Oracle® Application Server Containers for J2EE

Security Guide

10*g* (9.0.4)

Part No. B10325-02

September, 2003

ORACLE®

Oracle Application Server Containers for J2EE Security Guide, 10*g* (9.0.4)

Part No. B10325-02

Primary Author: Elizabeth Hanes Perry

Contributors: Rick Sapir, Alfred Franci

Contents

List of Tables

List of Figures

List of Examples

Send Us Your Comments

Oracle Application Server Containers for J2EE Security Guide, 10*g* (9.0.4)

Part No. B10325-02

Oracle Corporation welcomes your comments and suggestions on the quality and usefulness of this document. Your input is an important part of the information used for revision.

- Did you find any errors?
- Is the information clearly presented?
- Do you need more information? If so, where?
- Are the examples correct? Do you need more examples?
- What features did you like most about this manual?

If you find any errors or have any other suggestions for improvement, please indicate the title and part number of the document, and the chapter, section, and page number (if available). You can send comments to us in the following ways:

- Electronic mail: iasdocs_us@oracle.com
- FAX: 650-506-7365 Attn: Java Platform Group, Information Development Manager
- Postal service:
  Oracle Corporation
  Java Platform Group, Information Development Manager
  500 Oracle Parkway, M/S 1op6
  Redwood Shores, CA 94065
  USA

If you would like a reply, please give your name, address, telephone number, and (optionally) electronic mail address.

If you have problems with the software, please contact your local Oracle Support Services.

Preface

This manual discusses how to make effective use of the Oracle Application Server Containers for J2EE (OC4J) security features.

This preface contains these topics:

Audience

This manual is intended for experienced Java developers, deployers, and application managers who want to understand the security features of OC4J.

Documentation Accessibility

Our goal is to make Oracle products, services, and supporting documentation accessible, with good usability, to the disabled community. To that end, our documentation includes features that make information available to users of assistive technology. This documentation is available in HTML format, and contains markup to facilitate access by the disabled community. Standards will continue to evolve over time, and Oracle Corporation is actively engaged with other market-leading technology vendors to address technical obstacles so that our documentation can be accessible to all of our customers. For additional information, visit the Oracle Accessibility Program Web site at:

`http://www.oracle.com/accessibility/`

Accessibility of Code Examples in Documentation JAWS, a Windows screen reader, may not always correctly read the code examples in this document. The conventions for writing code require that closing braces should appear on an otherwise empty line; however, JAWS may not always read a line of text that consists solely of a bracket or brace.

Accessibility of Links to External Web Sites in Documentation This documentation may contain links to Web sites of other companies or organizations that Oracle Corporation does not own or control. Oracle Corporation neither evaluates nor makes any representations regarding the accessibility of these Web sites.

Organization

This document contains:

- Chapter 1, "Introduction"
- Chapter 2, "Overview of JAAS in Oracle Application Server"
- Chapter 3, "Configuring And Deploying the JAAS Provider"
- Chapter 4, "JAAS Provider Administration Tasks"

- Chapter 5, "Using the JAZN Admintool"
- Chapter 6, "Security and J2EE Applications"
- Chapter 7, "Custom LoginModules"
- Chapter 8, "JAAS and Enterprise Manager"
- Chapter 9, "Java 2 Security"
- Chapter 10, "Password Management"
- Chapter 11, "Oracle HTTPS for Client Connections"
- Chapter 12, "EJB Security"
- Chapter 13, "J2EE Connector Architecture Security"
- Chapter 14, "Configuring CSIv2"
- Chapter 15, "Security Tips"
- Appendix A, "JAAS Provider Standards and Samples"
- Appendix B, "JAAS Provider Schemas"

Related Documentation

For more information, see these Oracle resources:

