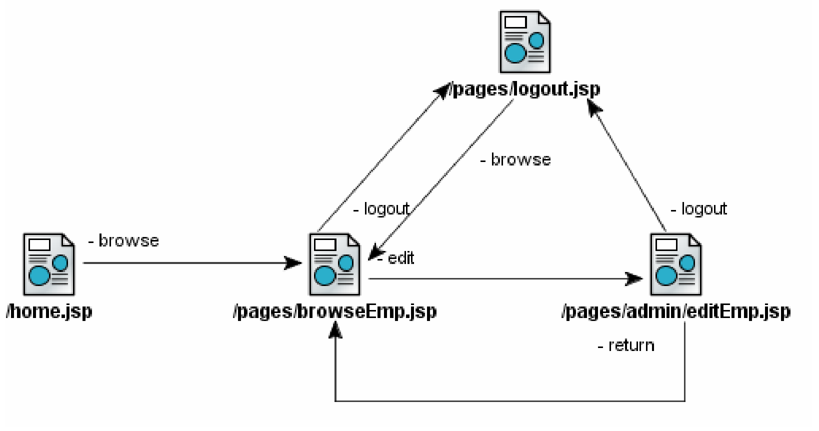
   Additional Information: JSTL allows you to apply simple calculations like this within your markup. In this case, you are incrementing the value that is currently held in the `attemptsAtLogin` variable, maintaining it as a running count of login attempts. Since the scope is set to “session” for both instances of setting the variable, the value will carry through between page reloads.
10. Switch to the Source tab in the visual editor and check that the code for this row in the table reads as follows (although the code formatting may vary):

   ```
   <tr>
     <td colspan="2">
       <c:choose>
         <c:when test="${sessionScope.attemptsAtLogin == null}">
           <c:set var="attemptsAtLogin"
                  scope="session"
                  value="0"/>
         </c:when>
         <c:otherwise>
           <font color="#ff0000">
             <strong>Error:</strong>
              Invalid User ID or Password.
           </font>
           <c:set var="attemptsAtLogin"
                  scope="session"
                  value="${sessionScope.attemptsAtLogin + 1}"/>
         </c:otherwise>
       </c:choose>
     </td>
   </tr>
   ```
11. Correct any errors. Click the Design tab. Click Save All.
12. Run this page again. Click the Login button and you will see the error message. You haven’t set up navigation for this button, so after viewing the error message, close the browser and stop the server.

Set Up a Logout Page

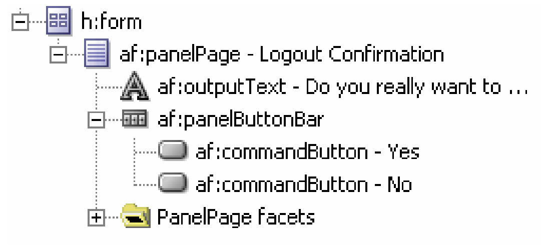
Now that you have a login page for the application you also need to provide a way to logout. Logging out of a web application requires invalidating the session using Java code. Invalidating the session throws away all of the data saved at the session level, including the authenticated credentials and variables (such as attemptsAtLogin). As soon as the session is invalidated, the OC4J container will start over with a blank state and the user will have to re-authenticate in the same way as the initial login. This section implements the session invalidation and creates a logout confirmation page to check that the user really wants to log out.

1. Open the faces-config.xml by clicking its tab in the editor. In the Diagram tab, add a JSF Page and change its name to “/pages/logout”.
2. Define a JSF Navigation Case from browseEmp.jsp to logout.jsp and name it “logout”. Create another navigation case from editEmp.jsp to logout.jsp and call it “logout”.
3. Create one more navigation case from logout,jsp to browseEmp.jsp and call it “browse”. The diagram should look something like this:



4. Click Save All.
5. Open the browseEmp.jsp editor and from the ADF Faces Core page of the Component Palette, drop a CommandLink into the menuGlobal facet in the top-right corner of the page. Change the *Text* property to “Logout”, set the *Action* property to “logout” (the navigation case you just defined), and set *Immediate* to “true” (so validation is not performed on required fields with missing data when Logout is clicked).
6. Select the new tag in the Structure window and copy it to the clipboard (CTRL-C).
7. Open the editEmp.jsp page and select the menuGlobal facet in the visual editor. Paste from the clipboard (CTRL-V) to add the tag,

