SAP NetWeaver '04 Security Tutorials

Protecting Access to a J2EE-Based Application Using UME Permissions

Document Version 1.00 – March 2, 2005

SAP AG
Neurottstraße 16
69190 Walldorf
Germany
T +49/18 05/34 34 24
F +49/18 05/34 34 20
www.sap.com

Disclaimer

SAP code or applications samples and tutorials are NOT FOR PRODUCTION USE unless specifically noted. You may not demonstrate, test, examine, evaluate or otherwise use them in a live operating environment or with data that has not been sufficiently backed up.

You may not rent, lease, lend, or resell SAP code or application samples and tutorials.

Documentation in the SAP Developer Network

You can find this documentation at the following Internet address:
`sdn.sap.com`

Typographic Conventions

Type Style	Description
Example Text	Words or characters quoted from the screen. These include field names, screen titles, pushbuttons labels, menu names, menu paths, and menu options. Cross-references to other documentation
Example text	Emphasized words or phrases in body text, graphic titles, and table titles
EXAMPLE TEXT	Technical names of system objects. These include report names, program names, transaction codes, table names, and key concepts of a programming language when they are surrounded by body text, for example, SELECT and INCLUDE.
`Example text`	Output on the screen. This includes file and directory names and their paths, messages, names of variables and parameters, source text, and names of installation, upgrade and database tools.
`Example text`	Exact user entry. These are words or characters that you enter in the system exactly as they appear in the documentation.
`<Example text>`	Variable user entry. Angle brackets indicate that you replace these words and characters with appropriate entries to make entries in the system.
`EXAMPLE TEXT`	Keys on the keyboard, for example, *F2* or *ENTER*.

Icons

Icon	Meaning
	Caution
	Example
	Note
	Recommendation
	Syntax

Additional icons are used in SAP Library documentation to help you identify different types of information at a glance. For more information, see *Help on Help → General Information Classes and Information Classes for Business Information Warehouse* on the first page of any version of *SAP Library*.

Contents

Protecting Access to a J2EE-Based Application Using UME Permissions

Task

In this tutorial, you will include access protection to the quick car rental application that is provided with the SAP NetWeaver Developer Studio. In this application, a JSP and servlet serve as the frontend client. The business logic is implemented using EJBs.

You will use a programmatic approach using UME permissions and actions so that only certain users are allowed to perform certain tasks. The following access protection rules apply:

- All car rental employees can access the application. They can view current reservations.
- Only those employees who work as booking agents can create or cancel reservations. Standard booking agents can create and cancel reservations for the standard vehicle types (economy, compact, intermediate, full size, and mini van).
- Only premium booking agents can create or cancel reservations for premium or luxury cars.

To perform the authorization check, you also have to require authentication. For this purpose, you will also UME authentication mechanisms.

Objectives

By the end of this tutorial, you will be able to:

- ✓ Use authentication in the JSP to protect access to your application.
- ✓ Use UME permissions to protect access to the application and to distinguish between users with different authorizations for the EJB methods.
- ✓ Specify consolidated permissions in UME actions.
- ✓ Perform the administrative steps for creating roles and assigning roles to users using the UME user administration management console.

Prerequisites

Systems, Installations, and Authorizations

- ☐ The SAP NetWeaver Developer Studio is installed on your computer. The minimum stack level required is stack level 11.
- ☐ You can access the J2EE Engine from the SAP NetWeaver Developer Studio for deployment.
- ☐ You can log on to the J2EE Engine with an administrator user.

Knowledge

- ☐ Java knowledge and basic knowledge of the J2EE programming model is advantageous.
- ☐ You have acquired some basic experience with the J2EE toolset in the Developer Studio.

Next Step:

Before beginning with the tutorial, you should familiarize yourself with the concepts necessary for using UME permissions with this tutorial [Page 6].

Concepts Necessary for Using UME Permissions with this Tutorial

Before beginning with the tutorial, you should be familiar with the concept for using UME roles, actions, and permissions. For an overview of these concepts, see the following topics:

- Permissions, Actions, and UME Roles [Page 7]

  This topic explains how the UME enforces authorizations using permissions, actions, and UME roles.

- Permission class for your application [Page 8]

  This topic describes how to implement an explicit permission class for your application, which you will then use in your application code to check UME permissions.

- actions.xml file [Page 11]

  This topic describes the structure and syntax to use for the `actions.xml` file, which you use to consolidate the permissions that you check in your application into actions. The administrator consolidates these actions into roles, which he or she can then assign to the users.

- UME archive file [Page 13]

  This topic describes the UME archive file, which contains the `actions.xml` file, so that the actions can be deployed to the J2EE Engine.

Next Step:

If you are familiar with these concepts, you can begin the tutorial. See Importing the Project for the J2EE-Based Car Rental Application [Page 13].

Permissions, Actions, and UME Roles

Definition

Authorizations are enforced in User Management Engine (UME) using permissions, actions, and roles.

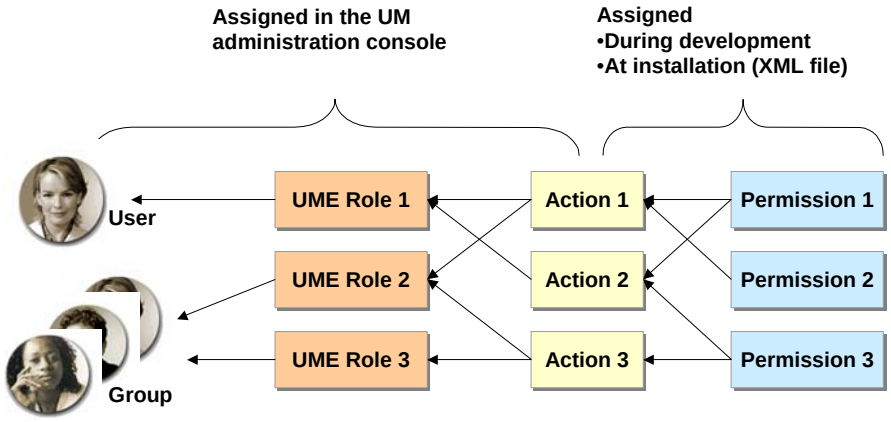
Internally in their Java code, applications define Java **permissions** and use them for access control.

An **action** is a collection of permissions. Every application defines its own set of actions and specifies the permissions assigned to the actions either in an XML file or (more seldom) dynamically in the code. The actions are listed in the user management administration console, where you can group them together into **roles**.

UME Roles group together actions from one or more applications. You assign roles to users in the user management administration console. By assigning roles to users, you define the users' authorizations.

Structure

The following figure illustrates the relationship between permissions, actions, and roles.



The advantage of having both actions and permissions is:

- Application developers can define finely grained permissions, but can hide the complexity by defining only a few actions.
- As the actions are normally defined in an XML file, they can be changed according to your requirements when you install the service.
- Administrators can assign actions to roles in the administration console. Permissions are not visible in the administration console.

Example

The user management administration console is an application running on User Management Engine. The application defines permissions in the code for activities such as changing a user's profile or modifying roles. In the XML file an action *Manage_Roles* is defined, that groups together all permissions that a user requires to administrate roles. This action includes permissions for viewing, modifying, and deleting roles.

For example, you could create a role called *Role Administrator* and assign the action *Manage_Roles* to it. Then you could assign any administrator that requires permissions to administrate roles to the *Role Administrator* role.

Interfaces

The corresponding UME interfaces are included in the packages:

- `com.sap.security.api`
- `com.sap.security.api.acl`
- `com.sap.security.api.logon`
- `com.sap.security.api.ticket`

Permission Class for Your Application

Definition

Permission class for checking permissions in your application. This class extends the abstract `java.security.Permission` class.

Use

Implement this permission class to check for authorizations in your application. You have two options for implementing this class:

- You can implement a permission class that extends one of the predefined classes provided by the UME.
- You can implement a permission class that extends the `java.security.BasicPermission` class. In this case, you also have to implement the corresponding constructors and methods.

Implementation Using Predefined Classes

We provide several predefined classes that you can use instead of implementing these constructors and methods directly. See the table below:

Predefined Permissions

Permission	Definition
`NamePermission`	This permission class maps a permission name to a single action, for example: `Name`="ViewReservation" `Name`="CreateReservation" `Name`="CancelReservation"
`ActionPermission`	This permission class maps a permission name to multiple actions, for example: `Name`="Reservation", `Action`="view" `Name`="Reservation", `Action`="create" `Name`="Reservation", `Action`="cancel"
`ValuePermission`	This permission class maps a permission name to a value, for example: `Name`="Value", `Action`="25" `Name`="Min", `Action`=">50" `Name`="Max", `Action`="<100"

Implementation Using BasicPermission

If you do not use one of these predefined permission classes, implement a class that extends the `java.security.BasicPermission` class. In this case, you have to implement the constructors and the abstract methods accordingly.