- *Oracle Application Server 10g Security Guide*
- *Oracle Application Server 10g Administrator's Guide*
- *Oracle Identity Management Concepts and Deployment Planning Guide*
- *Oracle Application Server Certificate Authority Administrator's Guide*
- *Oracle Application Server Single Sign-On Administrator's Guide*
- *Oracle Application Server Single Sign-On Application Developer's Guide*
- *Oracle Internet Directory Administrator's Guide*
- *Oracle Internet Directory Application Developer's Guide*
- *Oracle Application Server Containers for J2EE Services Guide*
- *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide*
- *Oracle Application Server Web Services Developer's Guide*
- The OC4J Javadoc

Printed documentation is available for sale in the Oracle Store at:

`http://oraclestore.oracle.com/`

To download free release notes, installation documentation, white papers, or other collateral, please visit the Oracle Technology Network (OTN). You must register online before using OTN; registration is free and can be done at:

`http://otn.oracle.com/membership/`

If you already have a username and password for OTN, then you can go directly to the documentation section of the OTN Web site at:

`http://otn.oracle.com/documentation/content.html`

For additional information, see:

- The Sun Java and J2EE Web pages, especially the Java Authentication and Authorization Service (JAAS) website at `http://java.sun.com/products/jaas/index-14.html`

Conventions

The following conventions are also used in this manual:

Convention	Meaning
. . .	Vertical ellipsis points in an example mean that information not directly related to the example has been omitted.
. . .	Horizontal ellipsis points in statements or commands mean that parts of the statement or command not directly related to the example have been omitted
boldface text	Boldface type in text indicates a term defined in the text, the glossary, or in both locations.
italic text	Italicized text indicates placeholders or variables for which you must supply particular values.
[ ]	Brackets enclose optional clauses from which you can choose one or none.

1 Introduction

This chapter describes the following topics:

- The Java 2 Security Model
- Principals and Subjects
- Authentication and Authorization
- Developing Secure J2EE Applications

For a broader description of Oracle Application Server security in middle-tier environments that connect to the Internet, see the *Oracle Application Server 10g Security Guide*. For information on Web services, see the Oracle Application Server Web Services Developer's Guide.

The Java 2 Security Model

The Java 2 Security Model enables configuration of security at all levels of restriction. This provides developers and administrators with increased control over many aspects of enterprise applet, component, servlet, and application security. The Java 2 Security Model is capability-based and enables you to establish protection domains, and set security policies for these domains.

For a tutorial on Java 2 Security, see `http://java.sun.com/docs/books/tutorial/security1.2/index.html`. For full information on Java 2 Security, see `http://java.sun.com/security`.

Principals and Subjects

Principals

A *principal* is a specific identity, such as a user named `frank` or a role named `hr`. A principal is associated with a subject upon successful authentication to a computing service. Principals are instances of classes that implement the `java.security.Principal` interface. A principal class must define a namespace that contains a unique name for each instance of the class.

Subjects

A *subject* represents a grouping of related information for a single user of a computing service, such as a person, computer, or process. This related information includes the subject's identities and security-related attributes (such as passwords and cryptographic keys).

Subjects can have multiple identities; principals represent identities in a subject. A subject becomes associated with a principal (user `frank`) upon successful authentication to a computing service—that is, the subject provides evidence (such as a password) to prove its identity.

Principals bind names to a subject. For example, a person subject, user `frank`, may have two principals:

- One binds the principal `frank doe` (name on his driver license) to the subject
- Another binds the identification principal `999-99-9999` (number on his student identification card) to the subject

Both principals refer to the same subject.

Subjects can also own security-related attributes (known as *credentials*). Sensitive credentials requiring special protection, such as private cryptographic keys, are stored in a private credential set. Credentials intended to be shared, such as public key certificates or Kerberos server tickets, are stored in a public credential set. Different permissions are required to access and modify different credential sets.

Subjects are represented by the `javax.security.auth.Subject` class.

To perform work as a particular subject, an application invokes the method `Subject.doAs(Subject, PrivilegedAction)` (or one of its variations). This method associates the subject with the current thread's `AccessControlContext` and then executes the specified request.

Authentication and Authorization

Software security depends on two fundamental concepts: authentication and authorization.

- *Authentication* is establishing the identity and credentials of a subject.

