SAP NetWeaver '04 Security Tutorials

Protecting Access to the Web Dynpro 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 the Web Dynpro Application Using UME Permissions

Task

In this tutorial, you will include access protection to the Web Dynpro car rental application. In this application, the Web Dynpro serves as a Web service client for Web services implemented using an EJB. The Web service and EJB are provided with the quick car rental application, which is a standard sample application provided with the J2EE Engine.

For this tutorial, you will implement the UME permissions so that only authorized 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 check permissions in the EJB, you will also need to authenticate the user as part of the Web Dynpro authentication. The authentication information will be passed to the EJB over the Web service in the form of a logon ticket.

Also, as an optional step, you will learn how to adjust the Web Dynpro input screen depending on the user's authorizations. If the user does not have the proper authorizations, you will set the input fields to read-only so that the user cannot enter any data.

Objectives

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

- ✓ Use authentication to protect access to your Web Dynpro application.
- ✓ Activate authentication on the Web service so that the user's logon ticket is passed to the EJB.
- ✓ Use UME permissions to distinguish between users with different authorizations for the entity bean methods.
- ✓ Specify consolidated permissions in UME actions.
- ✓ Perform the administrative steps for creating roles and assigning roles to users using the UME user management administration console.
- ✓ Adjust the Web Dynpro input screen according to the user's authorizations.

Prerequisites

Systems, installed applications, 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 using the Visual Administrator.

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.
- ☐ You have acquired some basic experience with developing Web Dynpro applications.

Existing Applications

This application uses the following sample applications:

- ☐ The quick car rental application, which is provided as a sample application with the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental application, which serves as a client for the quick car rental application.

Both of these applications are provided in projects that you can import into the SAP NetWeaver Developer Studio.

The quick car rental application is part of the standard set of sample applications contained in the example directory of the Developer Studio (*.../SAP/JDT/eclipse/examples*). The Web service is already defined there and only needs to be deployed on the J2EE Engine. You can familiarize yourself with the quick car rental application using the tutorial Creating a J2EE-Based Car Rental Application [SAP Library].
To familiarize yourself with the Web Dynpro car rental application, see the tutorial Using Web Dynpro to Avail of the Car Rental Web Service [SAP Library]. Note however, that the initial project to use for this tutorial includes some enhancements not provided with the familiarization tutorial.

Next Step:

Before beginning with the tutorial, you should familiarize yourself with the concepts necessary for using UME permissions with this tutorial [Page 8].

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:

- Authentication Between the Components Using Logon Tickets [Page 8]

  This topic explains how the Web Dynpro authenticates the user, obtains a logon ticket for the user, which it passes to the EJB in the backend over the Web service.

- Permissions, Actions, and UME Roles [Page 10]

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

- Permission class for your application [Page 11]

  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 14]

  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 16]

  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 Templates for the Web Dynpro Application [Page 16].

Authentication Between the Components Using Logon Tickets

Purpose

To pass a user's authentication information (user ID) to further services or components, Web Dynpro uses the logon ticket mechanism. After a user has been authenticated on the server, he or she receives a logon ticket that is used for further access (for example, on the Web service and in the backend).

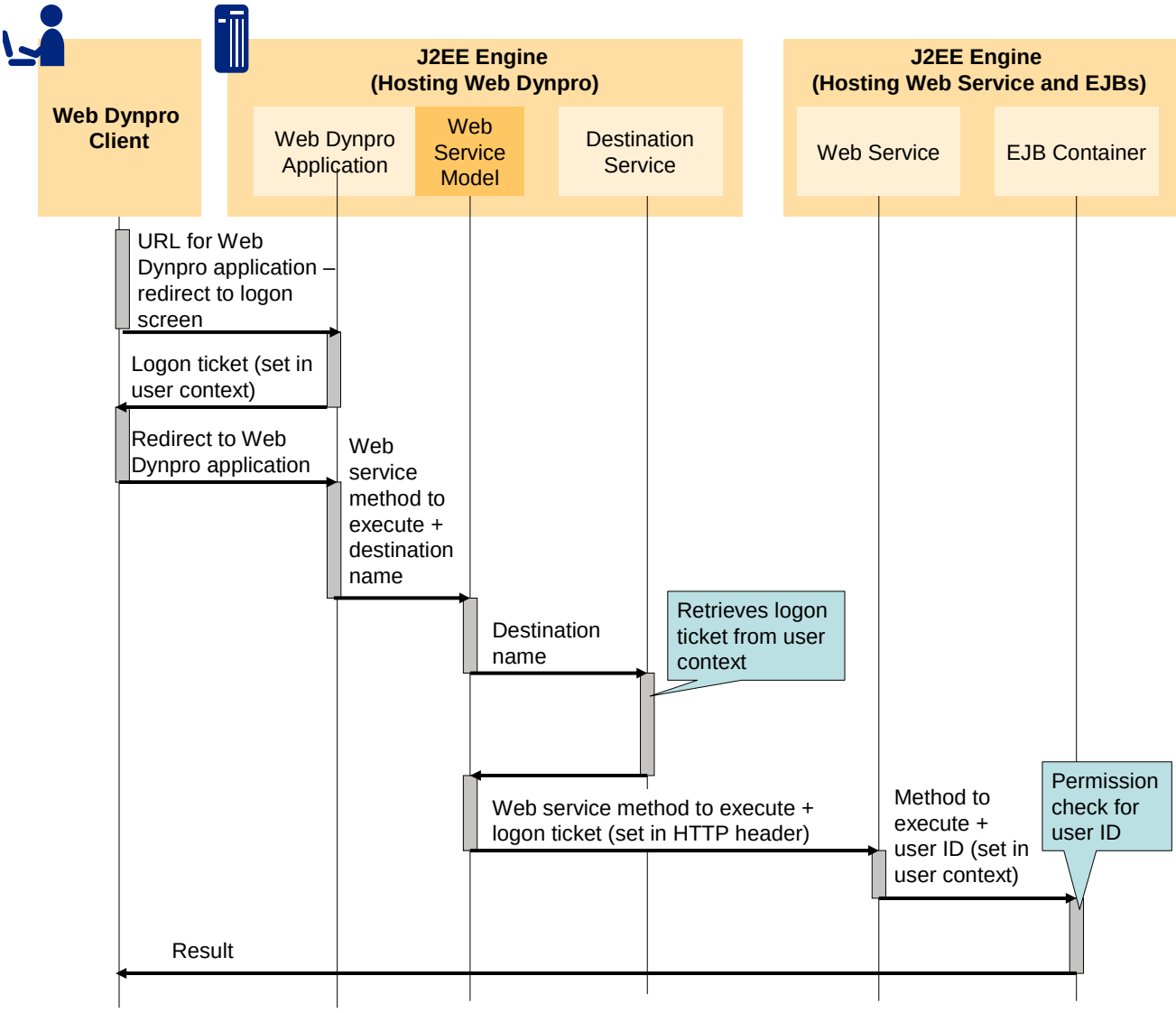
Prerequisites

- The Web Dynpro is set up to require authentication.
- The Web service requires authentication and accepts logon tickets.
- A destination for the application exists in the Destination service on the J2EE Engine. The destination is also configured to accept logon tickets.

Process Flow

Web Dynpro passes the user's logon ticket, which is stored in the user's context for the Web Dynpro, to the Web service. To obtain this logon ticket from the user context, the Web Dynpro's Web service model uses the Destination service. The Destination service retrieves the logon ticket and sets it in an HTTP header that is sent with the request to the Web service.

See the figure below:



The process flow is as follows:

1. The user calls the Web Dynpro application and is authenticated.
2. The J2EE Engine that hosts the Web Dynpro application issues the user a logon ticket, which is set in the user's context for the Web Dynpro application.
3. The user is redirected to the Web Dynpro application.
4. The Web Dynpro's Web service model retrieves the user's logon ticket from the user's context using the Destination service.

5. The Web service model sets the logon ticket in an HTTP header and calls the Web service method.
6. The Web service obtains the logon ticket from the HTTP header and sets the corresponding user ID in the user context in the backend so that it can be obtained by the EJB.
7. The EJB processes can now process the permission check using this user ID.

Result

The user is authenticated and the user ID exists in the backend context.

Permissions, Actions, and UME Roles

Definition

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

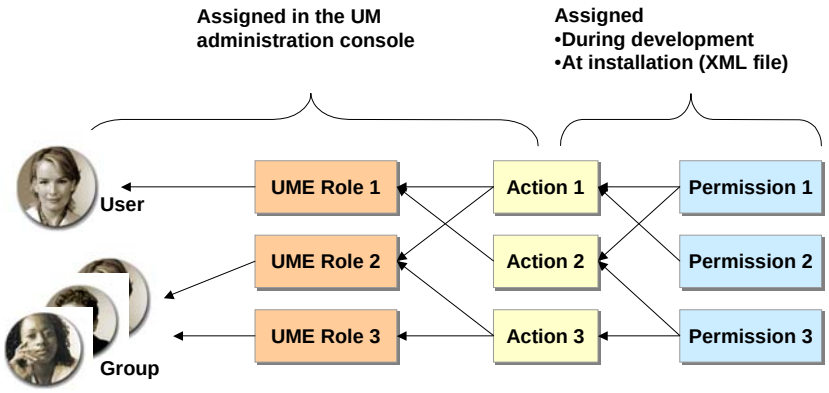
Internally in their Java code, applications define Java **permissions** and use them for access control.

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.

Continue with Importing the Project Templates for the Web Dynpro Tutorial [Page 16].

Importing the Project Templates for the Web Dynpro Tutorial

Use

This tutorial consists of a Web Dynpro project that calls the EJB methods provided with the J2EE quick car rental application over a Web service. Therefore, before you can begin with the tutorial, you need to import both the project for the car quick rental application as well as the Web Dynpro project. These projects are available in the following packages.

- The *J2EE_QuickCarRental* project is available with the SAP NetWeaver Developer Studio. It contains the EJB, the dictionary project, and the Web service used by the car rental application.
- The *TutWD_CarRental* project is the starting point for the Web Dynpro project used in this tutorial.