- Constructors

  In your class, implement both constructors from the `BasicPermission` class. These are `Name (string name)` and `Name (string name, string action)`.

- Methods

  The most important method that applies to your application's permission class is `implies (permission perm)`.

See the documentation for `java.security.BasicPermission` and `java.security.Permission` class for a complete list of the available methods.

Integration

Checking the Permissions

To check the permissions in your code, you can use either of the following methods:

- `checkPermission()`

  Checks whether the user is allowed to do the specific operation. If not, this is logged and an exception is thrown. Use this method to enforce that the user has the permission before performing a specific task.

- `hasPermission()`

  Similar to `checkPermission()`, but this method returns true or false instead of throwing an exception. Also, no logging occurs. Use this method to make decisions based on permissions, but without causing errors, for example, to hide areas on a page.

Example

See the following examples:

Example 1: Name Permission

```
package com.sap.engine.examples.permissions;

import com.sap.security.api.permissions.NamePermission;

public class TestPermission extends NamePermission {

    public TestPermission(String name)
    {
        super(name, null);
    }
}
```

Example 2: Action Permission

```
package com.sap.engine.examples.permissions;

import com.sap.security.api.permissions.ActionPermission;

public class TestPermission extends ActionPermission {

    public TestPermission(String name, String action)
    {
        super(name, action);
    }
}
```

Example 3: Checking an Action Permission using checkPermission

```
      try {
         user.checkPermission(new TestPermission("Reservation",
"create"));
         <code to execute if successful>;

      } catch (AccessControlException e) {
         <error handling>;
      }
```

actions.xml File

Definition

File that contains the collection of permissions in actions to use for your application.

Structure

Actions are specified in the `actions.xml` file using XML tag descriptors. See the table below for a list of the most frequently used tags and their parameters.

Action Tag Descriptors

Tag	Tag Description	Parameters	Parameter Description
`BUSINESSSERVICE`	Root element that describes the application in the rest of the tags.	`NAME`	Name of the application. It must be unique on the J2EE Engine.
`DESCRIPTION`	Provides a short description for the application or for actions.	`LOCALE`	Land and language for the short description.
		`NAME`	Short description for the application or action.
`ACTION`	Describes the individual actions.	`NAME`	Name of the action.
`PERMISSION`	Describes each permission.	`CLASS`	Specifies the permission class that is used for the permission.
		`NAME`	String used to use for the permission's first parameter, for example, `Economy`.
		`VALUE`	Used by `Action` and `Value` permissions. Contains the string value to use for the permission's second parameter, for example, `create`.

An action may contain multiple permission tags.

Example

The following example shows how to use the most frequently used tag descriptors.

```
<BUSINESSSERVICE NAME="MyApplication">
  <DESCRIPTION LOCALE="en" VALUE="My Application" />
 <!--  Detailed Business Service Actions  -->
 <ACTION NAME="ViewReservation">
  <DESCRIPTION LOCALE="en" VALUE="Permission for viewing reservations" />
  <PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
      NAME="Reservation" VALUE="view" />
  </ACTION>

  <ACTION NAME= "CreateReservation" >
   <DESCRIPTION LOCALE= "en" VALUE= "Permission for creating
reservations" />

<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="create" />

  </ACTION>

  <ACTION NAME= "CancelReservation" >
   <DESCRIPTION LOCALE= "en" VALUE= "Permission for canceling
reservations" />

<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="cancel" />

   </ACTION>

   <ACTION NAME= "AllPermissions" >
       <DESCRIPTION LOCALE= "en" VALUE= "Permission for all maintenance
tasks" />

<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="view" />


<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="create" />


<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="cancel" />
   </ACTION>

  </BUSINESSSERVICE>
```

You can also use wildcards for parameters that accept all values. See the example action below.

```
  <ACTION NAME= "AllPermissions" >
   <DESCRIPTION LOCALE= "en" VALUE= "Permission for creating
reservations" />

<PERMISSION CLASS="com.sap.engine.examples.permissions.PermissionClass"
       NAME="Reservation" VALUE="*" />

  </ACTION>
```

UME Archive File

Definition

Archive file that contains the `actions.xml` file for an application.

Structure

This file has the following characteristics:

- It is an archive file with the extension `.ump`
- It contains the `actions.xml` file.

Integration

You create and deploy this archive in the SAP NetWeaver Developer Studio as a Development Component. When deployed to the J2EE Engine, the actions specified in the `actions.xml` file can be accessed by the UME and can therefore be assigned to UME roles by an administrator.

Next Step:

You can now begin this tutorial. See Importing the Project for the J2EE-Based Car Rental Tutorial [Page 13].

Importing the Project for the J2EE-Based Car Rental Tutorial

Use

The project you will use for this tutorial is the J2EE-based quick car rental application that is provided with the SAP NetWeaver Developer Studio. Therefore, to begin working with this tutorial, you first have to import this project into the Developer Studio workspace.

Prerequisites

- ☐ You know the workspace directory for your SAP NetWeaver Developer Studio.
- ☐ You know the path to the examples provided with the Developer Studio. In a default installation this path is *C:\Program Files\SAP\JDT\eclipse\examples*.
- ☐ If you have previously worked on the J2EE quick car rental application using the Developer Studio, then this project is closed and its contents are removed from the Developer Studio workspace. In addition, the application does not exist as a deployed application on the J2EE Engine.
- ☐ The J2EE Engine and the Software Deployment Manager (SDM) are running.
- ☐ You can connect to the J2EE Engine from the Developer Studio for deployment. You also know the SDM password to use for the connection.

This tutorial assumes that you are working with the J2EE quick car rental project as it is provided with the J2EE Engine's example projects. Therefore, if you have previously worked on either the J2EE or the Web Dynpro car rental applications and deployed them to the J2EE Engine, we recommend you remove them not only from the Developer Studio's workspace, but also from the J2EE Engine and start with a completely new set of projects. Note the following:

- When deleting the projects from the Developer Studio, also delete the content from the workspace. Also delete the Dictionary project and the Helperclasses. (Switch to the Navigator to make sure that all projects have been removed.)
- To remove the application from the J2EE Engine, use the Deploy service in the Visual Administrator. Remove the application `sap.com/QuickCarRentalEar` and if applicable, `local/TutSD_CarRental`, and any of the corresponding components.
- You do not need to remove any entries from the database itself.

Procedure

Importing the project template into the SAP NetWeaver Developer Studio

1. Navigate to the location of the examples for the SAP NetWeaver Developer Studio.
2. Unpack the `J2EE_QuickCarRental.zip` archive into the workspace directory for your SAP NetWeaver Developer Studio.
3. Start the SAP NetWeaver Developer Studio.
4. Switch to the J2EE Development perspective and import the projects from the quick car rental application.
   a. To switch to the J2EE Development perspective, choose *Window* → *Open Perspective* → *J2EE Development*.
   b. Choose *File* → *Import.*
   c. Select `Multiple Existing Project into Workspace` and choose *Next*.
   d. In the *Select base folder* field, enter your workspace directory (or use the *Browse...* function).

      The projects for the car rental application appear. These are *Helperclasses*, *J2EE_QuickCarRentalEar*, *J2EE_QuickCarRentalEjb*, *J2EE_QuickCarRentalWeb* and *QuickCarRentalDictionary*.
5. Select all of these projects. Also select the *Open projects after import* indicator and choose *Finish*.

   The projects are opened in the SAP NetWeaver Developer Studio.

6. Deploy the dictionary archive to the J2EE Engine.
   a. Switch to the Dictionary perspective. (Choose *Window* → *Open Perspective* → *Dictionary*.)
   b. Choose the *Dictionary Explorer*.
   c. Select the *QuickCarRentalDictionary* project, open the context menu with the right mouse button, and choose *Deploy*.

      If this is the first time you deploy from the SAP NetWeaver Developer Studio, then you are prompted for the SDM password. Enter this password to continue with deployment.
7. Deploy the EAR archive to the J2EE Engine.
   a. Switch back to the J2EE Development perspective and the *J2EE Explorer*.
   b. Expand the *J2EE_QuickCarRentalEar* project.
   c. Select the *J2EE_QuickCarRentalEar.ear* node, open the context menu and choose *Deploy to J2EE Engine*.
8. To test the deployment, start a Web browser and enter the URL for the application.

   ```
   http://localhost:50000/QuickCarRental
   ```

If the application does not run, then make sure the J2EE Engine and the SDM are running and that the host and port are correct.

Initial Project Structure

Once you have imported the project template into the Developer Studio, you will see the following structures in the *J2EE Explorer*.

J2EE Project Structure
J2EE application project: **J2EE_QuickCarRentalEar**
J2EE Enterprise Java Bean project: **J2EE_QuickCarRentalEjb**
J2EE Web project: **J2EE_QuickCarRentalWeb**

Next Step:

You can now begin with the tutorial steps. See Using UME Permissions in the Application – Steps [Page 16].