  Authentication information is stored in a *user repository*. When a subject attempts to access a J2EE application, a *user manager* looks up the subject in the user repository and verifies the subject’s identity. A user repository can be a file or a directory server, depending on your environment. The Oracle Internet Directory is an example of a user repository.

  Although each J2EE application determines which user can use the application, it is the user manager that authenticates the user’s identity using the user repository.

  OC4J supports several different authentication options; for details, see "Authentication Environments" on page 6-5.

- *Authorization* is granting privileges to an authenticated subject.

  Developers specify authorization for subjects in the application’s J2EE and OC4J-specific deployment descriptors. These deployment descriptors indicate what roles are needed to access the different parts of the application. *Roles* are the identities that each application uses to indicate access rights to its different objects. The OC4J-specific deployment descriptors provide a mapping between the logical roles and the users and groups known by OC4J.

Secure Communications

To communicate securely, applications must satisfy the following goals:

- Secure communications—the data transmitted over the network cannot be intercepted, read, or altered by a third party. OC4J supports secure communications using the HTTP protocol over the Secure Sockets Layer.
- Network authentication—clients and servers must be able to authenticate themselves to one another over the network. This is achieved using digital certificates, single sign-on, or username/password combinations.
- Identity propagation—allowing one client to act as the agent of another client, using the original client's identity.

Secure Sockets Layer

The Secure Sockets Layer (SSL) is the industry-standard point-to- point protocol which provides confidentiality, via encryption, authentication and data integrity. Although SSL is used by many protocols, it is most important for OC4J when used with the HTTP browser protocol and in the AJP link between the OHS and OC4J processes.

Certificates

Applications need to transmit authentication and authorization information over the network. A *digital certificate*, as specified by the X.509 v3 standard, contains data establishing a principal's authentication and authorization information.   A certificate contains:

- A public key, which is used in Public Key Infrastructure (PKI) operations
- Identity information (for example, name, company, country, and so on)
- Optional digital rights which grant privileges to the owner of the certificate.

Each certificate is digitally signed by a *trustpoint*. The trustpoint signing the certificate can be a certificate authority such as VeriSign, a corporation, or an individual.

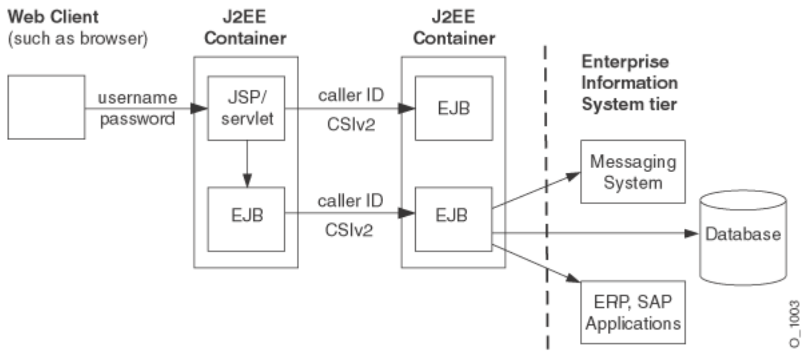
HTTPS

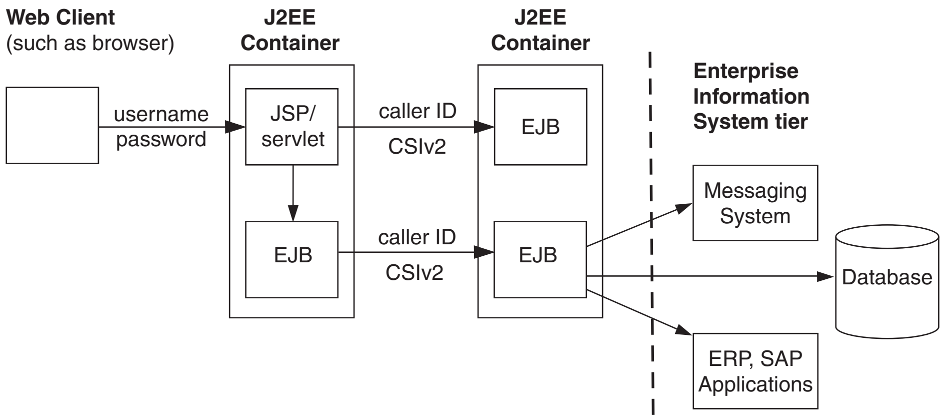
For convenience, this book uses "HTTPS" as shorthand when discussing HTTP running over SSL. Although there is an `https:` URL prefix, there is no HTTPS protocol as such.