   Additional Information: The logout link and its properties are the same in both pages and you would want to work this button into a template for an application that contains multiple pages.
8. In the JSF Navigation Diagram, double click ‘pages/logout.jsp to start the wizard. Click Next until you reach the HTML Options page. Set *Title* to “Logout Confirmation” and click Finish.
9. From the ADF Faces Core page of the Component Palette, drop a PanelPage component onto the page. Click “Title1” and change the *Text* “Logout Confirmation”.
10. Drop an OutputText component onto the Panel Page container and set its Value property to “Do you really want to log out?”.
11. Drop a PanelButtonBar next to the message. It will appear under the message. Drop two CommandButton components inside the af:panelButtonBar. Set the *Text* of the first to “Yes**”** and the *Text* of the second to “No”. The Structure window will now contain the following:



12. The No button will return the user to the browse page (regardless of where the user clicked the Logout link), so set its *Action* to “browse**”**.

13. We need to add code to the Yes button to invalidate the session and navigate to the home page. This requires creating a Java backing bean class file. Double click the Yes button in the visual editor. The Bind Action Property dialog will appear.

    Additional Information: The *backing bean* class file allows you to code Java logic that will execute upon a user interface component event (such as when the user clicks the Logout link).

14. Click New to display the Create Managed Bean dialog. Set the *Name* as “backing_logout” and the *Class* “hr.view.backing.Logout”. This class does not yet exist, so check the *Generate Class If It Does Not Exist* checkbox. Click OK. The file will be created.

15. In the Bind Action Property dialog, set the *Method* to “logoutButton_action” and click OK to create the method in the backing class file and open the file in the editor.

16. In the logoutButton_action() method, set the code to the following to invalidate the session and hence log the user off as follows:

```
public String logoutButton_action() throws IOException
{
  ExternalContext ectx = FacesContext.getCurrentInstance().getExternalContext();
  HttpServletResponse response = (HttpServletResponse)ectx.getResponse();
  HttpSession session = (HttpSession)ectx.getSession(false);
  session.invalidate();
  response.sendRedirect("../home.jsp");
  return null;
}
```

    Additional Information: The first line of code creates a FacesContext object for the current instance. The *ExternalContext* is an object which provides direct access to the HTTP Session and response objects which are needed to logout and return to the home page respectively. The next line creates a response object from the ExternalContext object. The next line creates a session object from the FacesContext object. The next line invalidates the session (effectively logging off). Finally, the code uses the response object to direct the browser to re-display the protected home page. Because the existing account has been logged off by the session invalidation, the login page will be presented to force another logon.

17. Wavy red lines will appear under the classes that cannot be located. This indicates the need for import statements. Import these classes (IOException, FacesContext, ExternalContext, HttpServletResponse and HttpSession) by selecting the name of the class from the pulldown available on the quick fixes (light bulb) icon in the left gutter of the code editor next to each line.

    Additional Information: You may alternatively import the classes using the ALT-ENTER keypress after placing the mouse cursor in the class name. (Use javax.faces.context.FacesContext, javax.faces.context.ExternalContext, javax.servlet.http.HttpServletResponse and javax.servlet.http.HttpSession if given choices.) The import section should appear something like the following:

```
import java.io.IOException;
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpSession;
```

18. Click Make in the toolbar to compile the file. Fix any errors. Click Save All.

Set Up the Application Entry Page

You need to have the OC4J container automatically load the home page when a user accesses the application's context root (http://host:port/emp). You will set up the home page as a welcome page for the application. *Welcome pages* are a servlet mechanism whereby if the application root is accessed directly, without a page being specified, the servlet knows what default page to supply. First, you will set up the context root as "emp".

1. Double click the ViewController project in the navigator to display the project properties. Select the J2EE Application node in the properties navigator and change both the *J2EE Web Application Name* and *J2EE Web Context Root* to "emp".