Using UME Permissions in the Application – Steps

The steps you will perform to protect access to the quick car rental application using UME permissions are:

1. To use the UME functions, you first have to include the UME libraries in your development project.
2. You will protect access to the application by including security mechanisms in the JSP:
    a. To be able to perform authorization checks, you must know the user that is attempting to access the application. Therefore, you will require authentication in the JSP.
    b. You will create the permission class to use for the JSP.
    c. You will check the permission in the JSP using the `checkPermission()` method.
3. You will also use UME roles to protect access to the EJB methods:
    a. You will create a permission class to be used by the EJB.
    b. To check the permission, you need to obtain the user's ID, which was acquired by the JSP and is stored in the current context.
    c. You will check the permission in the EJB's methods using the `checkPermission()` method.
4. You will rebuild and redeploy the application on the J2EE Engine.
5. You will consolidate the permissions by specifying actions (`ViewReservations`, `MaintainStandard` and `MaintainPremium`) in the `actions.xml` file.
6. You will build and deploy the corresponding archive file.
7. You will perform the administration steps:
    a. You will create the users to use for this tutorial.
    b. You will create the corresponding UME roles.
    c. You will assign the roles to the users.
8. You will test the role assignments.

Next Step:

Including the UME Libraries [Page 17]

Including the UME Libraries

Use

The UME permission methods are included in the `com.sap.security.api.sda` library.

Add this library to each of the Web, EJB and EAR projects.

Prerequisites

- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.

Procedure

Adding the library to the Web project

1. Select the *J2EE_QuickCarRentalWeb* project, open the context menu with the right mouse button and choose *Add/Remove Additional Libraries*.

   The *J2EE libraries/interfaces/service* dialog appears.
2. Select `com.sap.security.api.sda` and choose *OK*.
3. Confirm the message that you also have to add the libraries to the EAR project with *OK*.

Adding the library to the EJB project

Repeat these steps for the *J2EE_QuickCarRentalEJB* project.

Adding the library to the EAR project

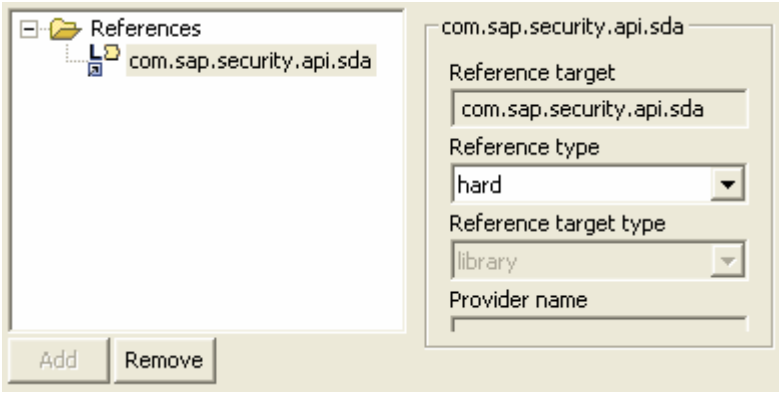
The procedure for adding the library to the EAR project is slightly different:

1. Expand the *J2EE_QuickCarRentalEAR* project.
2. Open the *application-j2ee-engine.xml* file by selecting it with a double-click.

   The properties for the file appear in the multi-page editor.
3. To add the library references, in the *General* tab page, select the *References* element and choose *Add*.
4. Choose *Select library/interface/service* from the dialog that appears.

5. Select `com.sap.security.api.sda` from the *J2EE libraries/interfaces/service* dialog and choose *OK*.

   The library is added as a reference to the EAR project as shown below.



6. Save the data.

Result

The UME permission methods can be resolved in the Web, EJB and EAR projects.

Next Step:

Protecting Access to the JSP Using Authentication and UME Permissions [Page 18].

Protecting Access to the JSP Using Authentication and UME Permissions

Use

In the first part of this tutorial, you will protect access to the application's Web client, which is a JSP with servlet.

Procedure

To enforce access protection to the JSP, you will implement the following steps:

1. Require authentication for access to the JSP application. In this way, you obtain the user's ID to use for the authorization check.
2. Create the `QuickReservationPermission` permission class to use for the JSP.
3. Implement the `checkPermission()` method in the servlet. This check makes sure that the user has the authorization for accessing the application.

   These permissions allow users to access the application. You will later differentiate between viewing and maintaining reservations when you protect access to the EJB methods.

Next Step:

Requiring Authentication [Page 19]

Requiring Authentication

Use

The first step in protecting access to the car rental application is to require user authentication. For this purpose, you will use the methods `getLoggedInUser()` and `forceLoggedInUser()`. These methods are available with the `UMFactory`.

`getLoggedInUser()` returns a currently logged in user. If no user is logged on, then use `forceLoggedInUser()` to request for authentication.

The two methods used in this tutorial for processing user input are the POST and GET methods. Therefore, you need to include the authentication checks in both the `doGet()` and `doPost()` methods in the JSP application.

Prerequisites

- ☐ The J2EE Perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.

Procedure

1. Expand *J2EE_QuickCarRentalWeb* → *source* → *com* → *sap* → *engine* → *examples* → *servlet* → *quickcarrental*.
2. Open the *QuickReservationServlet.java* file.

   The servlet code appears in the multi-tab editor.
3. Insert the imports for the UME classes.

```
import com.sap.security.api.IUser;
import com.sap.security.api.UMFactory;
```

4. In the `doGet()` section, insert the code for checking for a logged on user using the `getLoggedInUser()` method. If no user is logged on, then require authentication using the `forceLoggedInUser()` method. See the code sample below.

```
   protected void doGet(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {


      IUser user =
         UMFactory.getAuthenticator().getLoggedInUser(request,
response);
      if (null == user) {
         UMFactory.getAuthenticator().forceLoggedInUser(request,
            response);
         return;
      }


      doWork(request, response);
   }
```

5. Also insert the code in the `doPost()` section:

```
   protected void doPost(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {


      IUser user =
         UMFactory.getAuthenticator().getLoggedInUser(request,
response);
      if (null == user) {
         UMFactory.getAuthenticator().forceLoggedInUser(request,
            response);
         return;
      }


      doWork(request, response);
   }
```

6. Save the file.

Result

The application will now require user authentication.

Next Step:

Creating the Permission Class for the JSP [Page 21]

Creating the Permission Class for the JSP

Use

The next step in specifying which activities are to be protected with authorizations, you need to create a permission class to use for your application. (See Permission Class for Your Application [Page 8].)

For this part of the tutorial, you will implement an extension of the `NamePermission` class to protect access to the JSP application.

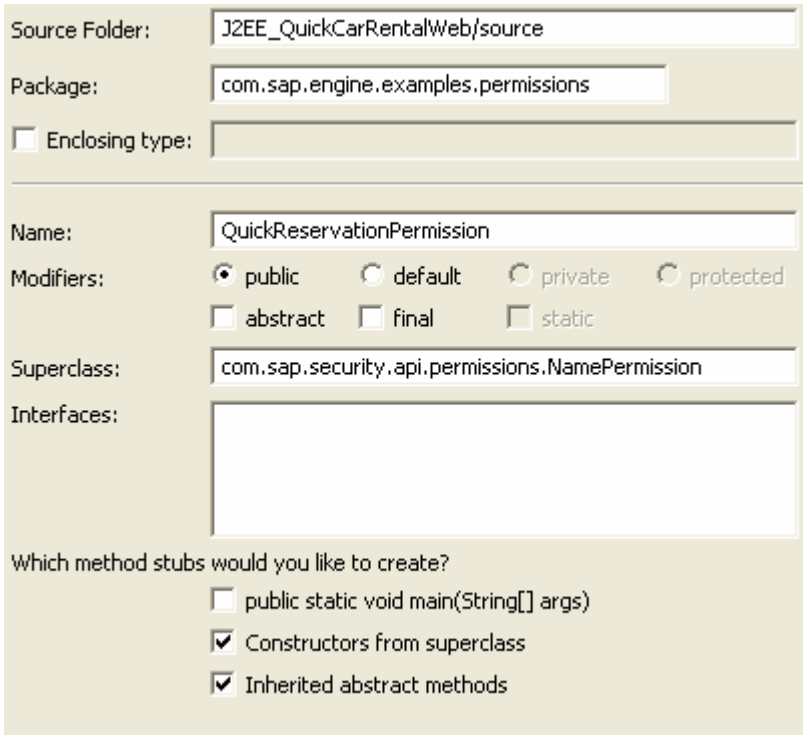
Prerequisites

- ☐ The J2EE Development perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's Web client project, *J2EE_QuickCarRentalWeb*, is displayed in the *J2EE Explorer*.