Identity Propagation

OC4J supports propagating the identity of principals from context to context. A Web client can establish its identity to a servlet; the servlet can then use that identity to communicate with other EJBs and servlets, as illustrated in Figure 1–1.

Figure 1–1 Identity Propagation Using CSIv2





Developing Secure J2EE Applications

J2EE software development is based on a develop-deploy-manage cycle. The Oracle JAAS Provider plays an important role in the deploy-manage part of the cycle. The Oracle JAAS Provider is integrated with J2EE security. This means that developers can use a declarative security model instead of having to integrate security programmatically, unburdening the developer.

The following list summarizes the J2EE development cycle, with an emphasis on the tasks specific to developing secure applications.

1. The software developer creates Web components, enterprise beans, applets, servlets, and/or application clients.

   The JAAS Provider offers programmatic interfaces, but the developer can create components without making use of those interfaces.

2. The application assembler takes these components and combines them into an Enterprise Archive (EAR) file.

   As part of this process, the application assembler specifies JAAS Provider options appropriate to the environment.

3. The deployer installs the EAR into an instance of OC4J.

   As part of the deployment process, the deployer may map roles to users.

4. The system administrator maintains and manages the deployed application.

   This task includes creating and managing JAAS roles and users as required by the application customers.

Part I
JAAS

This section discusses Java Authentication and Authorization (JAAS), in Oracle Application Server Containers for J2EE (OC4J). JAAS is one of the core Java security technologies.

This part contains the following chapters:

2 Overview of JAAS in Oracle Application Server

This chapter introduces support for Oracle Application Server Java Authentication and Authorization Service (JAAS), in Oracle Application Server Containers for J2EE (OC4J). JAAS enables application developers to integrate authentication, authorization, and delegation services with their applications.

This chapter contains these topics:

The JAAS Provider

OracleAS supports JAAS by implementing a JAAS Provider. The JAAS Provider implements user authentication, authorization, and delegation services that developers can integrate into their application environments. Instead of devoting resources to developing these services, application developers can focus on the presentation and business logic of their applications.

Note: Some class and component names contain the word "JAZN," which is a shortened name for the OC4J JAAS Provider.

The JAAS framework and the Java 2 Security model form the foundation of JAAS. The OracleAS JAAS Provider implements support for JAAS policies. Policies contain the rules (permissions) that authorize a user to use resources, such as reading a file. Using JAAS, services can authenticate and enforce access control upon resource users. The JAAS Provider is easily integrated with J2SE and J2EE applications that use the Java 2 Security model.

Provider Types

The OC4J JAAS implementation supports two different provider types. Each provider type implements a repository for secure, centralized storage, retrieval, and administration of provider data. This data consists of realm (users and roles) and JAAS policy (permissions) information.

- XML-Based Provider

  The XML-based provider is used for lightweight storage of information in XML files. The XML-based provider stores user, realm, and policy information in an XML file, normally `jazn-data.xml`.

  Note: XML files are used as property and configuration files by both LDAP-based and XML-based provider types. However, only the XML-based provider stores users, realms, and policies in an XML file, `jazn-data.xml`.

- LDAP-Based Provider

  The LDAP-based provider is based on the Lightweight Directory Access Protocol (LDAP) for centralized storage of information in a directory. The

LDAP-based provider stores user, realm, and policy information in the LDAP-based Oracle Internet Directory.

What Is JAAS?

JAAS is a Java package that enables applications to authenticate and enforce access controls upon users. The JAAS Provider is an implementation of the JAAS interface.