You can find both of these initial projects in the example directory for the SAP NetWeaver Developer Studio.

In addition, you can also find the completed project on the SDN. This project contains the quick car rental projects and the Web Dynpro project after completion of this tutorial. See *Home* → *Developer Areas* → *SAP NetWeaver Platform* → *Security*.

Prerequisites

- ☐ You know the workspace directory for your SAP NetWeaver Developer Studio.
- ☐ If you have previously worked on either the J2EE quick car rental project or the Web Dynpro car rental applications using the SAP NetWeaver Developer Studio, then these projects are closed and removed from the SAP NetWeaver Developer Studio workspace.
- ☐ 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*.

This tutorial assumes that you are working with the car rental applications as they are provided with initial 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 applications from the J2EE Engine, use the Deploy service in the Visual Administrator. Remove the applications `sap.com/QuickCarRentalEar` and `local/TutWD_CarRental` and any of their corresponding components.
- You do not need to remove any entries from the database itself.

Procedure

Importing the project templates 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* and *WebDynpro_CarRental.zip* archives into the workspace directory for the 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 Projects 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*, *QuickCarRentalDictionary* and *TutWD_CarRental*.
   e. 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.

Depending on your settings, you may receive warnings that apply to the Web Dynpro project. You can ignore these warnings.

5. 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.
6. 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*.
7. Deploy the Web Dynpro project to the J2EE Engine.
   a. Switch to the Web Dynpro perspective.
   b. Expand *TutWD_CarRental* → *Web Dynpro* → *Applications*.
   c. Select the *CarRentalApp* node, open the context menu and choose *Deploy New Archive and Run*.

      The Web Dynpro car rental application starts.

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. Also make sure that the host and port are correct in the model for the Web service (under *TutWD_CarRental* → *Web Dynpro* → *Models* → *Logical Ports* → *Config1Port_Document*).

Initial Project Structure

Once you have imported the project template into the Developer Studio, you will see the following structures in the corresponding views.

Web Dynpro Explorer

Web Dynpro Project Structure
Web Dynpro Project: **TutWD_CarRental**
Web Dynpro Application: **CarRentalApp**
Web Dynpro Model: **CarRentalModel**
Web Dynpro Component: **CarRentalComp**

J2EE Explorer

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

Navigator

The Navigator shows all of the projects, including the helper classes and the dictionary project needed by the quick car rental application.

Navigator
Helperclasses
J2EE application project: **J2EE_QuickCarRentalEar**
J2EE Enterprise Java Bean project: **J2EE_QuickCarRentalEjb**
J2EE Web project: **J2EE_QuickCarRentalWeb**
Dictionary project: **QuickCarRentalDictionary**
Web Dynpro Project: **TutWD_CarRental**

Next Step:

Integrating UME Permissions in the Web Dynpro Application - Steps [Page 20]

Integrating UME Permissions in the Web Dynpro Application - Steps

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

1. To use the UME functions in the EJB methods, you first have to include the UME libraries in the J2EE development project. You also have to include a reference to the WS Security service in the Web Dynpro project.
2. To acquire the user's ID so that you can later check the corresponding role assignment, you will require authentication for access to the application:
    a. You will require authentication for access to the Web Dynpro client.
    b. You will require authentication for access to the Web service and specify that the Web service accepts logon tickets.
    c. To pass the user's logon ticket to the Web service, the Web Dynpro uses the Destination service. Therefore, you will create a corresponding destination on the J2EE Engine.
    d. You will set the name of the destination to use in the Web Dynpro client.
3. You will then use UME permissions to protect access to the EJB methods:
    a. You will create a permission class for the EJB.
    b. To check the permission, you will obtain the user's ID from the current context.
    c. You will check the permission in the EJB's methods using the `checkPermission()` method.
4. To retrieve the error messages that occur if a user does not have the correct permission, you will adjust the error message handling in the Web Dynpro application.
5. You will rebuild and redeploy the applications on the J2EE Engine.
6. You will consolidate the permissions by specifying the actions to use in this tutorial in the `actions.xml` file.
7. You will build and deploy the corresponding archive file.
8. 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.
9. You will test the role assignments.

Optional Step

You can also check the UME permissions in the Web Dynpro client. To learn how to integrate the UME permissions in the Web Dynpro client, see the following steps. In these steps, you will use the `hasPermission()` method to check for authorizations. If the user does not have the authorization to create or maintain reservations, then you will set the corresponding fields to read-only.

1. You include the UME libraries in the Web Dynpro project.
2. You will create a permission class to use for the Web Dynpro client.
3. You will check the permission in the Web Dynpro client.
4. You will rebuild and redeploy the application.
5. You will add the corresponding action to the actions.xml file and deploy it to the J2EE Engine.
6. You will modify the UME roles.
7. You will retest the access protection.

Next Step:

Including the UME Libraries and Web Service References [Page 21]

Including the UME Libraries and Web Service References

Use

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

Add these libraries to both the EJB and EAR projects.

In addition, you have to add the reference to the WS Security service `tc/sec/wssec/service` to the Web Dynpro project.

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 libraries to the EJB project

1. Select the *J2EE_QuickCarRentalEjb* project, open the context menu with the right mouse button and choose *Add/Remove Additional Libraries*.

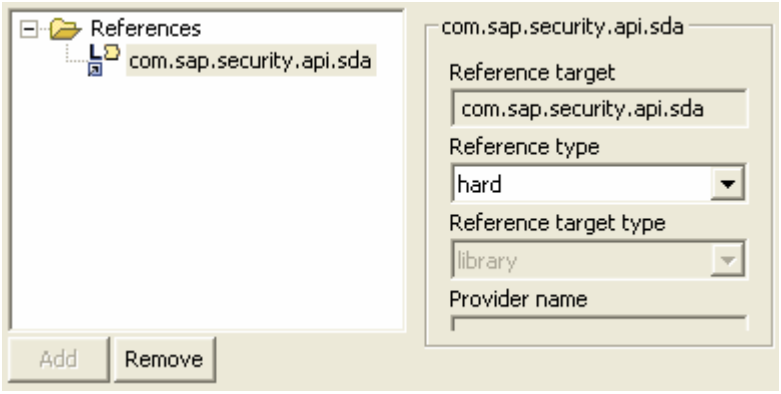
   The *J2EE libraries/interfaces/service* dialog appears.
2. Select `com.sap.security.api.sda` and choose *OK*.
3. Confirm the message that you also have to add the libraries to the EAR project with *OK*.

Adding the libraries to the EAR project

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 libraries are added as references to the EAR project as shown below.



6. Save the data.

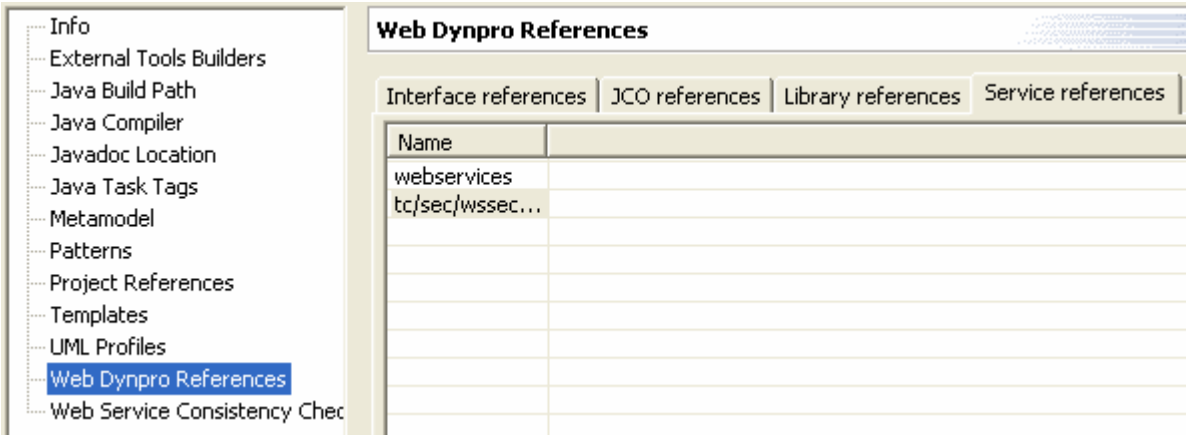
Adding the libraries to the Web Dynpro project

1. Switch to the Web Dynpro perspective and the *Web Dynpro Explorer*.
2. Select the *TutWD_CarRental* project, open the context menu and choose *Properties*.

   The *Properties for TutWD_CarRental* dialog appears.
3. In the left pane, select `Web Dynpro References`; in the right pane, choose the *Service References* tab page.
4. If the `tc/sec/wssec/service` is not included in the list of service references, insert it as follows:
   a. Choose *Add*.

      The *Create new reference* dialog appears.
   b. Enter **`tc/sec/wssec/service`** as the service name for the WS Security service and choose *OK*.

The reference is added to the list of service references. See the figure below.



5. Choose *OK* to continue.
6. Save the metadata.

Result

The security libraries that contain the UME permission methods are added to the EJB and EAR projects. A reference to the WS Security service is added to the Web Dynpro project.

Next Step:

Specifying Authentication for the Application [Page 23]

Specifying Authentication for the Application

Use

To be able to check permissions in the application, you first need to know the user ID for the user being checked. Therefore, you need to specify that authentication is required for the Web Dynpro application.

The Web Dynpro authenticates the user and passes the user's ID to the EJB methods in the backend over the Web service. For this purpose, it uses the logon ticket mechanism. Therefore, you also need to specify that the Web service requires authentication and that it accepts logon tickets.

To obtain the logon ticket and pass it to the Web service, Web Dynpro uses the Destination service. Therefore, in addition to specifying authentication on the Web Dynpro and on the Web service, you must create an HTTP destination on the J2EE Engine and specify its use within the Web Dynpro application.

Procedure

1. Specify authentication for access to the Web Dynpro client.
2. Specify authentication for the Web service.
3. Create an HTTP destination on the J2EE Engine.
4. Use this HTTP destination within the Web Dynpro client.