Procedure

1. Select the *J2EE_QuickCarRentalWeb* project, open the context menu and choose *New* → *Java Class…*

   The *New Java Class* dialog appears.
2. Enter **`com.sap.engine.examples.permissions`** in the *Package:* field.
3. Enter **`QuickReservationPermission`** in the *Name:* field.
4. Enter **`com.sap.security.api.permissions.NamePermission`** in the *Superclass* field (or use the *Browse...* function).
5. For the method stubs, select *Constructors from superclass* and *Inherited abstract methods*. See the figure below.



6. Choose *Finish* to continue.

   The Developer Studio creates the `QuickReservationPermission`, which extends the `NamePermission` class.

7. In the code for the permission class change the argument `arg0` to **`name`** and the argument `arg1` to **`action`** throughout the class.
8. Delete the `// TODO Auto-generated constructor stub` lines.
9. Save the file.

Result

The Java class `QuickReservationPermission` is created. See the code sample below.

```
package com.sap.engine.examples.permissions;

import com.sap.security.api.permissions.NamePermission;

public class QuickReservationPermission extends NamePermission {

   /**
    * @param name
    */

   public QuickReservationPermission(String name) {
      super(name);
   }


   /**
    * @param name
    * @param action
    */

   public QuickReservationPermission(String name, String action) {
      super(name, action);
   }
}
```

Next Step:

Checking the Permission in the Application [Page 23]

Checking the Permission in the Application

Use

In this step, you will use the `checkPermission()` method to make sure the user has the appropriate authorizations for the different activities.

Prerequisites

- ☐ The J2EE Development perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's Web client project, *J2EE_QuickCarRentalWeb*, is displayed in the *J2EE Explorer*.
- ☐ You have created the `QuickReservationPermission` Java class.

Procedure

1. Expand *J2EE_QuickCarRentalWeb* → *source* → *com* → *sap* → *engine* → *examples* → *servlet* → *quickcarrental*.
2. Open or return to the *QuickReservationServlet.java* file by selecting it with a double-click.
3. Add the import statements for the permission and exception classes.

```
import
com.sap.engine.examples.permissions.QuickReservationPermission;
import java.security.AccessControlException;
```

4. Insert the permission check in the code.

   The permission check will take place in the `viewAllBookings()` method. Add the incoming parameter `user` for this method and include the `checkPermission()` method in the `try` block. Check for the permission named `"access"`). Catch the `AccessControlException` if the user does not have this permission. See the sample code below.

```
   private void viewAllBookings(
      HttpServletRequest request,
      QuickOrderProcessorLocal order,
      IUser user)
      {
      HttpSession session = request.getSession(true);
      QuickBookingModel[] bookings;

      try {
         user.checkPermission(new
QuickReservationPermission("access"));

         try {
            bookings = order.viewActiveBookings();
            session.setAttribute(
            Constants.RESERVATIONS,
            formatBookings(bookings));
         } catch (QuickCarRentalException e) {

   session.setAttribute(Constants.CLIENT_MESSAGE,e.getMessage());
         }
      } catch (AccessControlException e) {
```

```
   session.setAttribute(Constants.CLIENT_MESSAGE,e.getMessage());
      }
   }
```

5. To check the permissions, the application must know the user's ID. Therefore, adjust the code sections in the `doGet()` and `doPost()` methods to pass the parameter `user` to the `doWork()` and `viewAllBookings()` methods as follows:

doGet()

```
   protected void doGet(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {

      IUser user =
UMFactory.getAuthenticator().getLoggedInUser(request, response);
      if (null == user) {
         UMFactory.getAuthenticator().forceLoggedInUser(request,
response);
         return;
      }
      doWork(request, response, user);
   }
```

doPost()

```
protected void doPost(
      HttpServletRequest request,
      HttpServletResponse response)
      throws ServletException, IOException {
         IUser user =
UMFactory.getAuthenticator().getLoggedInUser(request, response);
         if (null == user) {
            UMFactory.getAuthenticator().forceLoggedInUser(request,
response);
            return;
         }
         doWork(request, response, user);
   }
```

doWork()

```
   public void doWork(
      HttpServletRequest request,
      HttpServletResponse response,
      IUser user)
      throws ServletException {
      QuickOrderProcessorLocal order = initializeController();
      handleRequest(request, response, order);
      viewAllBookings(request, order, user);
      HttpSession session = request.getSession(true);
      RequestDispatcher dispatcher =
         request.getRequestDispatcher("/view");
      try {
         dispatcher.forward(request, response);
      } catch (IOException e) {
         e.printStackTrace();
         throw new ServletException(e.getMessage());
      }
}
```

6. Save the file.

Result

The application checks to see if the current user has the authorizations to view the car rental reservations.

Next Step:

Protecting Access to the EJB Methods Using UME Permissions [Page 25]

Protecting Access to the EJB Methods Using UME Permissions

Use

Although you can use UME permissions and check the relevant permissions within the frontend application, it is often better or perhaps necessary to differentiate between the various tasks in the backend. The methods processed with an application may be available to other applications, for example, as a Web service and you may not know what type of application is actually being used to access the business logic.

Therefore, when implementing authorization protection in your application, we recommend implementing the protection as close to the business logic as possible, in this case, in the EJB methods.

Procedure

To differentiate the tasks by using UME permissions, you will:

1. Create the permission class to use for the EJB permissions.
2. Obtain the user ID from the context.
3. Add the `checkPermission()` method and corresponding exceptions to each of the methods `cancelBooking()`, `saveBooking()`, and `viewActiveBookings()`.

Next Step:

Creating the Permission Class for the EJB Methods [Page 26]

Creating the Permission Class for the EJB Methods

Use

The next step in specifying which activities are to be protected with authorizations is to create a permission class to use for the EJB methods.

For your permission class, you will extend the pre-defined `ActionPermission` class. You will use this class to differentiate between the actions for viewing, creating, and canceling reservations for the different car types.

For more information about the types of permission classes available, see Permission Class for Your Application [Page 8].

Prerequisites

- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.

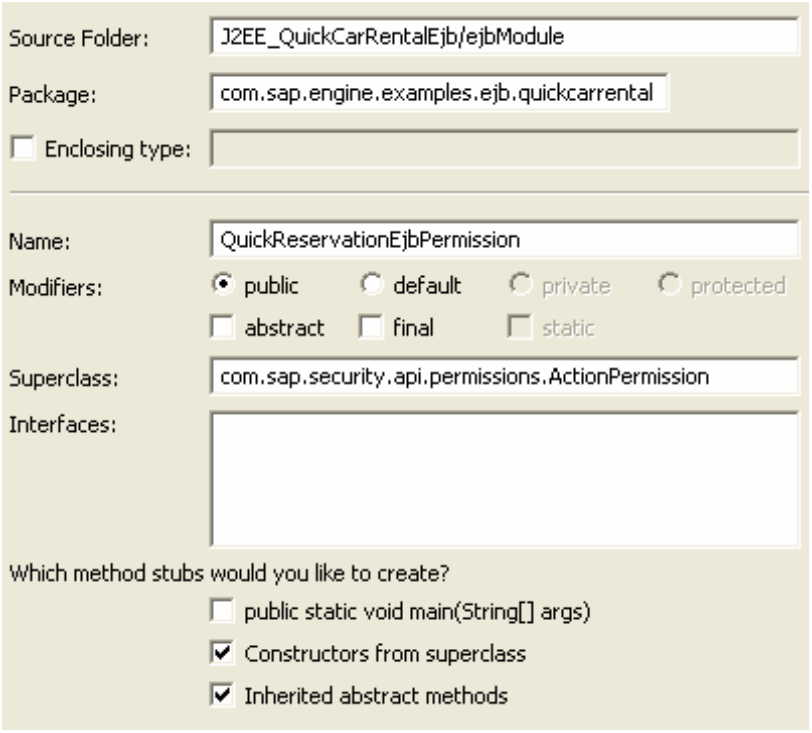
Procedure

1. Select the *J2EE_QuickCarRentalEjb* project, open the context menu with the right mouse button and choose *New* → *Java Class…*

   The *New Java Class* dialog appears.

2. Enter **`com.sap.engine.examples.ejb.quickcarrental`** in the *Package:* field.
3. Enter **`QuickReservationEjbPermission`** in the *Name:* field.
4. Enter **`com.sap.security.api.permissions.ActionPermission`** in the *Superclass* field (or use the *Browse...* function).

5. For the method stubs, select *Constructors from superclass* and *Inherited abstract methods*. See the figure below.



6. Choose *Finish* to continue.

   The Developer Studio creates the `QuickReservationEjbPermission`, which extends the `ActionPermission` class.