   Additional Information: The *J2EE Web Context Root* is the setting that defines the URL that will be used to access the application once it is deployed. The *J2EE Web Application Name* is used internally in the deployment descriptor and configuration files.

2. Click OK. On the web.xml file under the ViewController\Web Content\WEB-INF node, select Properties from the right-click menu.
3. Select the Welcome File Lists node in the navigator and click New.
4. Click Add and enter the path "/faces/home.jsp" in the Create Welcome File dialog. Click OK.
5. Click OK. Click Save All.
6. Run home.jsp page under Web Content. Change the URL in the Address (Location) field of the browser to "http://host:port/emp" and press ENTER. The home page will load again because this context root URL sends control to the file defined in the Welcome File List node of web.xml.
7. Close the browser and stop the server.

Switch Security On

Even though all the roles, security constraints, and physical pages are in place to secure the application, you still need to switch it all on. This is, again, defined in the application's web.xml file:

1. Open the properties dialog for the web.xml and select the Login Configuration node in the navigator.
2. Select the *Form-Based Authentication* radio button. Enter "security/login.jsp" for both *Login Page* and *Error Page*.

   Additional Information: The implication of using the same page for both purposes is that if the first attempt to login fails the user will be able to try again immediately, rather than by having to navigate back to the login page. The JSTL code you added to the login page will detect that the user is trying again and will print the error message relating to the first failed attempt. If you don't specify an error page, the container will use a default authorization failed page.

3. Click OK, click Save All.

Caution

The form based authentication used here is only truly secure when used in conjunction with secure HTTP (HTTPS). This ensures that the login information will be encrypted when it is posted to the server. Without this additional layer of security, it is possible to obtain the login credentials by monitoring the network traffic between the browser and the server. For information on how to set up HTTPS refer to Chapter 10 of the Oracle Application Server Administrator's Guide—"Overview of Secure Sockets Layer(SSL) in Oracle Application Server".

Ensure That Security is Triggered

This section tests the application and makes another required setting to the redirect properties of some pages.

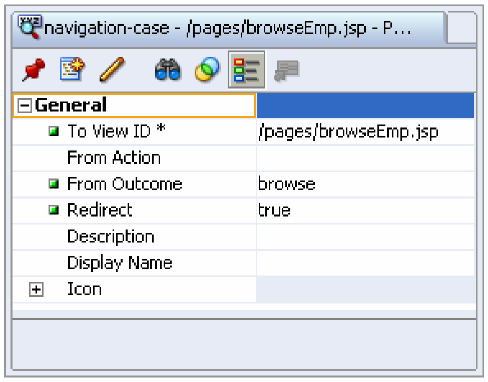
1. Run home.jsp to test the application. Click Login and you will forward to the logged-in home browse page without being challenged for your username and password.

   Additional Information: This is not what you would expect. The problem lies in the way that the need for authentication and authorization is detected by the container. You have secured certain paths in the application such as "/faces/pages/*", but if you look at the URL in the browser now you will see that it is still showing "faces/home.jsp" even though the protected "faces/pages/home.jsp" page shows in the browser.

   This situation reflects the way that JSF navigation works. A navigation rule will, by default, act as a forward within the container. A *forward* occurs when a servlet transparently rewrites the URL for a page to a new value, then sends the output of that new page to the screen without informing the browser. Therefore, the browser still shows the old URL. In this case, however, this is not good because we want to explicitly issue a request to a protected URL to trigger the container security mechanisms. Fortunately, we can alter the JSF navigation behavior to explicitly request the key protected pages and trigger security.

2. Close the browser and stop the server.

3. Open the faces-config.xml file in the editor and switch to the Overview tab. Select Navigation Rules and select the "/home.jsp" entry in the *From View ID* list. Select the *browse* Navigation Case (/pages/browseEmp.jsp) and, in the Property Inspector, set Redirect to "true" as shown here:



General	
To View ID *	/pages/browseEmp.jsp
From Action	
From Outcome	browse
Redirect	true
Description	
Display Name	
Icon	

   Additional Information: This value forces the browser to re-request the explicit URL defined by the browse rule and this will trigger the security check.