Next Step:

Specifying Authentication for Access to the Web Dynpro Client [Page 24]

Specifying Authentication for Access to the Web Dynpro Client

Use

The first step is to specify authentication for the Web Dynpro application. Afterwards, only users who are successfully authenticated on the J2EE Engine can access the Web Dynpro car rental application.

For this step, you will set the `Authentication` flag to `true` in the application's properties.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro project, *TutWD_CarRental*, is displayed in the *Web Dynpro Explorer*.

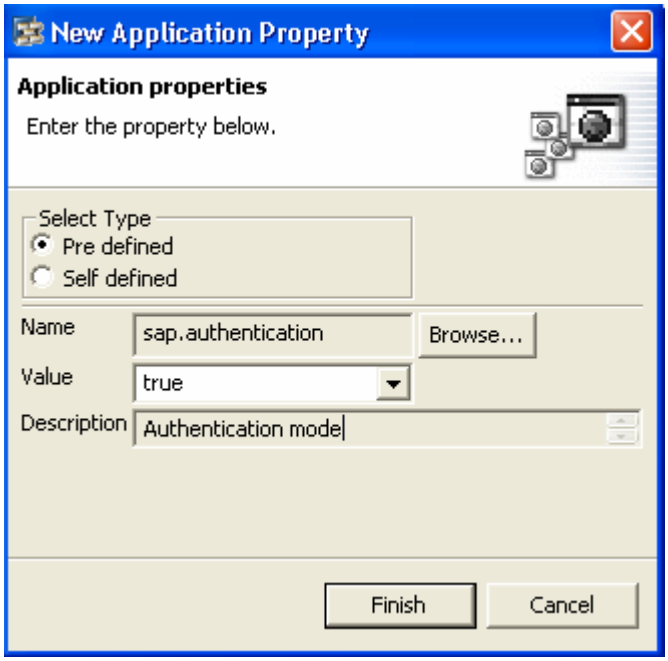
Procedure

1. Expand *TutWD_CarRental* → *Web Dynpro* → *Applications*.
2. Select the *CarRentalApp* with a double-click.

   The properties sheet for the car rental application appears.
3. Choose the *Application properties* tab page.
4. To add the authentication property, choose *New*.
5. In the dialog that follows, select `Pre defined` as the property type.
6. Choose *Browse…* . and select `Authentication` as the property in the dialog that follows. Choose *OK* to continue.

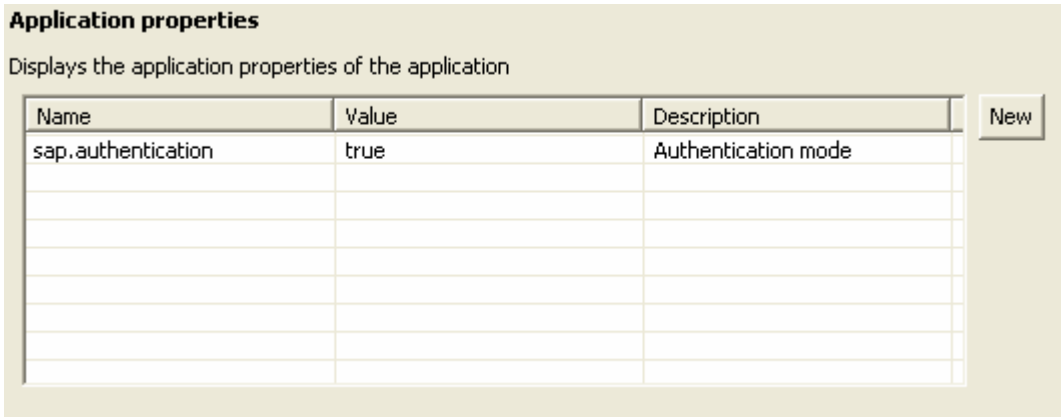
   The name `sap.authentication` appears in the *Name* field.

7. For the *Value*, select `true`.



8. Choose *Finish* to continue.

   The `sap.authentication` property appears in the property sheet. See the figure below.



9. Save the metadata.

Result

After you redeploy the application, users must authenticate themselves to be able to access the Web Dynpro application. However, you have to complete the steps for activating authentication and including the permission checks before you will rebuild and redeploy the application.

Next step:

Specifying Authentication for Access to the Web Service [Page 26]

Specifying Authentication for Access to the Web Service

Use

In the last step, you configured your application to use authentication, which for Web Dynpro, is the use of logon tickets. The next step is to activate authentication on the Web service.

Prerequisites

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

Procedure

1. If you are not already in working in the J2EE perspective, then switch to it.
2. Expand the *J2EE_QuickCarRentalEjb* project.
3. Select *Web Service Configurations* with a double-click.

   The *WS Deployment Descriptor* sheet appears.
4. Expand *Web Service Configurations* → *QuickCarRentalService* → *Config1*.
5. Select `Security`.

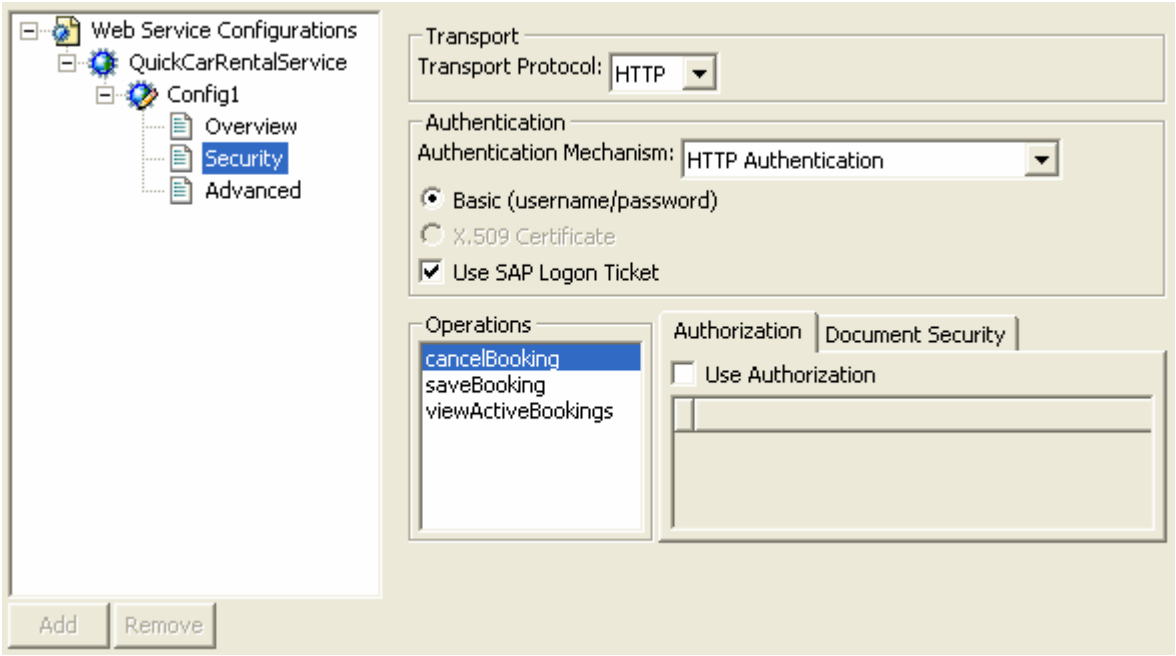
   The security options appear in the right pane.
6. Select the *Authentication Mechanism:* `HTTP Authentication`.
7. Select the *Use SAP Logon Ticket* option.

   The *Basic (username/password) option* is selected automatically.

   Because the permissions will be checked in the EJB methods in the backend, the Web service will not require any authorizations. Document security is also not needed for this tutorial.

See the figure below.



8. Save the data.

Result

The Web service will also require authentication.

Next Step:

Creating the HTTP Destination [Page 27]

Creating the HTTP Destination

Use

To transfer the user's logon ticket to the Web service, the Web Dynpro uses the Destination service on the J2EE Engine. Therefore, the next step is to create an HTTP destination to be used by the Web Dynpro application.

Prerequisites

- ☐ You have a user with administration rights on the J2EE Engine.
- ☐ The J2EE Engine is running and you can connect to it using the Visual Administrator.
- ☐ You know the URL for the Web service configuration.

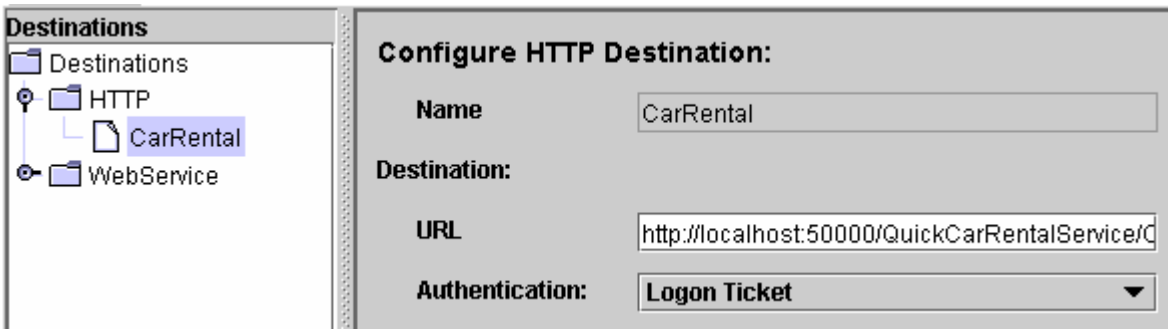
You can find the URL path in the Web Dynpro project under *TutWD_CarRental* → *Web Dynpro* → *Models* → *CarRentalModel* → *Logical Ports* → *Config1Port_Document* in the *Access* tab page. The parameter `~style=document` is necessary.

Procedure

1. Log on to the J2EE Engine as an administrator using the Visual Administrator.
2. Expand <*SID*> → *Server* → *Services*.
3. Select the `Destinations` service.
4. Under *Destinations*, select `HTTP` and choose *New*.