7. Change the arguments (`arg0`, `arg1`) for the `QuickReservationEjbPermission` to `cartype` and `action`.

8. Delete the `// TODO Auto-generated constructor stub` line.

9. Save the file.

Result

The Java class `QuickReservationPermission` is created. See the code sample below.

```
package com.sap.engine.examples.ejb.quickcarrental;
import com.sap.security.api.permissions.ActionPermission;
public class QuickReservationEjbPermission extends ActionPermission {

   /**
    * @param cartype
    * @param action
    */

   public QuickReservationEjbPermission(String cartype, String action) {
      super(cartype, action);
   }
}
```

Next Step:

Obtaining the User ID from the Context [Page 28]

Obtaining the User ID from the Context

Use

To be able to check for the appropriate permissions in the EJB methods, you must have access to the user ID to be checked. You can obtain the currently logged on user from the session context by using the `getCallerPrincipal()` and `getName()` methods. Afterwards, you need to convert this user ID, which is a string value, to the `IUser` type from the UME's UserFactory. The method to use for this purpose is `getUserByUniqueName()`.

Prerequisites

- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's EJB project, *J2EE_QuickCarRentalEjb*, is displayed in the *J2EE Explorer*.

Procedure

1. Expand *J2EE_QuickCarRentalEjb* → *ejb-jar.xml*.
2. Open the *QuickOrderProcessorBean* by selecting it with a double-click.

   The EJB's information appears in the multi-page editor.
3. Choose the *Bean* tab page.

   The source code for the EJB appears in the editor.
4. Add the following imports for the UME factory to the list of imports.

```
import com.sap.security.api.IUser;
import com.sap.security.api.UMException;
import com.sap.security.api.UMFactory;
```

5. Obtain the user's ID and convert it to the `IUser` type using the following code. Add this code to the `saveBooking()`, `cancelBooking()`, and `viewActiveBookings()` methods. See the examples below.

Method saveBooking()

```
public QuickBookingModel saveBooking(
   String vehicleTypeId,
   String dateFromString,
   String dateToString,
   String pickupLocation,
   String dropoffLocation)
   throws QuickCarRentalException {


   try {
      String username = myContext.getCallerPrincipal().getName();
      IUser user =
           UMFactory.getUserFactory().getUserByUniqueName(username);
   } catch (UMException e) {
      throw new QuickCarRentalException("Could not get user name.");
   }
      Date dateFrom = getDate(dateFromString);
      Date dateTo = getDate(dateToString);
...
```

Method cancelBooking()

```
public String cancelBooking(String bookingId)
      throws QuickCarRentalException {

      try {
         String username = myContext.getCallerPrincipal().getName();
         IUser user =
            UMFactory.getUserFactory().getUserByUniqueName(username);
         try {
            QuickBookingLocal booking =
               bookingHome.findByPrimaryKey(bookingId);
            booking.setStatus(Constants.STATUS_CANCELLED);
         } catch (FinderException e) {
            e.printStackTrace();
            throw new QuickCarRentalException(e.getMessage());
         }
      } catch (UMException e) {
         throw new QuickCarRentalException("Could not get user name.");
      }
      return bookingId + " cancelled.";
   }
```

Method viewActiveBookings()

```
   public QuickBookingModel[] viewActiveBookings()
      throws QuickCarRentalException {
      ArrayList bookings = new ArrayList();

      try {
         String username = myContext.getCallerPrincipal().getName();
         IUser user =
            UMFactory.getUserFactory().getUserByUniqueName(username);
      } catch (UMException e) {
         throw new QuickCarRentalException("Could not get user
name.");
      }

      try {
         Collection active =
            bookingHome.findByStatus(Constants.STATUS_ACTIVE);
         for (Iterator iterator = active.iterator();
iterator.hasNext();) {
            bookings.add(
               getBookingModel((QuickBookingLocal) iterator.next()));
         }

      } catch (FinderException e) {
         e.printStackTrace();
         throw new QuickCarRentalException(e.getMessage());
      }
      QuickBookingModel[] result = new
QuickBookingModel[bookings.size()];
      bookings.toArray(result);
      return result;
   }
```

6. Save the data.

Result

The EJB has access to the user ID that it is to use for checking for permissions.

Next Step:

Checking the Permission in the EJB Methods [Page 30]

Checking the Permission in the EJB Methods

Use

The next step is to make sure that the corresponding permissions are checked when the EJB methods are accessed. For this purpose, you will include the `checkPermission()` method and the corresponding exception in the `saveBooking()`, `cancelBooking()` and `viewActiveBookings()` methods in the Quick Order Processing Bean.

Prerequisites

- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's EJB project, *J2EE_QuickCarRentalEjb*, is displayed in the *J2EE Explorer*.

Procedure

1. If it is not already open, open the *QuickOrderProcessorBean* by selecting it with a double-click.
2. Choose the *Bean* tab page.
3. Add the import statements for the permission and exception classes.

```
import com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPe
rmission;
import java.security.AccessControlException;
```

4. Adjust the code for the `saveBooking()`, `cancelBooking()` and `viewActiveBookings()` methods to check the permissions as shown below.
   a. Start with the `saveBooking()` method. Add the `checkPermission()` statement and exception handling. Check for the car type specified by `vehicleTypeId` and the action `"create"`. Because you need the user ID for checking the permission, add these statements after the call for obtaining the user ID.

Method saveBooking()

```
   public QuickBookingModel saveBooking(
      String vehicleTypeId,
      String dateFromString,
      String dateToString,
      String pickupLocation,
      String dropoffLocation)
      throws QuickCarRentalException {


      try {
         String username = myContext.getCallerPrincipal().getName();
```

```
         IUser user =
UMFactory.getUserFactory().getUserByUniqueName(username);
         try {
            user.checkPermission(
               new QuickReservationEjbPermission(vehicleTypeId,
"create"));
         } catch (AccessControlException e) {
            e.printStackTrace();
            throw new QuickCarRentalException(
               user.getLastName()
                  + " may not create reservations for " +
vehicleTypeId + " car types." );
         }
      } catch (UMException e) {
         throw new QuickCarRentalException("Could not get user name.
" + e1);
      }

...
```

b. In the method `cancelBooking()`, add the `checkPermission()` statement and exception handling. In this method, you have to first obtain the vehicle type ID; the method to use is the `getVehicleId()` method. Then check for the car type specified by `vehicleTypeId` and the action `"cancel"`. Make sure you nest the **`try`** blocks so that the cancel function is performed correctly. See the example below.

Method cancelBooking()

```
   public String cancelBooking(String bookingId)
      throws QuickCarRentalException {

      try {
         String username = myContext.getCallerPrincipal().getName();
         IUser user =
            UMFactory.getUserFactory().getUserByUniqueName(username);

         String vehicleTypeId;

         try {
            QuickBookingLocal booking =
                  bookingHome.findByPrimaryKey(bookingId);
            vehicleTypeId = booking.getVehicleTypeId();
            try {
               user.checkPermission(
                  new QuickReservationEjbPermission(vehicleTypeId,
                        "cancel"));
               booking.setStatus(Constants.STATUS_CANCELLED);
            } catch (AccessControlException e) {
               e.printStackTrace();
               throw new QuickCarRentalException(
                  user.getLastName()
                     + " may not cancel reservations for " +
vehicleTypeId
                     + " car types." );
            }

//Comment or remove the following lines of code
//       try {
```

```
//          QuickBookingLocal booking =
//                bookingHome.findByPrimaryKey(bookingId);
//          booking.setStatus(Constants.STATUS_CANCELLED);

         } catch (FinderException e) {
            e.printStackTrace();
            throw new QuickCarRentalException(e.getMessage());
         }

      } catch (UMException e) {
         throw new QuickCarRentalException("Could not get user
name.");
      }

      return bookingId + " cancelled.";
   }
```

c. In `viewActiveBookings()`, add the `checkPermission()` statement and exception handling. Check for all car types and the action `"view"`.

Method viewActiveBookings()

```
   public QuickBookingModel[] viewActiveBookings()
      throws QuickCarRentalException {
      ArrayList bookings = new ArrayList();

      try {
         String username = myContext.getCallerPrincipal().getName();
         IUser user =
UMFactory.getUserFactory().getUserByUniqueName(username);

         try {
            user.checkPermission(
               new QuickReservationEjbPermission("*", "view"));
         } catch (AccessControlException e) {
                     e.printStackTrace();
                     throw new QuickCarRentalException(
                        user.getLastName()
                           + " may not view reservations." );
         }

      } catch (UMException e) {
         throw new QuickCarRentalException("Could not get user name.
" +
e);
      }

      try {
...
```

5. Save the file.

Result

The EJB will check the permissions for viewing, creating, and canceling reservations when a user attempts to perform these activities.

Next Step:

Rebuilding the Projects and Redeploying the Application [Page 33].

Rebuilding the Projects and Redeploying the Application

Use

The next step is to make your changes available on the J2EE Engine by rebuilding and redeploying the application.

Prerequisites

- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.
- ☐ You have completed the steps for requiring authentication, creating the permission classes and checking the permissions in the application.
- ☐ All modified files are saved.

   An asterisk (*) appears next to any file names for files that have not been saved.
- ☐ The J2EE Engine and SDM server are running and you can connect to the J2EE Engine from the Developer Studio.