JAAS is designed to complement the existing code-based Java 2 security. JAAS implements a Java version of the standard Pluggable Authentication Module (PAM) framework. This enables an application to remain independent from the authentication service.

JAAS extends the access control architecture of the Java 2 Security Model to support principal-based authorization.

This section describes JAAS support for the following authentication, authorization, and user community (realm) features. The JAAS Provider enhances some of these features.

- Login Module Authentication
- Roles
- Realms
- Policies and Permissions

See Also:

- "JAAS Framework Features" on page 2-7 for information on how the OracleAS JAAS Provider enhances the JAAS framework to explicitly define key authorization, authentication, and user community (realm) features
- JAAS documentation at the following Web site for more specific discussions of key JAAS features:

  `http://java.sun.com/products/jaas/`

Login Module Authentication

To associate a principal (such as `frank`) with a subject, a client attempts to log into an application. In login module authentication, the `LoginContext` class provides the basic methods used to authenticate subjects such as users, roles, or computing services. The `LoginContext` class consults configuration settings to determine whether the authentication modules (known as login modules) are configured for use with the particular application that the subject is attempting to access. Different login modules can be configured with different applications; furthermore, a single application can use multiple login modules.

Because the `LoginContext` separates the application code from the authentication services, you can plug a different login module into an application without affecting the application code.

Actual authentication is performed by the method `LoginContext.login()`. If authentication succeeds, then the authenticated subject can be retrieved by invoking `LoginContext.getSubject()`. The real authentication process can involve multiple login modules. The JAAS framework defines a two-phase authentication process to coordinate the login modules configured for an application.

After retrieving the subject from the `LoginContext`, the application then performs work as the subject by invoking `Subject.doAs()` or `Subject.doAsPrivileged()`.

Roles

The JAAS framework does not explicitly define roles or groups. Instead, roles or groups are implemented as concrete classes that use the interface `java.security.Principal`.

The JAAS framework does not define how to support the role-based access control (RBAC) role hierarchy, in which you can grant a role to a role.

Realms

The JAAS framework does not explicitly define user communities. However, the J2EE reference implementation (RI) defines a similar concept of user communities called *realms*. A realm provides access to users and roles (groups) and optionally provides administrative functionality. A user community instance is essentially a realm that is maintained internally by the authorization system. The J2EE RI Realm API supports user-defined realms through subclassing.

See Also:

"JAAS Provider Realm Framework" on page 4-5 for JAAS Provider enhancements to realms

Applications

The JAAS framework does not explicitly define an application or subsystem for partitioning authorization rules.

Policies and Permissions

A *policy* is a repository of JAAS authorization rules. The policy includes grants of *permissions* to principals, thus answering the question: given a grantee, what are the granted permissions of the grantee?

Policy information is supplied by the JAAS Provider. The JAAS framework does not define an administrative API for policy administration. The administrative API provided by the JAAS Provider is an Oracle extension.

Table 2–1 describes the Sun Microsystems implementation of policy file parameters.

Table 2–1 Policy File Parameters

Where	Is Defined As	Example
subject	one or more principal(s)	`duke`
codesource	*`codebase`*, *`signer`*	`http://www.example.com`, *`mysigner`*

Sun Policy Example

The following example shows a typical entry in the JAAS policy file as implemented by Sun's implementation of the JAAS policy provider:

```
grant   CodeBase "http://www.example.com",
        Principal com.sun.security.auth.SolarisPrincipal "duke"
{
        permission java.io.FilePermission "/home/duke", "read, write";
};
```

Code from `www.example.com` running as a `SolarisPrincipal` with the username `duke` has the permission that permits the executing code to read and write files in `/home/duke`.