   Although there is a node for Web services, select `HTTP`. Web service destinations are used for deployable proxies and Web Dynpro uses standalone proxies. Therefore, the destination that will used by the Web Dynpro needs to be an HTTP destination and not a Web service destination.
5. In the dialog that follows, enter **`CarRental`** as the destination name and choose *OK*.
6. Enter the Web service's URL and select `Logon Ticket` as the authentication mechanism.

   See the figure below.



7. Save the destination.

Result

The Destination service retrieves the logon ticket from the Web Dynpro and passes it to the Web service available at the corresponding destination's URL.

The URL specified in the Destination service overrides any URL specified in the Web service configuration specified in the deployment description.

Next Step:

Using the HTTP Destination Within the Web Dynpro [Page 29]

Using the HTTP Destination Within the Web Dynpro

Use

Once you have created the HTTP destination on the J2EE Engine, you must specify that this destination is to be used by the Web Dynpro application. For this purpose, you will insert the `_setHTTPDestinationName()` method in the component controller where the methods for saving, canceling, and viewing reservations are executed.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro project, *TutWD_CarRental*, is displayed in the *Web Dynpro Explorer.*

Procedure

1. If you are not already working in the Web Dynpro perspective in the SAP NetWeaver Developer Studio, then switch to it.
2. Expand *TutWD_CarRental* → *Web Dynpro* → *Web Dynpro Components* → *CarRentalComp*.
3. Select the *Component Controller* with a double-click.

   The `CarRentalComp` controller appears.
4. Select the *Implementation* tab page.
5. Adjust the source code. Insert a call to the `_setHTTPDestinationName()` method before the `execute()` method is called. Insert this call in each of the methods `execute_ActiveBookings()`, `execute_Save()`, and `execute_Cancel()`.

   See the examples below.

Method execute_ActiveBookings()

```
public void execute_ActiveBookings( )
{
  //@@begin execute_ActiveBookings()
            IWDMessageManager msgMgr =
wdComponentAPI.getMessageManager();
            try {
//
      wdContext.currentActiveBookingsElement().modelObject().execute();

      Request_QuickCarRentalServiceViDocument_viewActiveBookings modObj
                                    =
wdContext.currentActiveBookingsElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();

                  wdContext.nodeResponse_ActiveBookings().invalidate();

            } catch (Exception ex) {
                  msgMgr.reportException(ex.getMessage(), false);
            }
  //@@end
}
```

Method execute_Save()

```
public void execute_Save( )
{
  //@@begin execute_Save()
      try {
//       wdContext.currentSaveBookingElement().modelObject().execute();
                  Request_QuickCarRentalServiceViDocument_saveBooking
modObj =

      wdContext.currentSaveBookingElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();

                  wdContext.nodeResponse().invalidate();

                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportSuccess("Success! ");

            } catch (Exception ex) {
                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportException(ex.getLocalizedMessage(),
true);
            }
  //@@end
}
```

Method execute_Cancel()

```
public void execute_Cancel( )
{
  //@@begin execute_Cancel()
            try {
//
wdContext.currentCancelBookingElement().modelObject().execute();

      Request_QuickCarRentalServiceViDocument_cancelBooking modObj =

      wdContext.currentCancelBookingElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();


      wdContext.nodeResponse_CancelBooking().invalidate();

                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportSuccess(
                        "Success! "
                              +
wdContext.currentResponse_CancelBookingElement().
                                    getResult());
```

```
            } catch (Exception ex) {
                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportException(ex.getLocalizedMessage(),
true);
            }
  //@@end
}
```

6. Save the metadata.

Result

The Web Dynpro application will use this destination to retrieve the logon ticket and pass it to the Web service.

Next Step:

Protecting Access to the EJB Methods Using UME Permissions [Page 31]

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 32]

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 11].

Prerequisites

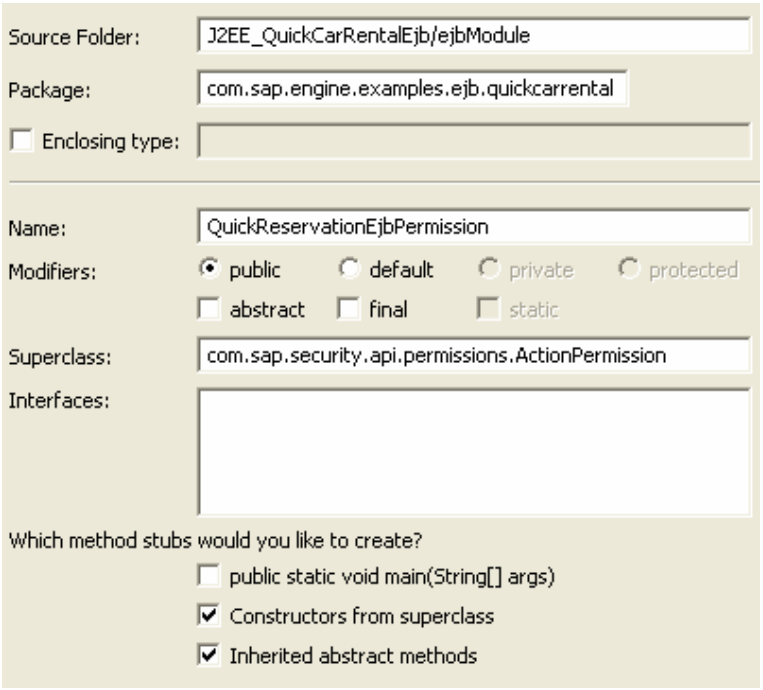
- ☐ The J2EE perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *J2EE Explorer*.

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 33]

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 36]

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:

Adjust the Message Handling [Page 39].

Adjusting the Message Handling

Use

As an intermediate step in this tutorial, you must adjust the message handling to display the access error messages produced by the EJB if the user does not have the necessary authorizations. These messages are produced by the EJB in the `QuickCarRentalException`.

To retrieve these error messages, use the `_getFaultString()` method for the quick car rental exception. Add the processing of this method to each of the methods that will produce access control error messages. These are `execute_ActiveBookings()`, `execute_Save()`, and `execute_Cancel()`.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro project, *TutWD_CarRental*, is displayed in the *Web Dynpro Explorer*.

Procedure

1. Switch to the Web Dynpro perspective.
2. Open or return to the *Component Controller* (under *TutWD_CarRental* → *Web Dynpro* → *Web Dynpro Components* → *CarRentalComp*).
3. Choose the *Implementation* tab page.
4. Add the `QuickCarRentalException` class to the list of imports.

```
import
com.sap.tut.wd.carrental.model.proxies.QuickCarRentalException;
```

5. Adjust the code in each of the methods to process the `QuickCarRentalException`. Adjust the code for each of the methods `execute_ActiveBookings()`, `execute_Save()`, and `execute_Cancel()`.To retrieve the error message use this class's `_getFaultString()` method.

Because the `CarRentalComp` controller handles the errors as the class `Exception`, but the actual error message is contained in the `QuickCarRentalException` class, you have to cast the exception accordingly before retrieving the error message.
See the examples below.

Method execute_ActiveBookings()

```
            try {

      //wdContext.currentActiveBookingsElement().modelObject().execute(
);
            Request_QuickCarRentalServiceViDocument_viewActiveBookings
modObj =

      wdContext.currentActiveBookingsElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();

                  wdContext.nodeResponse_ActiveBookings().invalidate();

            } catch (Exception ex) {
                  if (ex instanceof QuickCarRentalException) {
                        QuickCarRentalException qex =
(QuickCarRentalException) ex;
                        msgMgr.reportException(qex._getFaultString(),
true);
                  } else {
                        msgMgr.reportException(ex.getMessage(), false);
                  }
            }
            //@@end
      }
```

Method execute_Save()

```
      public void execute_Save() {
            //@@begin execute_Save()
            try {

      //wdContext.currentSaveBookingElement().modelObject().execute();
                  Request_QuickCarRentalServiceViDocument_saveBooking
modObj =

      wdContext.currentSaveBookingElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();
                  wdContext.nodeResponse().invalidate();

                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportSuccess("Success! ");

            } catch (Exception ex) {
                  if (ex instanceof QuickCarRentalException) {
                        QuickCarRentalException qex =
(QuickCarRentalException) ex;
                        MessageManager msgMgr =
                              (MessageManager)
wdComponentAPI.getMessageManager();
                        msgMgr.reportException(qex._getFaultString(),
true);
                  } else {
                        MessageManager msgMgr =
                              (MessageManager)
wdComponentAPI.getMessageManager();

      msgMgr.reportException(ex.getLocalizedMessage(), true);
                  }
            }
            //@@end
      }
```

Method execute_Cancel()

```
      public void execute_Cancel() {
            //@@begin execute_Cancel()
            try {

      //wdContext.currentCancelBookingElement().modelObject().execute()
;
                  Request_QuickCarRentalServiceViDocument_cancelBooking
modObj =

      wdContext.currentCancelBookingElement().modelObject();
                  modObj._setHTTPDestinationName("CarRental");
                  modObj.execute();
      wdContext.nodeResponse_CancelBooking().invalidate();

                  MessageManager msgMgr =
                        (MessageManager)
wdComponentAPI.getMessageManager();
                  msgMgr.reportSuccess(
                        "Success! "
                              + wdContext

      .currentResponse_CancelBookingElement()
                                    .getResult());

            } catch (Exception ex) {
                  if (ex instanceof QuickCarRentalException) {
                        QuickCarRentalException qex =
(QuickCarRentalException) ex;
                        MessageManager msgMgr =
                              (MessageManager)
wdComponentAPI.getMessageManager();
                        msgMgr.reportException(qex._getFaultString(),
true);
                  } else {
                        MessageManager msgMgr =
                              (MessageManager)
wdComponentAPI.getMessageManager();

      msgMgr.reportException(ex.getLocalizedMessage(), true);
                  }
            }
            //@@end
      }
```

6. After adjusting the code, save the metadata.

Result

The component controller will process the access control exceptions produced by the EJB.