Procedure

1. Choose *Project* → *Rebuild All*.

   This step rebuilds the EJB, Web, and if applicable, the Web Dynpro projects.
2. You still have to rebuild the application archive. Select the *J2EE_QuickCarRentalEar* project, open the context menu and choose *Build Application Archive*.
3. Expand the *J2EE_QuickCarRentalEar* project.
4. Select the *J2EE_QuickCarRentalEar.ear* file, open the context menu and choose *Deploy to J2EE engine*.

   The status of deployment appears in the *Deploy Output View* pane.

Result

Your application is deployed to the J2EE Engine. However, before you can test the application, you must specify the actions and perform the corresponding administration tasks.

Next Step:

You now have to create the actions that correspond to the permissions that are checked in the application.

Defining Actions in the actions.xml File [Page 34].

Defining Actions in the actions.xml File

Use

You have now included permission checks in your application to check if a user has the authorization to view reservations or to maintain a reservation for a specific vehicle type. To perform these activities, the user must have the appropriate permissions assigned in his or her user account.

To make the administration of these permission assignments easier, you will group individual permissions together in actions that the administrator will assign to the user's roles. To specify these groups of permissions in corresponding actions, create a file with the name `actions.xml`. You will create this file in a Development Component (DC) project in the SAP NetWeaver Developer Studio.

The `actions.xml` file that you create for this tutorial will contain the following actions:

- `AccessQuickCarRental`

  Users that are assigned to roles containing this action will be able to access the quick car rental JSP application. There is no differentiation between the car types and no maintenance permissions are provided. The corresponding permissions are checked in the JSP application using the `QuickReservationPermission` class.

- `ViewReservations`

  Users that are assigned to roles containing this action will be able to view reservations. The corresponding permissions are checked in the EJB methods using the `QuickReservationEjbPermission` class.

- `MaintainStandard`

  Users that are assigned to roles containing this action will be able to maintain standard vehicle types (economy, compact, intermediate, full size, and mini van). The permissions are also checked in the EJB methods using the `QuickReservationEjbPermission` class.

- `MaintainPremium`

  Users that are assigned to roles containing this action are also able to maintain the premium and luxury vehicle types. The corresponding permissions are also checked in the EJB methods using the `QuickReservationEjbPermission` class.

Prerequisites

- ☐ The J2EE Development perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.

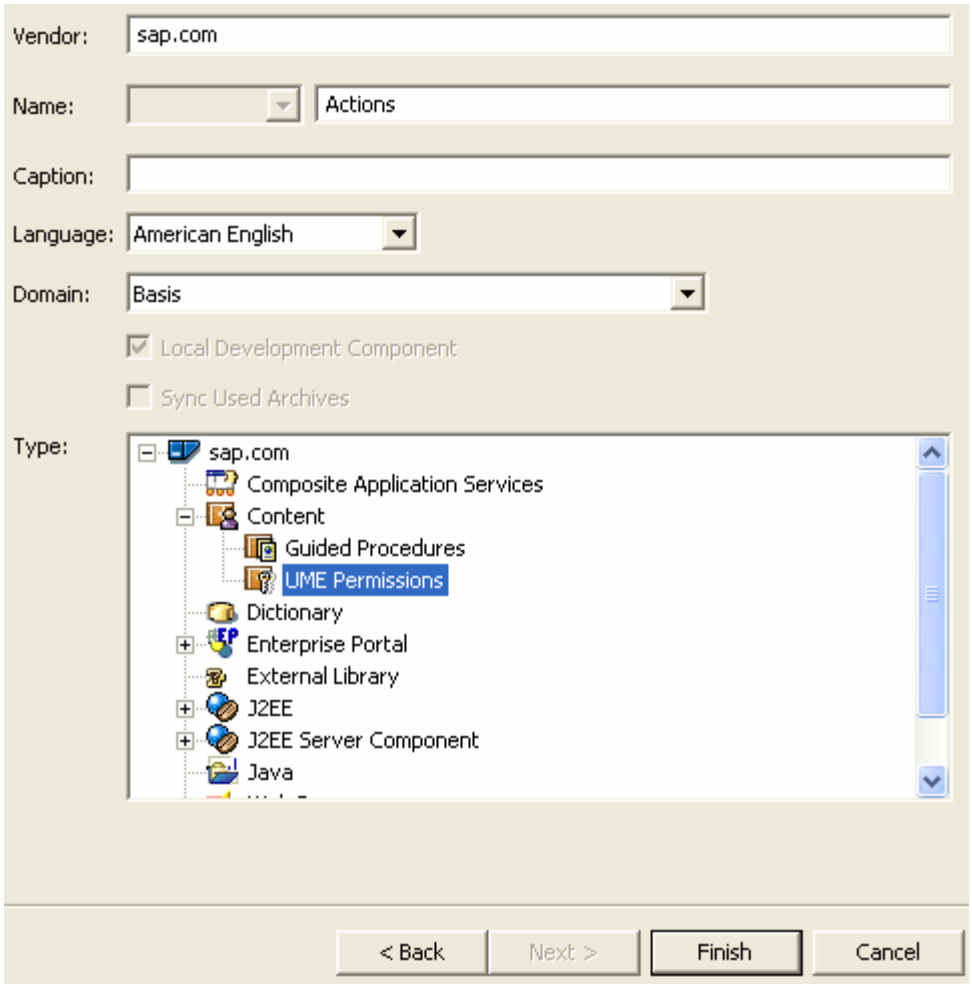
Procedure

1. Switch to the *Navigator*.
2. Choose *File* → *New* → *Project* from the menu.

   The *New Project* dialog appears.
3. Select `Development Component` and choose *Next*.

   The *New Development Component Project* dialog appears.

4. Expand *Local Development*, select *MyComponents* and choose *Next*.
5. Enter a name for the project, for example, **`Actions`**, in the *Name:* field.
6. Under *Type:*, expand the *Content* node, select *UME Permissions* and choose *Finish*. See the figure below.



The SAP NetWeaver Developer Studio creates a project with the name *LocalDevelopment~Actions~sap.com*.

7. Expand this project.
8. Select the *src* node, open the context menu and choose *New* → *File*.
9. Enter **`actions.xml`** in the *File name:* field and choose *Finish*.

   The XML file is created.
10. Choose the *Source* tab page from the multi-page editor.

11. Enter the following XML tags that specify the actions for `AccessQuickCarRental`, `ViewReservations`, `MaintainStandard`, and `MaintainPremium` in the file:

```
<BUSINESSSERVICE NAME="QuickCarRental">
  <DESCRIPTION LOCALE="en" VALUE="Car Rental actions for UME
      role tutorial" />
 <!--  Detailed Business Service Actions  -->
 <ACTION NAME="AccessQuickCarRental">
  <DESCRIPTION LOCALE="en" VALUE="Access Quick Car Rental" />
  <PERMISSION CLASS="com.sap.engine.examples.permissions.QuickReservati
onPermission" NAME="access" />
  </ACTION>

  <ACTION NAME= "ViewReservations" >
       <DESCRIPTION LOCALE= "en" VALUE= "View Reservations" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
   </ACTION>

   <ACTION NAME= "MaintainStandard" >
       <DESCRIPTION LOCALE= "en" VALUE= "Maintain standard reservations
" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Economy" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Economy" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Compact" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Compact" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Intermediate" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Intermediate" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Full Size" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Full Size" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Mini Van" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Mini Van" VALUE= "cancel" />
   </ACTION>
```

```
   <ACTION NAME= "MaintainPremium" >
       <DESCRIPTION LOCALE= "en" VALUE= "Create Reservations for Premiu
m
            and Luxury" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Premium" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Luxury" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Premium" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Qu
ickReservationEjbPermission"
         NAME= "Luxury" VALUE= "cancel" />
   </ACTION>

  </BUSINESSSERVICE>
```

12. Save the data.

Next Step:

Build and Deploy the Archive File [Page 37]

Build and Deploy the Archive File

Use

The next step in this tutorial is to build and deploy the archive file that contains the `actions.xml` file.

Prerequisites

- ☐ The *Navigator* is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *Navigator*.

Procedure

1. Select the *LocalDevelopment~<ProjectName>~sap.com* project, open the context menu and choose *Development Component* → *Build*.
2. In the dialog that follows, select your actions and choose *OK*.

   The SAP NetWeaver Developer Studio creates a software development archive (SDA) with the name *sap.com~<ProjectName>.sda* under *LocalDevelopment~<ProjectName>~sap.com* → *gen* → *default* → *deploy*. This archive contains the UME archive file, which contains the `actions.xml` file.

3. Select the *sap.com~<ProjectName>.sda* file, open the context menu and choose *Deploy*.

   The actions are deployed to the J2EE Engine.

Next Step:

In the next steps, you will perform the administrative tasks necessary so that you can test the permission checks in your application. See Creating the Users [Page 38].