4. Click Save All.

5. Run home.jsp by selecting it in the navigator and clicking Run. Click Login. The login page will now display instead of the browse page.

6. Log in as SKING with an incorrect password of "emp2". Click Login. The error message will appear. Enter "SKING" and "emp" and click Login. The browse page will display.

7. Click Logout to view the Logout Confirmation page. Click No to return to the browse page.

8. Click Edit to navigate to the edit page. Click Logout to test the navigation to the logout page. This time, click Yes on the confirmation page.

9. The login page will display. This is the full login and logout cycle. You can try logging in with CDAVIES or AHUNOLD to test those accounts.

10. Close the browser and stop the server.

What Did You Just Do?

You defined the logical roles that are used within the application to judge the privilege level of each user and used those roles to protect certain pages based on their location under the web root. When a user is authenticated he or she will only be allowed to access pages in locations granted to the role he or she belongs to.

You also defined a custom login page for use by the J2EE container using the magic form name (j_security_check) and field names that inform any J2EE container that the form will call authentication and authorization services. You also added logic to display an error message that will display if the user has made more than one attempt to log in.

In addition, you implemented a logout page and its accompanying backing bean that invalidates the user's session. Once the session has been invalidated, the browser will be redirected back to the initial home page using a navigation rule that you created in the JSF navigation diagram. This allows the user to log in again, possibly using different credentials. Because the old session was invalidated, any state associated with it, such as the count of login attempts will be discarded.

Lastly, you switched on security for the application in the web.xml file and ensured that key navigation rules within the application trigger security checking by redirecting rather than forwarding.

IV. Add Security to the User Interface

This final phase of this practice examines how you can apply security attributes within the user interface to gain a deeper level of control over what the user sees and to access relevant security information such as the user ID of the authenticated user.

Note

Several of the tasks defined in this practice apply to multiple pages throughout the application. To help with the logical flow of the preceding practices we have gathered these tasks into this phase of development. However, as with the Logout link, the following tasks should be worked into the initial application template development process.

The Servlet API provides various calls that can be used to extract information such as the authenticated user (the `getRemoteUser( )` method in the HttpSession class) and the roles of that user. These API calls are straightforward to use in code, and the JSF ExternalContext object provides a shortcut for you to use. For example, the following code fragment will set the user ID in a code object, userName:

```
FacesContext ctx = FacesContext.getCurrentInstance();
ExternalContext ectx = ctx.getExternalContext();
userName = ectx.getRemoteUser();
```

Accessing this information within the structure of a page is a bit different from accessing it in backing bean code. JSF screens are all based on the use of Expression Language to bind attributes to useful information and therefore Expression Language is the key to accessing this information.

The JSF-Security Project

The *JSF-Security Project* is an open source effort to simplify the use of security information in JSF. The JSF-Security project extends the JSF expression language, adding a scope called "securityScope." The *securityScope* provides access to information such as the authenticated user ID and the roles of that user, from within standard expressions.

Note

The home page for this project is sourceforge.net/projects/jsf-security. Details about the expression syntax and functionality is available at jsf-security.sourceforge.net/.

The JSF-Security extension is not the only way to achieve Expression Language access to security information. You can alternatively create a JSF managed bean that exposes properties to Expression Language. However, using the JSF-Security extension in this practice means that you won't have to write code, and you can concentrate on the job of applying security rather than accessing it.

Install JSF-Security

To use the JSF-Security expressions, you will need to download the extension from sourceforge.net/projects/jsf-security using these steps.

1. Connect to sourceforge.net/projects/jsf-security. Click the "Download Java Server Faces Security Extensions" link.

2. On the next page click the “Download” link associated with the jsf-security package.
3. Find the latest release and select the zip file if you are working in Windows or tar.gz if you are on Linux, Macintosh, or Unix.

   Additional Information: The files with “-src” in the name include the source code as well as the final JAR file. You do not need this version unless you are interested in seeing how the code works.
4. On the next page, select a download mirror.
5. Once the archive is downloaded, open it and extract the jsf-security.jar. Drop this jar file into your application’s APP_HOME\ViewController\public_html\WEB-INF\lib directory using Windows Explorer or another file utility.