Next Step:

Rebuilding the Projects and Redeploying the Application [Page 43]

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.
5. If you are working with the Web Dynpro tutorial, then deploy the Web Dynpro application:
   a. Change to the Web Dynpro perspective.
   b. Choose the *Web Dynpro Explorer*.
   c. Expand *TutWD_CarRental* → *Web Dynpro* → *Applications*.
   d. Select the *CarRentalApp*, open the context menu and choose *Deploy New Archive and Run*.

Result

Your application is deployed to the J2EE Engine.

If you are working with the Web Dynpro tutorial, then this application is also started. 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. See Defining Actions in the actions.xml File [Page 44].

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, you will create a file with the name `actions.xml` that you will later deploy to the J2EE Engine.

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

- `ViewReservations`

  Users that are assigned to roles containing this action will be able to view reservations.

- `MaintainStandard`

  Users assigned to roles containing this action will be able to maintain standard vehicle types (economy, compact, intermediate, full size, and mini van).

- `MaintainPremium`

  Users assigned to roles containing this action will be able to maintain the premium and luxury vehicle types.

The corresponding permissions are checked in the EJB methods using the `QuickReservationEjbPermission` class.

To create this file in the EJB project, you will use the *Navigator* in the SAP NetWeaver Developer Studio.

Prerequisites

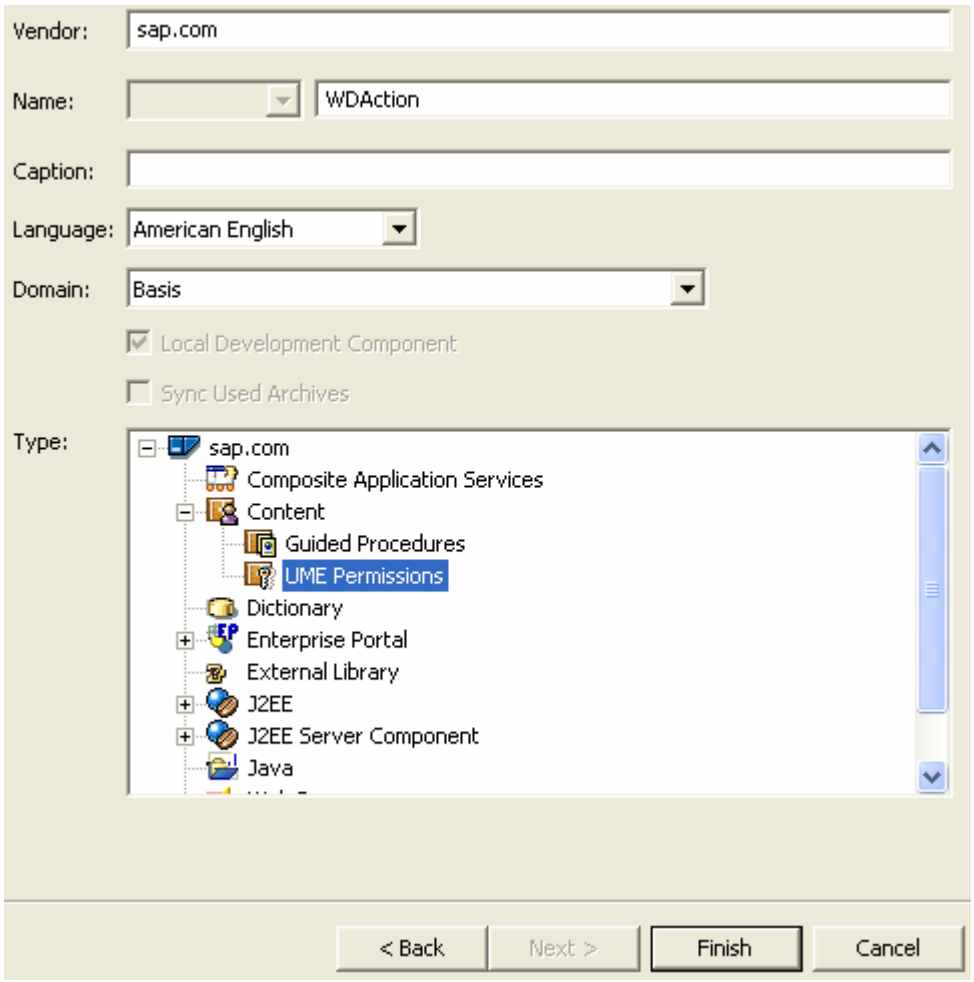
- ☐ The *Navigator* is displayed in the SAP NetWeaver Developer Studio.
- ☐ The quick car rental application's projects are displayed in the *Navigator*.

Procedure

1. 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, **`WDAction`**, in the *Name:* field. (The name is limited to 8 characters.)
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 `ViewReservations`, `MaintainStandard`, and `MaintainPremium` in the file.

If you already worked with the J2EE-based tutorial, then make sure that the business service name and the action names differ from those used in the J2EE-based tutorial so that you can distinguish them later in the administration.

```
<BUSINESSSERVICE NAME="WDQuickCarRental">
  <DESCRIPTION LOCALE="en" VALUE="Car Rental actions for UME
            role tutorial" />
 <!--  Detailed Business Service Actions  -->
  <ACTION NAME= "WDViewReservations" >
       <DESCRIPTION LOCALE= "en" VALUE= "View Reservations" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
   </ACTION>

   <ACTION NAME= "WDMaintainStandard" >
       <DESCRIPTION LOCALE= "en" VALUE= "Maintain standard reservation
s" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Economy" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Economy" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Compact" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Compact" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Intermediate" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Intermediate" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Full Size" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Full Size" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Mini Van" VALUE= "create" />
```

```
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Mini Van" VALUE= "cancel" />
   </ACTION>

   <ACTION NAME= "WDMaintainPremium" >
       <DESCRIPTION LOCALE= "en" VALUE= "Create Reservations for Premi
um
                        and Luxury" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "*" VALUE= "view" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Premium" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Luxury" VALUE= "create" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Premium" VALUE= "cancel" />
      <PERMISSION CLASS= "com.sap.engine.examples.ejb.quickcarrental.Q
uickReservationEjbPermission"
         NAME= "Luxury" VALUE= "cancel" />
   </ACTION>

  </BUSINESSSERVICE>
```

12. Save the data.

Next Step:

Build and Deploy the Archive File [Page 47]

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 48].

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)