XML-Based Example

The JAAS XML-based provider can store policy data in the file `jazn-data.xml`. In the following example, a segment of the `jazn-data.xml` file grants the `jazn.com /administrators` permission to modify realm data, to drop realms, and to create roles:

```
<!--JAZN Policy Data -->
<jazn-policy>
  <grant>
     <grantee>
        <principals>
            <principal>
                <realm>jazn.com</realm>
                <type>role</type>
                <class>oracle.security.jazn.spi.xml.XMLRealmRole
                     </class>
                <name>jazn.com/administrators</name>
             </principal>
         </principals>
     </grantee>
     <permissions>
        <permission>
           <class>oracle.security.jazn.policy.AdminPermission</class>
           <name>oracle.security.jazn.realm.
               RealmPermission$jazn.com$modifyrealmmetadata</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>
            <name>oracle.security.jazn.realm.
                RealmPermission$jazn.com$droprealm</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>
            <name>oracle.security.jazn.realm.RealmPermission$jazn.
                com$createrole</name>
        </permission>
        <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>createrealm</actions>
          </permission>
     </permissions>
  </grant>
</jazn-policy>
```

See Also:

- "Sample jazn-data.xml Code" on page A-2 to view a complete `jazn-data.xml` file.

JAAS Framework Features

Table 2–2 contains the JAAS framework features implemented by the Oracle Application Server JAAS Provider.

Table 2–2 JAAS Provider Features

Feature	Description	See Also
Authentication	■ Integrates with Oracle Application Server Single Sign-On (SSO) for SSO login authentication in J2EE application environments. ■ Supplies an out-of-the-box `RealmLoginModule` class for non-SSO environments, such as OracleAS Core or Java Edition ■ Supports any JAAS-compliant custom `LoginModule`	Chapter 6, "Security and J2EE Applications"
Declarative Model	■ Integrates J2EE deployment descriptors, such as. `web.xml`, with JAAS security ■ Supports programmatic model as well	Chapter 3, "Configuring And Deploying the JAAS Provider"
Role-based access control (RBAC)	■ Provides centralized role-based access control, including support for hierarchical roles	"Role-Based Access Control (RBAC)" on page 2-12
Realms	■ Organizes users and roles (groups) around user communities. An Oracle API package (`oracle.security.jazn.realm`) is provided to support user and role management. This API includes a `RealmPrincipal` interface that extends from `java.security.Principal` and associates a realm with users and roles.	"Realms" on page 2-4 "JAAS Provider Realm Framework" on page 4-5
Management	■ Manages settings and data using command-line tool (Admintool) or programmatic level APIs ■ Supports a centrally managed provider type with Oracle Internet Directory	Chapter 4, "JAAS Provider Administration Tasks"
`JAZNUserManager`	■ Provides an implementation of the OC4J `UserManager` that integrates with both the XML-based and the LDAP-based providers.	"OC4J and the JAAS Provider" on page 6-3 Chapter 6, "Security and J2EE Applications"

User Managers

OC4J security employs a user manager to authenticate and authorize users and groups that attempt to access a J2EE application. You base your choice of user manager on performance and security needs.

All `UserManager` classes implement the `com.evermind.security.UserManager` interface. `UserManager` classes manage users, groups, and passwords through methods such as `createUser()`, `getUser()`, and `getGroup()`.

OC4J provides two predefined user managers, `JAZNUserManager and XMLUserManager`. `JAZNUserManager` supports both XML-based and LDAP-based providers. We recommend using `JAZNUserManager` because it is based on the JAAS specification and is integrated with Oracle Application Server Single Sign-On and Oracle Internet Directory. `JAZNUserManager` is the default security provider, because it offers powerful and flexible security control. Customers can also supply their own classes that implement the `UserManager` interface.

Note: For a discussion of creating a custom `UserManager`, see `http://otn.oracle.com/sample_code/tech/xml/xmlnews/News_Security.html`

Table 2–3 lists the user managers provided by OC4J.

Table 2–3 OC4J User Managers And Repositories

User Manager Class	User Repository
`oracle.security.jazn.oc4j.JAZNUserManager`	Two types: ▪ using the XML-based provider — `jazn-data.xml` ▪ using the LDAP-based provider—Oracle Internet Directory
`com.evermind.server.XMLUserManager`	`The principals.xml` file
Custom user manager	Customized user repository

See "Specifying UserManagers" on page 2-11 for directions on how to use Enterprise Manager to define the default `UserManager` for all applications or a single `UserManager` for a specific application.