Add a User ID Indicator
The first task in this section is to display the name of the authenticated user at the top of each page in an af:outputFormatted component. The tag you will add could have been part of the template if security had been turned on earlier in the development process.

1. Open the /pages/browseEmp.jsp page from the navigator. Select the Logout link above the page title to expand the Structure window to the PanelPage facets area.
2. In the Structure window under the PanelPage facets, locate the infoUser facet. From the right mouse context menu choose **Insert inside infoUser | OutputFormatted** on this node.

   Additional Information: As mentioned, the af:panelPage layout component provides many facets, such as infoUser, for standardizing the layout of your screens. Of course, you do not have to use these facets to display this kind of information—it can be displayed anywhere. However, the facets provide a useful way of ensuring a consistent appearance for all of your pages.
3. Set the *Value* property of the new af:outputFormatted component to “Signed in as #{securityScope.remoteUser}”. At runtime this will display the authenticated user ID.
4. Before copying this to the edit page, run the browse page. You will be redirected to the login page. Log in as SKING, and verify that the logged in username appears in this component. Close the browser.
5. Select the af:outputFormatted node under infoUser in the Structure window and copy to the clipboard (CTRL-C).
6. Open /pages/admin/editEmp.jsp. Expand the Structure window in the same way to find the infoUser facet. Select that facet and paste from the clipboard (CTRL-V) to copy the customized af:outputFormatted component.
7. Click Save All.

Define Access for the admin and manager Roles
Within the TUHRA application, the salary and commission information and New button should be available only to authenticated users with the “admin” role. You have already configured the application to prevent unauthorized users from accessing the Edit page using a security constraint with a URL pattern. This section hides the New button on the browse page and the salary and commission fields on the edit page for all but admin users. It also hides the Edit and New button area for users in the user role. As a review, the security requirements of this application follow:

Privilege	user	manager	admin
View any employee record	Y	Y	Y
View salary and commission information	N	N	Y
Edit an employee record	N	Y	Y
Create an employee record	N	N	Y

1. Open browseEmp.jsp in the visual editor. Select the New button.

2. In the Property Inspector, select the *Rendered* property in the Property Inspector. This property can accept an expression that will evaluate to a Boolean. If it evaluates to "true", the component will display, otherwise it will not. Click the "Bind to data" button (database drum icon) in the Property Inspector's toolbar.
3. In the Rendered binding dialog, enter the following expression:
   ```
   #{securityScope.userInRole['admin']}
   ```
4. This will ensure that only users with the admin role (for example, SKING) will see this button. Copy the expression to the clipboard (CTRL-C). Click OK to dismiss the expression dialog. Open the editEmp.jsp file.
5. Select the Salary field in the visual editor. Click "Bind to data" in the Rendered property and paste the expression into the editor. Click OK. Repeat this step for the CommissionPct field.
6. In the browseEmp.jsp file, select the Edit button. In the Structure window, select the af:tableSelectOne component above the edit button tag.
7. Click "Bind to data" and in the Rendered binding dialog, enter the following expression:
   ```
   #{securityScope.userInRole['admin,manager']}
   ```
   Additional Information: This expression ensures that only admin and manager role users are able to see the components in this container. The af:tableSelectOne container also displays the radio group column and the prompt "Select and" in the table heading area. All of this will be hidden for users in the user role.
8. Click OK. Click Save All.
9. Run the home.jsp file, log in as CDAVIES (password of "emp"). CDAVIES belongs to the user role and cannot see the tableSelectOne components (radio group column, buttons, and "Select and" prompt).
10. Log out and log in as AHUNOLD (same password). This user belongs to the manager role and cannot see the New button but can see the Edit button. Select a row and click Edit. The Edit page will appear but the Salary and CommissionPct fields will not be displayed.
11. Repeat the logout and log in as SKING. You will see the New button because this user belongs to the admin role. Select an employee and click Edit. You will see the Salary and CommissionPct fields.
12. Close the browser and stop the server.