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
   ```

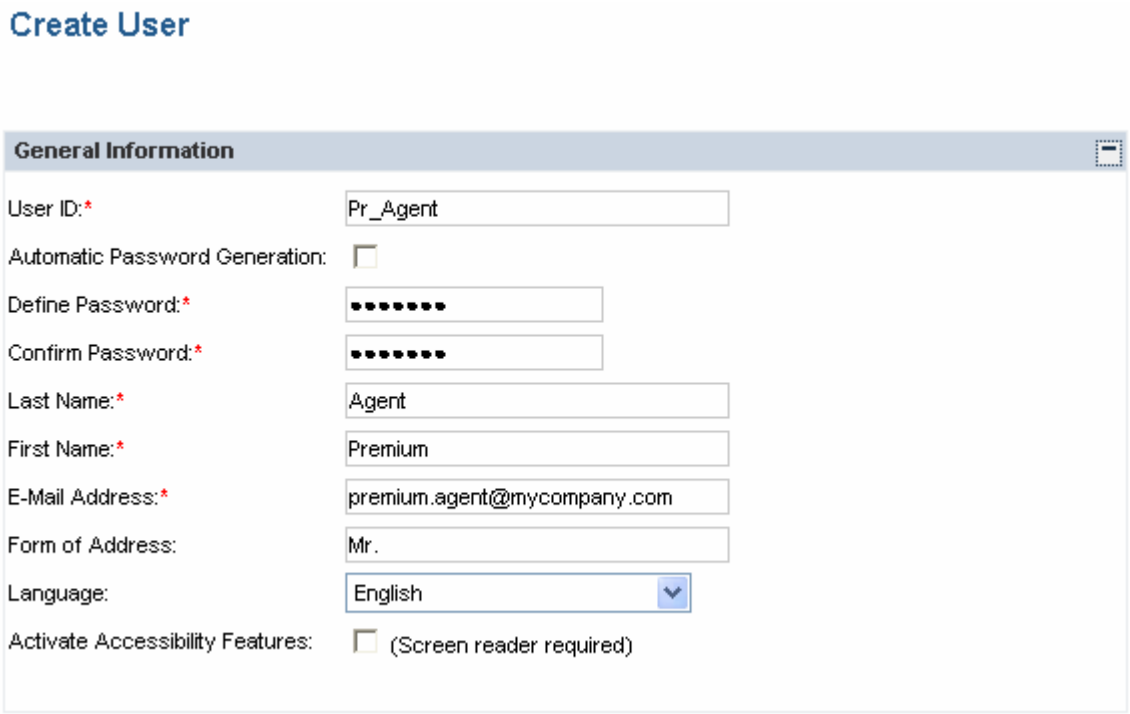
2. Log on as your administrator user.

   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 50]

Creating UME Roles

Use

The next step is to create the UME roles to use for the application. These roles will contain the actions you created for the application. See 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.	`WDQuickCarRental.WDViewReservations`
`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.	`WDQuickCarRental.WDViewReservations` `WDQuickCarRental.WDMaintainStandard`

`PremiumAgent`	Agents that are allowed to maintain reservation for all vehicle types.	`WDQuickCarRental.WDViewReservations`
		`WDQuickCarRental.WDMaintainStandard`
		`WDQuickCarRental.WDMaintainPremium`

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. Also, remove any other actions that may be included, for example, the action `QuickCarRental.AccessQuickCarRental`, which is used in the J2EE-Based tutorial with J2EE security roles.

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 or modify the `Employee` role:

   a. Choose the symbol for *Create New*  or Modify/View  .

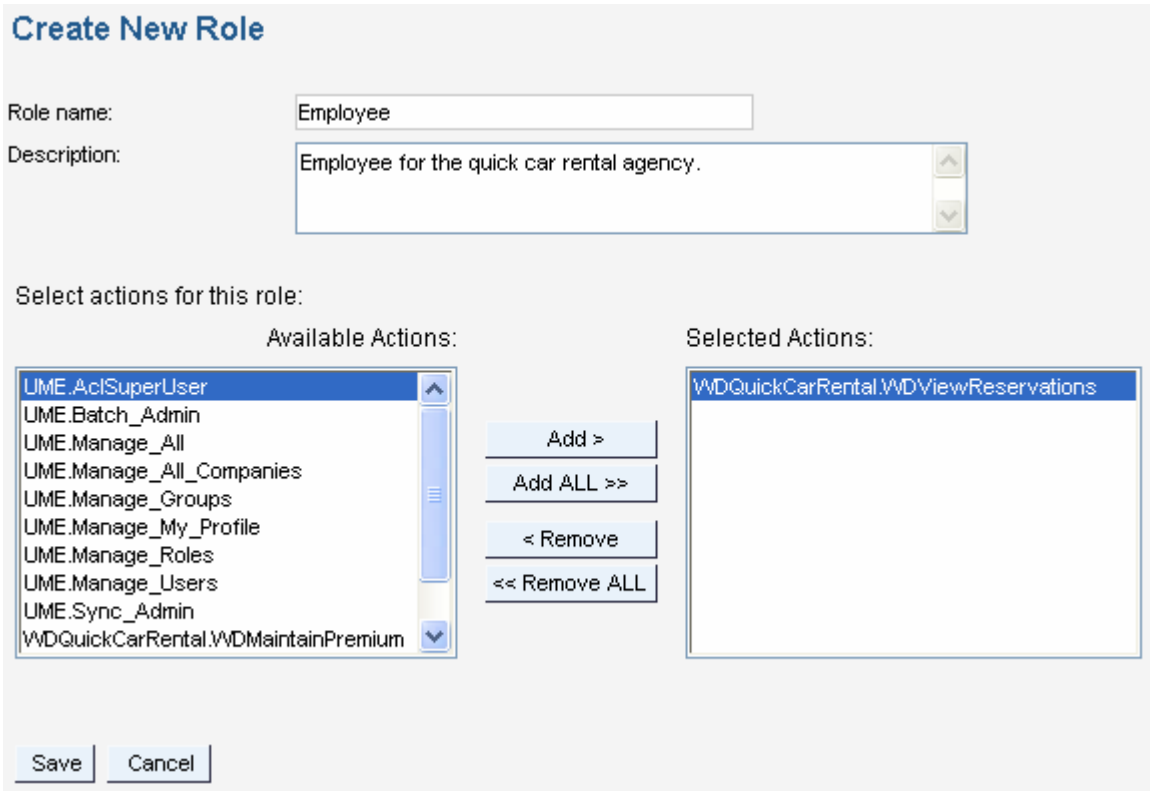
   The *Create a new role* or (*Modify Role*) screen appears.

   b. If you are creating the role, then enter **`Employee`** as the role name and a short description.

   c. Select the `WDQuickCarRental.WDViewQuickCarRental` action from the *Available Actions* and choose *Add*. Remove any other actions.

   This action is added as *Selected Actions*.

   See the figure below.



   d. Save the data.

3. Create (or modify) the role `BookingAgent`:

   a. Choose *Create New* (or *Modify/View*).

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

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

- `WDQuickCarRental.WDViewReservations`
- `WDQuickCarRental.WDMaintainStandard`

The actions are added as *Selected Actions*.

Remove any other existing actions.

d. Save the data.

4. Create the role `PremiumBookingAgent`.

a. Choose the symbol for *Create New* (or Modify/View).

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

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

- `WDQuickCarRental.WDViewReservations`
- `WDQuickCarRental.WDMaintainStandard`
- `WDQuickCarRental.WDMaintainPremium`

The actions are added as *Selected Actions*.

Remove any other existing actions.

d. 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 Users to the Roles [Page 52]

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 however, 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`, and `Pr_Agent` 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.

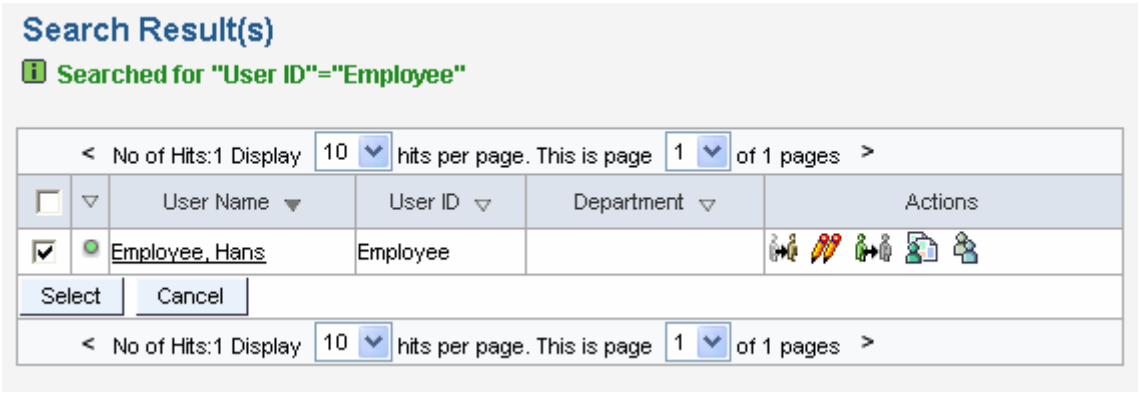
4. To assign a user to the role, choose the symbol for *Add Users …* ().

   The *Search for User* screen appears.

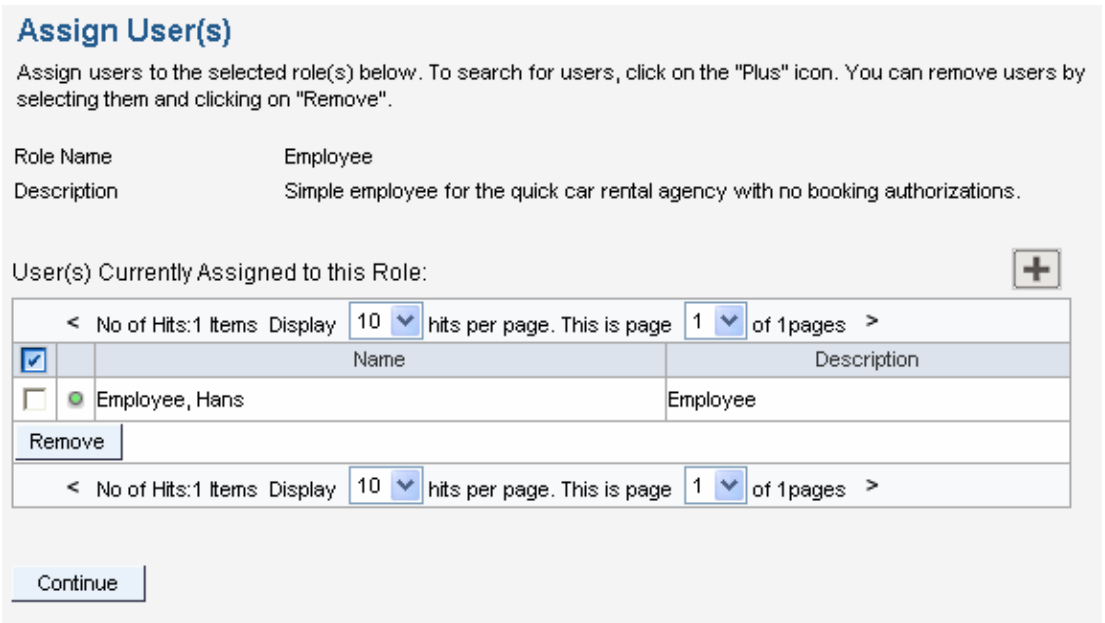
5. Enter **`Employee`** for the user ID and choose *Search*.

   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 *Continue*.

   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`.

Result

The users are assigned to the corresponding roles.

Next Step:

Testing the Access Protection [Page 68]

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

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. To make sure that all sessions are cleared, close any open Web browsers and start a new one.
2. Access the Web Dynpro car rental application.

   `http://localhost:50000/webdynpro/dispatcher/local/TutWD_CarRental/CarRentalApp/`

   You can also start the application from the SAP NetWeaver Developer Studio. In the Web Dynpro Explorer, expand *TutWD_CarRental* → *Web Dynpro* → *Applications*. Choose *Run* from the context menu for the *CarRentalApp* application.
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.

   Use the function *Display all Bookings* to refresh the list.
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 premium or luxury vehicle type.
7. Close the Web browser.

Testing the Role Assignment for User Agent

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.

1. Start a new Web browser.
2. Access the Web Dynpro car rental application again.
3. Log on to the application as the user **`Agent`**. (If necessary, change the initial password.)
4. Create a reservation using the vehicle type **`Economy`**.

   This reservation is created.
5. Attempt to create a reservation using the vehicle type **`Luxury`** or **`Premium`**.

   You receive an error.

6. Attempt to cancel a reservation that you created in the first test that has the luxury or premium vehicle type.

   You receive an error.

7. Close the Web browser.

Testing the Role Assignment for User Employee

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.

1. Start a new Web browser.
2. Access the Web Dynpro car rental application.
3. Log on to the application as the user **`Employee`**. (If necessary, change the initial password.)
4. Attempt to create or cancel car reservations.

   You receive errors.

5. Close the Web browser.

Result

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

Next Step:

If you want to continue with the optional step of adding a permission check in the Web Dynpro client, then see Checking Permissions in the Web Dynpro Frontend Client [Page 56].

Checking Permissions in the Web Dynpro Frontend Client

Use

You have now checked the permissions for using the Web Dynpro car rental application in the EJB methods in the backend. As the next step in this tutorial, you will also learn how to perform permission checks in the Web Dynpro frontend client.

We recommend performing the permission checks as close to the business logic as possible, in this case, in the EJB methods. However, it may also be desirable to perform checks in the frontend client, for example, to modify screen output based on a user's permissions.

For this section in the tutorial, we will show how to use the `hasPermission()` method in the Web Dynpro client. If the user does not have the authorization to create reservations, then you will change the properties for the input fields to read-only.

Prerequisites

- ☐ The project for the Web Dynpro car rental application is available in the SAP NetWeaver Developer Studio.

Procedure

1. Include the UME libraries in the Web Dynpro project.
2. Create the permission class to use for the Web Dynpro client.
3. Check the permission in the Web Dynpro client.
4. Define actions in an `actions.xml` file used by the Web Dynpro.
5. Rebuild the project.
6. Create the UME archive file used by the Web Dynpro.
7. Deploy the Web Dynpro application.
8. Modify the users' UME roles.
9. Test the access protection.

Next Step:

Including the UME JAR File in the Web Dynpro Project [Page 57]

Including the UME JAR File in the Web Dynpro Project

Use

To be able to include the `hasPermission()` check in the Web Dynpro client, you must first include the JAR file `com.sap.security.api.jar` in the build path in the Developer Studio. This archive contains the various UME interfaces, to include the IUser interface, which provides the `hasPermission()` method.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental project is displayed in the *Web Dynpro Explorer*.
- ☐ You know the location of the `com.sap.security.api.jar`.

  You can find this file on the J2EE Engine in the `\bin\ext\com.sap.security.api.sda` directory for the server.