Creating the Users

Use

The quick car rental application is now protected using authentication and UME permissions. To access the application and the various methods, users must be assigned to the appropriate roles. In the next steps, you will act as the administrator to set up the users and corresponding role assignments.

In the first of the administrative steps, you will create the following users:

- `Employee` (car rental employee)
- `Agent` (standard booking agent)
- `Pr_Agent` (premium booking agent)
- `OtherUser` (another user with no authorization to use the application)

You will later assign the UME roles to these users.

If these users already exist on the server, then you can omit this step.

Prerequisites

- ☐ The J2EE Engine is running.
- ☐ You have a user ID with administrator rights, for example, `Administrator`.

Procedure

1. Start the UME user administration management console.

   `http://localhost:50000/useradmin`

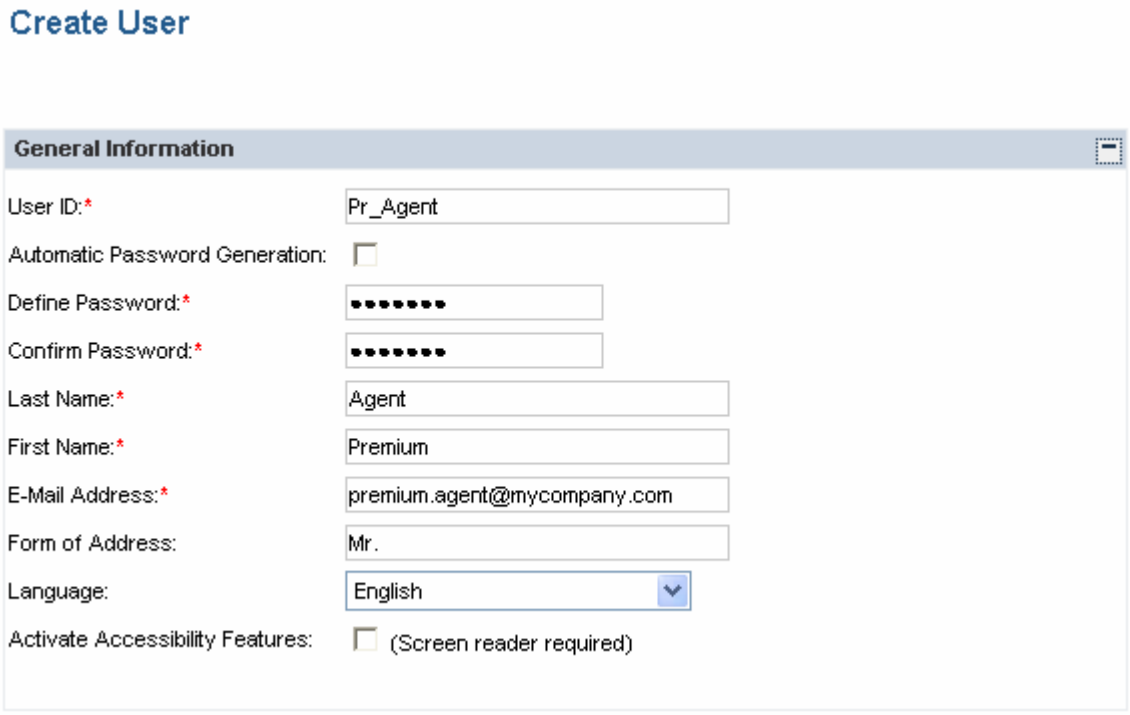
2. Log on as your administrator user.

   The *User Management* screen appears.

3. Under *Users*, choose *Create User*.

4. Enter the data for the user.

   See the example below for the user `Pr_Agent`:



5. Scroll down and choose *Create* at the bottom of the screen.
6. Repeat for the other users.

Next Step:

Creating UME Roles [Page 40].

Creating UME Roles

Use

Once the actions have been deployed on the J2EE Engine, the administrator can create the UME roles that contain the corresponding actions for the application.

For this tutorial, you will create the roles for accessing the application and assign them to the users as shown in the table below.

UME Roles and Their Corresponding Actions

UME Role	Description	Actions
Employee	Employee without any authorizations for booking reservations. Employees may view reservations however.	`QuickCarRental.AccessQuickCarRental` `QuickCarRental.ViewReservations`

BookingAgent	Standard agent that is allowed to create and cancel reservations. However, standard agents may not create or cancel reservations for premium or luxury vehicle types.	`QuickCarRental.AccessQuickCarRental`
		`QuickCarRental.ViewReservations`
		`QuickCarRental.MaintainStandard`

PremiumAgent	Agents that are allowed to maintain reservation for all vehicle types.	`QuickCarRental.AccessQuickCarRental`
		`QuickCarRental.ViewReservations`
		`QuickCarRental.MaintainStandard`
		`QuickCarRental.MaintainPremium`

If you have already created these roles, then you do not need to create new ones. Verify however, that these roles contain the corresponding actions.

Prerequisites

- ☐ You are logged on to the user management administration console as an administrator.

Procedure

1. In the user management console, choose *Roles*.

   The *Create/Maintain Roles* screen appears.

2. Create the role `Employee`:

   a. Choose the symbol for *Create New*  .

   The *Create a new role* screen appears.

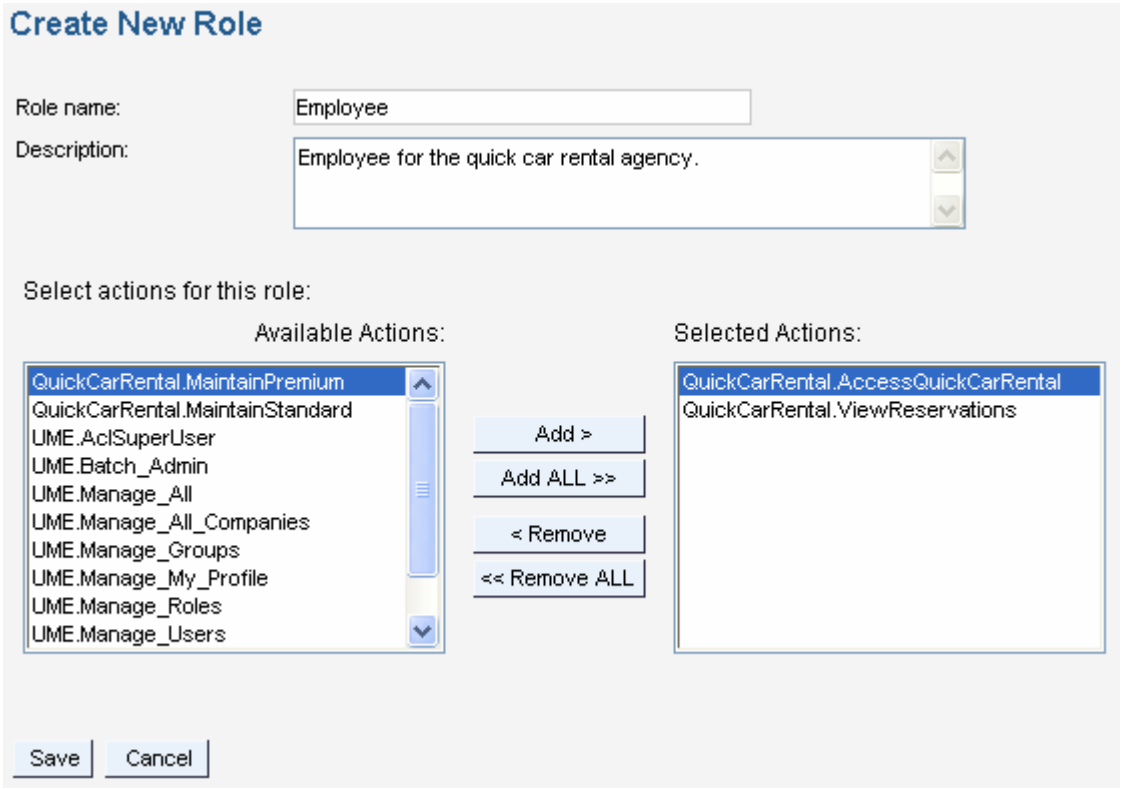
   b. Enter **`Employee`** as the role name and a short description.

   c. Select the following *Available Actions* and choose *Add*:

   - `QuickCarRental.AccessQuickCarRental`
   - `QuickCarRental.ViewReservations`

   The actions are added as *Selected Actions*.

   See the figure below.



   d. Save the data.