WHAT DID YOU JUST DO?

In this final phase for setting up security, you incorporated security details into the pages. You first added a security library that assists with JSF page control by adding a security scope that holds users and role information. You then added a user indicator to demonstrate how this security information could be available to the components on the page. You also set up security for a menu tab so only the admin role will see the Reference tab. Lastly, you modified an expression on the af:tableSelectOne component of the Search page so the buttons for adding, modifying, and deleting employee records are only shown to managers and administrators.

This hands-on practice has shown how to set up container-based security and how you can access security information through Expression Language. EL can be used both for the purposes of display, as in the case of the authenticated user, and also to dynamically control access to areas of the screen using the combination of expressions and the *Rendered* property.

Conclusion

This white paper discussed the OC4J security feature, JAZN and how it uses standard Java features (JAAS) to provide authentication and authorization services to web applications. It explained how JAZN can use either an LDAP or an XML user credentials store to authenticate and authorize users.. It also provided steps for implementing security in a JSF application using JDeveloper 10g. The sample application contains examples of all the major features of an implementing security at the page and field levels including setting up the roles and user accounts, creating login and logout pages, turning JAZN security on for the application, and displaying the logged-in user. These techniques should prove useful when you develop web applications using JDeveloper 10g.

About the Authors

Peter Koletzke is a technical director and principal instructor for the Enterprise e-Commerce Solutions practice at Quovera, in Mountain View, California, and has worked in the database industry since 1984. Peter has presented at various Oracle users group conferences more than 170 times and has won awards such as Pinnacle Publishing's Technical Achievement, Oracle Development Tools Users Group (ODTUG) Editor's Choice, ECO/SEOUC Oracle Designer Award, the ODTUG Volunteer of the Year, and NYOUG Editor's Choice. He is an Oracle Fusion Middleware Regional Director and Oracle Certified Master. Peter is coauthor of the Oracle Press books: *Oracle JDeveloper 10g for Forms & PL/SQL Developers* (with Duncan Mills); *Oracle JDeveloper 10*g *Handbook* and *Oracle9*i *JDeveloper Handbook* (with Dr. Paul Dorsey and Avrom Roy-Faderman); *Oracle JDeveloper 3 Handbook*, *Oracle Developer Advanced Forms and Reports*, *Oracle Designer Handbook*, *2nd Edition*, and *Oracle Designer/2000 Handbook* (with Dr; Paul Dorsey).

Quovera is a business consulting and technology integration firm that specializes in delivering solutions to the high technology, telecommunications, semiconductor, manufacturing, software and services, public sector, and financial services industries. Quovera deploys solutions that deliver optimized business processes quickly and economically, driving increased productivity and improved operational efficiency. Founded in 1995, the company has a track record of delivering hundreds of strategy, design, and implementation projects to over 250 Fortune 2000 and high growth middle market companies. Quovera's client list includes notable organizations such as Cisco Systems, ON Semiconductor, New York State, Sun Microsystems, Lawrence Livermore National Laboratory, Seagate, Toyota, Fujitsu, Visa, and Cendant. www.quovera.com.

Duncan Mills is a Java evangelist and Consulting Product Manager at Oracle, specializing in JavaServer Faces, the Oracle ADF Framework and related J2EE technologies. He has been working with Java and Oracle Products in a variety of application development and DBA roles since 1988. For the past 13 years he has been working at Oracle, currently working as part of the JDeveloper Java IDE development team. Duncan is a frequent presenter at Oracle User Groups and Java Events around the world achieving awards such as Best Oracle Presentation from the annual conference of the UK Oracle User Group and Contributor of the Year from ODTUG. He is an Oracle Technology Network ACE and publishes frequently on the OTN site as well as in the Oracle Development Tools User Group Journal and the Java Developers Journal.

For nearly three decades, **Oracle**, the world's largest enterprise software company, has provided the software and services that let organizations get the most up-to-date and accurate information from their business systems. With over 275,000 customers—including 98 of the Fortune 100—Oracle supports customers in more than 145 countries. For more information about Oracle, visit www.oracle.com