  ```
  C:\usr\sap\<SID>\JC00\j2ee\cluster\server0\bin\ext\
  com.sap.security.api.sda\com.sap.security.api.sda.jar
  ```

Procedure

1. Select the *TutWD_CarRental* project, open the context menu and choose *Properties*.

   The *Properties for TutWD_CarRental* dialog appears.

2. In the left pane, select `Java Build Path`.

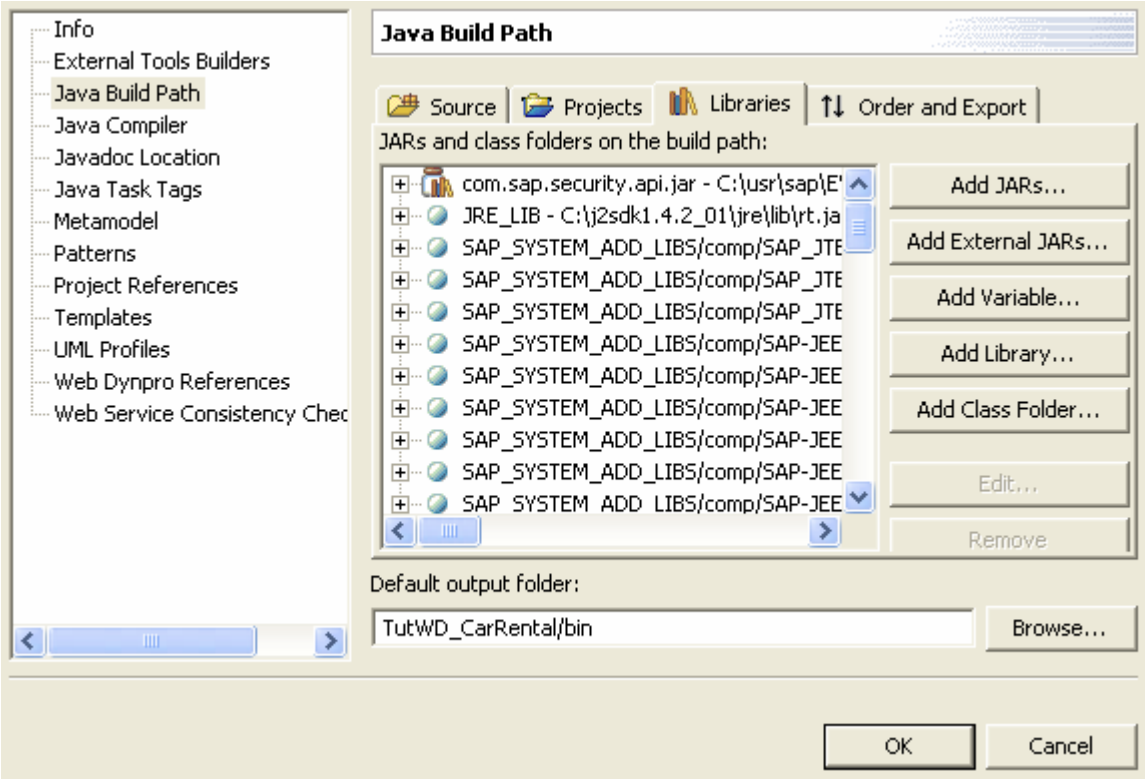
   In the right pane, the tab pages for the source files, projects, libraries and the order and export files appear.

3. Select the *Libraries* tab page.

4. Select *Add External JARs...*.

5. Browse to the location of the `com.sap.security.api.jar` file in the file system, select it, and choose *Open*.

   The JAR file is added to the list of libraries. See the graphic below.



6. Choose *OK* to continue.

Result

The Java build path for the Web Dynpro project is extended to include this JAR file. The UME IUser interface, which provides the `hasPermission()` method, can therefore be accessed by the Web Dynpro project.

Next Step:

Creating the Permission Class for the Web Dynpro [Page 59]

Creating the Permission Class for the Web Dynpro

Use

As with the EJBs, you have to create a permission class that is used to check the permissions in the Web Dynpro. In this case, you will create a permission class that extends the pre-defined `NamePermission` class. For more information about permission classes, see Permission Class for Your Application [Page 11].

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental project is displayed in the *Navigator*.

Procedure

1. Switch to the *Navigator*.
2. Select the *TutWD_CarRental* project, open the context menu and choose *New* → *Other…*
3. In the left pane of the dialog that follows, select `Java`; in the right pane, select `Class` and choose *Next*.

   The *New Java Class* dialog appears.
4. Enter the package and a name for the permission in the corresponding fields. Use **`com.sap.tut.wd.carrental`** as the *Package:* and **`WDCarRentalPermission`** for the *Name*.
5. Enter **`com.sap.security.api.permissions.NamePermission`** in the *Superclass:* field.
6. For the method stubs, select *Constructors from superclass* and *Inherited abstract methods*.
7. Choose *Finish* to continue.

   The Developer Studio creates the `WDCarRentalPermission`, which extends the `NamePermission` class.
8. Change the arguments for the `WDCarRentalPermission`. Change `arg0` to `name` and `arg1` to `action`.
9. Delete the `// TODO Auto-generated constructor stub` line.
10. Save the file.

Result

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

```
package com.sap.tut.wd.carrental;

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

public class WDCarRentalPermission extends NamePermission {

	/**
	 * @param name
	 */
	public WDCarRentalPermission(String name) {
		super(name);
	}

	/**
	 * @param name
	 * @param action
	 */
	public WebDynproCarRentalPermission(String name, String action) {
		super(name, action);
	}
}
```

Next Step:

Checking the Permission in the Web Dynpro Client [Page 60]

Checking the Permission in the Web Dynpro Client

Use

The next step is to adjust the Web Dynpro screen according to the user's authorizations. If the user does not have the authorization to maintain reservations, you will set the corresponding fields to read only and hide the buttons for creating or displaying reservations. For this purpose, you will call the `hasPermission()` method and adjust the screen accordingly.

Necessary Imports

The table below shows the packages and classes that are needed for processing the permission check and adjusting the Web Dynpro screen.

Packages and Classes

Package or Class	Description
`com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractInputField`	Interface to the input field properties
`com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractButton`	Interface to the button properties
`com.sap.tc.webdynpro.progmodel.api.WDVisibility`	Class specifying the visibility property of UI elements
`com.sap.tc.webdynpro.services.sal.um.api.IWDClientUser`	Interface to the Web Dynpro client user
`com.sap.tc.webdynpro.services.sal.um.api.WDClientUser`	Provides the methods for accessing the user properties
`com.sap.tc.webdynpro.services.sal.um.api.WDUMException`	Exceptions for user management

Input Fields to Set to Read-Only

Each input field has a property that indicates whether the field is read-only or ready for input. To access this property, you must know each field's identifier. See the tables below.

Input Field Identifiers

Field	Identifier
Pickup Date	`dateFromString`
Return Date	`dateToString`
Pickup Location	`pickupLocation`
Dropoff Location	`dropoffLocation`
Vehicle Type	`vehicleTypeId`

Button Identifiers

Button	Identifier
Rent a Car	`Rent`
Display all Bookings	`DisplayAll`

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental project, *TutWD_CarRental*, is displayed in the *Web Dynpro Explorer*.

Procedure

1. Choose the *Web Dynpro Explorer*.
2. Expand *TutWD_CarRental* → *Web Dynpro* → *Web Dynpro Components* → *CarRentalComp* → *Views*.
3. Open the *FormView* by selecting it with a double-click.

   The information for the initial form view appears in the right pane.
4. Choose the *Implementation* tab page.
5. Insert the necessary includes as shown below.