The following sections describe the JAZN and XML user managers:

- Using JAZNUserManager
- Using XMLUserManager

Using JAZNUserManager

The `JAZNUserManager` class is the default user manager. The primary purpose of the `JAZNUserManager` class is to leverage the JAAS Provider as the security infrastructure for OC4J.

There are two JAAS Providers supplied with OC4J security: XML-based and LDAP-based.

- The XML-based provider is a fast, lightweight implementation of the JAAS Provider API. This provider type uses XML to store user names and encrypted passwords. The user repository is stored in the `jazn-data.xml` file, in a location specified in the jazn.xml file. For details, see Chapter 3, "Configuring And Deploying the JAAS Provider".

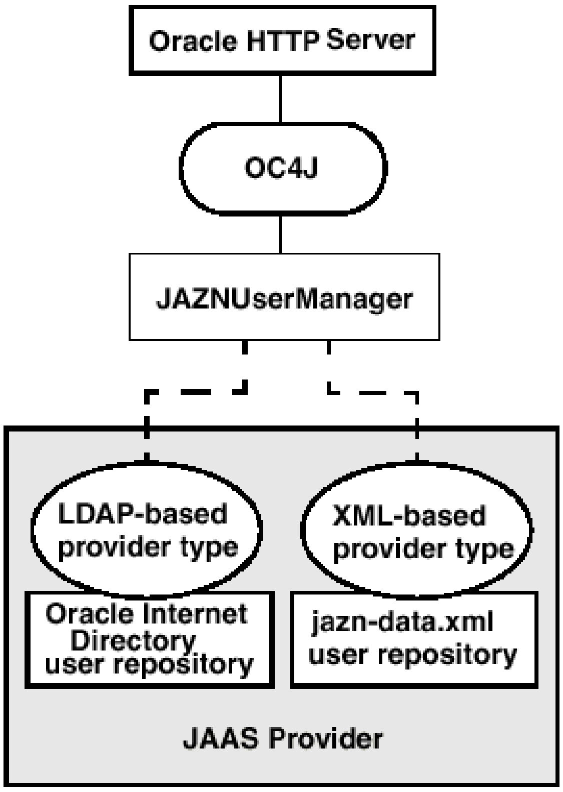
  Select JAZN-XML as the user manager in the Enterprise Manager. Configure its users, roles, and groups using the JAZN Admintool. For information on the Admintool, see Chapter 5, "Using the JAZN Admintool".

- The LDAP-based provider is scalable, secure, enterprise-ready, and integrated with Single Sign-On. The LDAP-based provider is the only JAAS Provider that supports Single Sign-On.

  Select JAZN-LDAP as the user manager in the Enterprise Manager. Configure its users and groups using the Delegated Administrative Service (DAS) from Oracle Internet Directory. The user repository is an Oracle Internet Directory instance, which requires that the application server instance be associated with an infrastructure. If it the server is not associated with an Oracle Internet Directory instance, then the LDAP-based provider is not a security option.For information on using the Enterprise Manager, see Chapter 8, "JAAS and Enterprise Manager".

Figure 2–1 shows the two different JAAS Providers supplied with OC4J.

Figure 2–1 OC4J Security Architecture Under the JAZNUserManager Class



Using XMLUserManager

The `XMLUserManager` class is a simple user manager that manages users, groups, and roles in an XML-based system. It stores user passwords in the clear, and therefore is not as secure as the `JAZNUserManager`. All `XMLUserManager` configuration information is stored in the `principals.xml` file, which is the user repository for the `XMLUserManager` class.

Note: The `XMLUserManager` class is supported for backward compatibility only. Oracle recommends that you use one of the JAAS provider types.

Specifying UserManagers

The user manager, employing the user name and password, verifies the user's identity using information in the user repository. The user manager contains your definitions for users, groups, or roles. The default user manager is the `JAZNUserManager`.

You can define a user manager for all applications or for specific applications.

- Global user manager—The global (default) user manager is inherited by all applications that have not defined a specific user manager.
- Specific user manager—This user manager is defined solely for a single application. It is not used by any other application.