3. Create the role `BookingAgent`:
   a. Enter **`BookingAgent`** as the role name and a short description.
   b. Select the following *Available Actions* and choose *Add*:
      - `QuickCarRental.AccessQuickCarRental`
      - `QuickCarRental.ViewReservations`
      - `QuickCarRental.MaintainStandard`

      The actions are added as *Selected Actions*.
   c. Save the data.
4. Create the role `PremiumBookingAgent`.
   a. Enter **`PremiumBookingAgent`** as the role name and a short description.
   b. Select the following *Available Actions* and choose *Add*:
      - `QuickCarRental.AccessQuickCarRental`
      - `QuickCarRental.ViewReservations`
      - `QuickCarRental.MaintainStandard`
      - `QuickCarRental.MaintainPremium`

      The actions are added as *Selected Actions*.
   c. Save the data.

Result

The UME roles `Employee`, `BookingAgent` and `PremiumBookingAgent` are created. These roles contain the actions that you specified in the actions file for the application.

Next Step:

Assigning the Roles to the Users [Page 42]

Assigning Users to the Roles

Use

Once you have created the roles, assign the corresponding users to them.

If you have already created the user and role assignments, then you do not need to perform this procedure in detail. Verify that the user and role assignments are set up accordingly.

Prerequisites

- ☐ You are logged on to the user management administration console as an administrator.
- ☐ The users `Employee`, `Agent`, `Pr_Agent`, and `OtherUser` exist on the J2EE Engine.
- ☐ The roles `Employee`, `BookingAgent`, and `PremiumBookingAgent` exist on the J2EE Engine.

Procedure

1. Choose *Roles*.

   The *Create/Maintain Roles* screen appears.

2. If no roles are displayed in the *Roles* section, then use the *Search* function to display all of the roles.

3. Select the `Employee` role and choose the symbol for *Assign Users to…* ().

   The *Assign User(s)* screen appears.

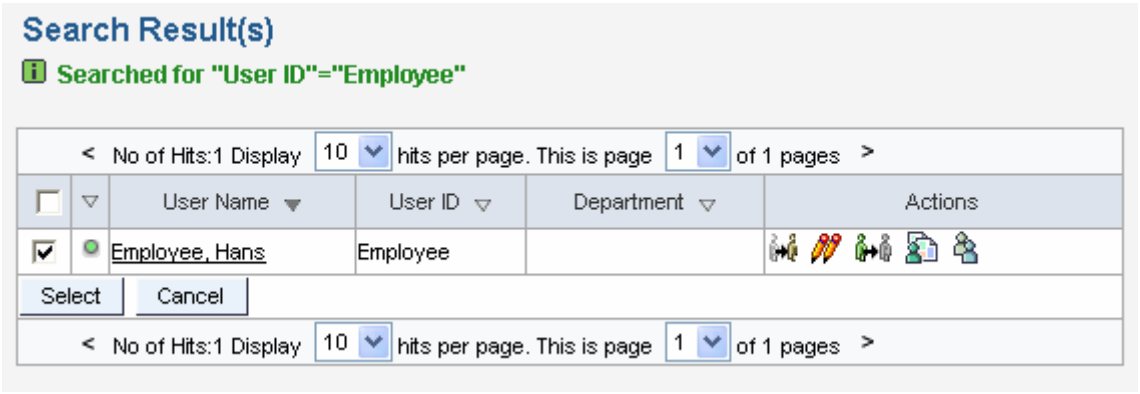
4. To assign a user to the role, choose the symbol for *Add Users …* ().

   The *Search for User* screen appears.

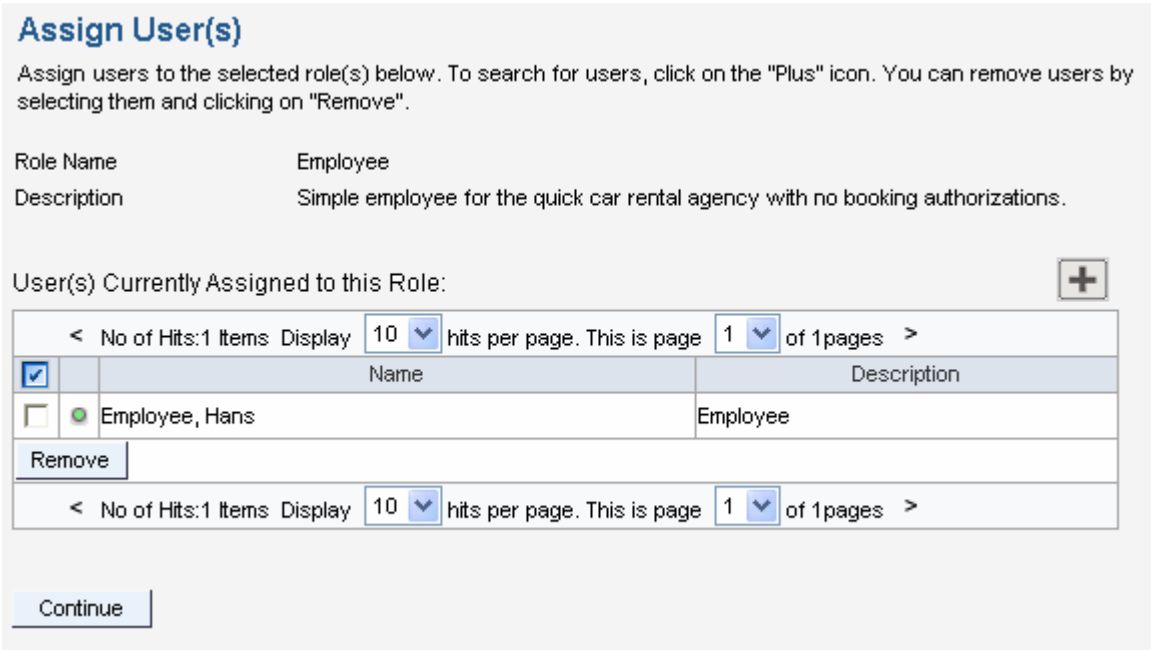
5. Enter **`Employee`** for the user ID and choose *Search*.

   The *Search Result(s)* screen appears.

6. Select the `Employee` entry and choose *Select*. See the figure below.



The user `Employee` is assigned to the `Employee` role. The *Assign User(s)* screen reappears, which now lists the user ID `Employee`. See the figure below.



7. Choose *Done*.

   The *Create/Maintain Roles* screen reappears.

8. Repeat steps 3-6 for the other roles. Assign the user `Agent` to the role `BookingAgent` and the user `Pr_Agent` to the role `PremiumBookingAgent`. Do not assign any of these roles to the user `OtherUser`.

Result

The users are assigned to the corresponding roles.

Next Step:

Testing the Access Protection [Page 45]

Testing the Access Protection

Use

You can now test the role assignments. For this test, you will log on to the application using each of the users. An error message should appear if the role assignment for the user does not allow him or her perform the corresponding action. See the table below.

Users and Corresponding Authorizations

User	Permitted Tasks	Non-Permitted Tasks
`Pr_Agent`	• View reservations • Create and cancel reservations for all vehicle types	Not applicable

`Agent`	• View reservations
	• Create and cancel reservations for the vehicle types:
	  ○ Economy
	  ○ Compact
	  ○ Intermediate
	  ○ Full Size
	  ○ Mini Van
		Not allowed to create or cancel reservations for the vehicle types:
		• Premium
		• Luxury

`Employee`	View reservations	Not allowed to create or cancel reservations
`OtherUser`	Not applicable	Not allowed to access the quick car rental application

Prerequisites

- ☐ The J2EE Engine is running.
- ☐ You have completed the tutorial steps and assigned the roles to the users.

Procedure

Testing the Role Assignment for User Pr_Agent

1. Start a new Web browser.
2. Access the quick car rental application.

   `http://localhost:50000/QuickCarRental`
3. Log on to the application as the user **`Pr_Agent`**. (If necessary, change the initial password.)
4. Create a reservation using the vehicle type Economy.

   This reservation is created.

5. Create a reservation using the vehicle type Luxury or Premium.

   This reservation is also created.

6. Create and cancel additional reservations.

   All attempts should be successful.

   When you are finished, make sure you have at least one reservation with a standard vehicle type and one with a vehicle type Premium or Luxury.

7. Close the Web browser.

Testing the Role Assignment for User Agent

Start a new Web browser and repeat these steps for the user `Agent`. `Agent` should be able to create and cancel reservations for all vehicle types except for Premium and Luxury.

Testing the Role Assignment for User Employee

Start a new Web browser and repeat these steps for the user `Employee`. `Employee` should be able to view the existing reservations, but should not be able to create or cancel any reservations.

Testing the Role Assignment for a User with no Authority to Access the Application

Start a new Web browser and repeat these steps for a user that is not assigned to any of the quick car rental roles, for example, `OtherUser`.

Result

You have protected access to the quick car rental application using UME permissions, actions and roles.

You are now finished with this tutorial.

If you want to see how to use J2EE security roles to protect access to the application, see Protecting Access to a J2EE-Based Application Using J2EE Security Roles [SAP Library].

If you want to see how to use UME permissions to protect access to the Web Dynpro car rental application, see Protecting Access to the Web Dynpro Application Using UME Permissions [SAP Library].