```
import
com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractButton
;
import com.sap.tc.webdynpro.progmodel.api.WDVisibility;
import
com.sap.tc.webdynpro.clientserver.uielib.standard.api.IWDAbstractInputF
ield;
import com.sap.tc.webdynpro.services.sal.um.api.IWDClientUser;
import com.sap.tc.webdynpro.services.sal.um.api.WDClientUser;
import com.sap.tc.webdynpro.services.sal.um.api.WDUMException;
import com.sap.tut.wd.carrental.wdp.IPrivateFormView;
```

6. Adjust the `wdDoModifyView()` method. Obtain the user's ID and insert the `hasPermission()` check. If the user does not have the permission to create reservations, then set the read-only property for each of the input fields to true. See the code sample below:

```
public static void wdDoModifyView(IPrivateFormView wdThis,
IPrivateFormView.IContextNode wdContext,
com.sap.tc.webdynpro.progmodel.api.IWDView view, boolean firstTime)
  {
    //@@begin wdDoModifyView

      try {
            IWDClientUser user = WDClientUser.getCurrentUser();
            IWDAbstractInputField fi;
            IWDAbstractButton button;
      if (user.hasPermission(new WDCarRentalPermission("WDmaintain"))
== false) {

                  fi =
(IWDAbstractInputField)view.getElement("dateFromString");
                  fi.setReadOnly(true);

                  fi =
(IWDAbstractInputField)view.getElement("dateToString");
                  fi.setReadOnly(true);

                  fi =
(IWDAbstractInputField)view.getElement("pickupLocation");
                  fi.setReadOnly(true);

                  fi =
(IWDAbstractInputField)view.getElement("dropoffLocation");
                  fi.setReadOnly(true);
```

```
                    fi =
(IWDAbstractInputField)view.getElement("vehicleTypeId");
                    fi.setReadOnly(true);

                    button =
(IWDAbstractButton)view.getElement("Rent");
                    button.setVisible(WDVisibility.NONE);

                    button =
(IWDAbstractButton)view.getElement("DisplayAll");
                    button.setVisible(WDVisibility.NONE);
               }

        } catch (WDUMException e) {
               e.printStackTrace();
        }
      //@@end
    }
```

7. Save the metadata.

Result

If the user does not have the permission to maintain reservations, then the corresponding fields are set to read only and the buttons are not visible.

Next Step:

Rebuilding and Redeploying the Project [Page 63]

Rebuilding and Redeploying the Project

Use

You have now finished with the implementation of checking permissions in the Web Dynpro client. You will now rebuild and redeploy the Web Dynpro project.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental project, *TutWD_CarRental*, is displayed in the *Web Dynpro Explorer*.
- ☐ You have completed the steps for creating the permission classes and checking the permissions in the coding.
- ☐ All modified files and metadata 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. Expand *TutWD_CarRental* → *Web Dynpro* → *Application*.
2. Select the *CarRentalApp* node, open the context menu and choose *Deploy New Archive and Run*.

   The deployment status is shown in the *Deploy Output View* section.

   Although the application is deployed and started, you still have to create the actions and deploy them to the J2EE Engine.

Result

The Web Dynpro application is deployed to the J2EE Engine.

Next Step:

Defining Actions for the Web Dynpro Project [Page 64]

Defining Actions for the Web Dynpro Project

Use

As with the actions that you created for the permissions checked in the EJB methods, you also have to create actions that apply to the permissions checked in the Web Dynpro client. For this tutorial, you will include these actions in the `actions.xml` file that you created for the J2EE application. However, depending on the scope and structure of your application, you can create a separate `actions.xml` file to be used by the Web Dynpro client application.

Prerequisites

- ☐ The Web Dynpro perspective is displayed in the SAP NetWeaver Developer Studio.
- ☐ The Web Dynpro car rental project, *TutWD_CarRental*, is displayed in the *Navigator*.

Procedure

1. Return to the *Navigator*.
2. Expand your development component project and open the `actions.xml` file that you created in the last part of the tutorial (in the *src* directory).
3. Choose the *Source* tab page from the multi-page editor.

4. Enter the XML tags for the action to access the Web Dynpro car rental application as follows:

```
...
   <ACTION NAME= "WDMaintainPremium" >
       <DESCRIPTION LOCALE= "en" VALUE= "Create Reservations for
Premium
                        and Luxury" />
      <PERMISSION CLASS=
com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPermissio
n"
        NAME= "*" VALUE= "view" />
      <PERMISSION CLASS=
"com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPermissi
on"
         NAME= "Premium" VALUE= "create" />
      <PERMISSION CLASS=
"com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPermissi
on"
         NAME= "Luxury" VALUE= "create" />
      <PERMISSION CLASS=
"com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPermissi
on"
         NAME= "Premium" VALUE= "cancel" />
      <PERMISSION CLASS=
"com.sap.engine.examples.ejb.quickcarrental.QuickReservationEjbPermissi
on"
         NAME= "Luxury" VALUE= "cancel" />
   </ACTION>

      <ACTION NAME="AccessWebDynproCarRental">
            <DESCRIPTION LOCALE="en" VALUE="Access Web Dynpro Car
Rental" />
            <PERMISSION
CLASS="com.sap.tut.wd.carrental.WDCarRentalPermission"
                  NAME="WDmaintain" />
      </ACTION>
</BUSINESSSERVICE>
```

5. Save the data.

Next Step:

Modifying the UME Roles and User Assignments [Page 66]

Modifying the UME Roles and User Assignments

Use

Once you have deployed the application and the new UME action to the J2EE Engine, modify the UME roles so that the users `Agent` and `Pr_Agent` can access the Web Dynpro car rental application and perform the corresponding tasks. `Employee` should not be allowed to perform any tasks.

Therefore, to allow the users `Agent` and `Pr_Agent` access to the Web Dynpro application, you will add the action `AccessWebDynproCarRental` to the roles `BookingAgent` and `PremiumBookingAgent`.

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 and log on as an administrator user.

   `http://localhost:50000/useradmin`

   The *User Management* screen appears.

2. Choose *Roles*.

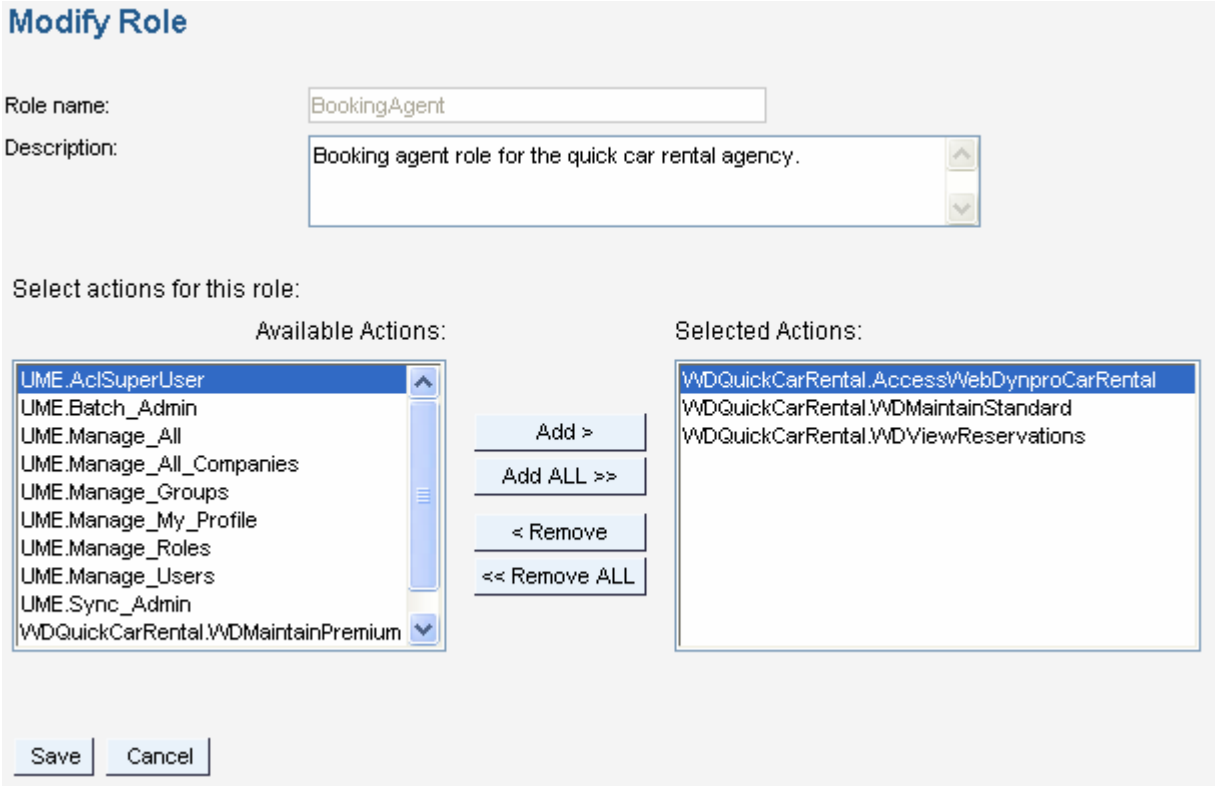
   The *Create/Maintain Roles* screen appears.

3. Search for the `BookingAgent` role, select it and choose the symbol for *Modify/View*.

   The *Modify Role* screen appears. It shows the available actions and those actions already contained in the role.

4. Select the action `WDQuickCarRental.AccessWebDynproCarRental` from the *Available Actions* and choose *Add*.

   This role is added to the list of *Selected Actions* for the role. See the figure below.



5. Save the role.

   You return to the *Create/Maintain Roles* screen.

6. Repeat for the `PremiumBookingAgent` role.

Result

The users that are assigned to these roles are allowed to access the Web Dynpro car rental application and perform tasks. Users who do not have this action in their role assignments will be able to access the Web Dynpro application, but the input fields will be set to read-only.

Next Step:

Testing the Access Protection [Page 68]

Testing the Access Protection

Use

You can now test the new authorization assignments. As in the previous section of the tutorial, the users `Agent` and `Pr_Agent` are able to create and maintain reservations, while `Employee` does not have the appropriate permissions. However, instead of receiving an error message, in this test, the input fields are set to read-only and the functions for creating or displaying reservations are not available.

Prerequisites

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

Procedure

1. Close any open Web browsers.
2. Start the Web Dynpro application. Either choose *Run* from the context menu for the *CarRentalApp* in the Developer Studio or enter the URL for the application directly in the Web browser.

   `http://localhost:50000/webdynpro/dispatcher/local/TutWD_CarRental/CarRentalApp`

   You receive a logon screen.
3. Logon as each of the users and attempt to maintain reservations.

   To make sure you obtain a new session for each user, close the Web browser between the tests for the different users.

Result

`Agent` and `Pr_Agent` should be able to maintain reservations as in the previous section of the tutorial. `Employee` should receive the initial screen for the application but all input fields should be set to read-only and the functions for creating or displaying reservations should not be visible.

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 J2EE-based quick car rental 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 J2EE-based quick car rental application, see Protecting Access to a J2EE-Based Application Using UME Permissions [SAP Library].