Note: Within a single OC4J instance you can specify different values for the application-specific UserManager instance and the global UserManager instance. When you do this, we recommend that you not mix custom UserManagers and Oracle-supplied UserManagers. You can use different custom UserManagers for the application and the global instance, and you can use different Oracle-supplied UserManagers for the application and the global instance, but you should avoid using a custom UserManager for the one instance and an Oracle-supplied UserManager for the other.

- In some cases, if an application inherits from another application instead of inheriting from the global application, then the application's parent user manager will be the global `UserManager` instance instead of the `UserManager` instance specified in the parent application.

Capability Model of Access Control

The *capability model* is a method for organizing authorization information. The JAAS Provider is based on the Java 2 Security Model, which uses the capability model to control access to permissions. With the capability model, authorization is associated with the principal (a user named `frank` in the following example). Table 2–4 shows the permissions that user `frank` is authorized to use:

Table 2–4 User Permissions

User	Has These File Permissions
`frank`	Read and write permissions on a file named `salaries.txt` in the `/home/user` directory

When user `frank` logs in and is successfully authenticated, the permissions described in Table 2–4 are retrieved from the JAAS Provider (whether the LDAP-based Oracle Internet Directory or XML-based provider) and granted to user `frank`. User `frank` is then free to execute the actions permitted by these permissions.

Role-Based Access Control (RBAC)

RBAC enables you to assign permissions to roles. You grant users permissions by making them members of appropriate roles. Support for RBAC is a key JAAS feature. This section describes the following RBAC features:

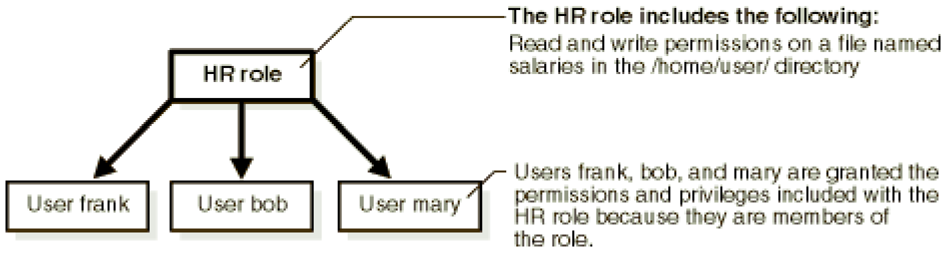
- Role Hierarchy
- Role Activation

Role Hierarchy

RBAC simplifies the management problems created by direct assignment of permissions to users. Assigning permissions directly to multiple users is potentially a major management task. If multiple users no longer require access to a specific permission, you must individually remove that permission from each user.

Instead of directly assigning permissions to users, permissions are assigned to a role, and users are granted their permissions by being made members of that role. Multiple roles can be granted to a user. A role can also be granted to another role, thus forming a *role hierarchy* that provides administrators with a tool to model enterprise security policies. Figure 2–2 provides an example of role-based access control.

Figure 2–2 Role-Based Access Control


The HR role includes the following:
Read and write permissions on a file named salaries in the /home/user/ directory
HR role
User frank
User bob
User mary
Users frank, bob, and mary are granted the permissions and privileges included with the HR role because they are members of the role.

When a user's responsibilities change (for example, through a promotion), the user's authorization information is easily updated by assigning a different role to the user instead of a massive update of access control lists containing entries for that individual user.

For example, if multiple users no longer require write permissions on a file named `salaries` in the `/home/user` directory, those privileges are removed from the `HR` role. All members of the `HR` role then have their permissions and privileges automatically updated.

Role Activation

A user is typically granted multiple roles. However, not all roles are enabled by default. An application can selectively enable the required roles to accomplish a specific task in a user session with the `run-as` security identity and `Subject.doAS()`. Selectively enabling roles upholds the principle of least privilege: the application is not enabling permissions or privileges unnecessary for the task. This limits the damage that can potentially result from an accident or error.

3 Configuring And Deploying the JAAS Provider

This chapter describes the configuration tasks you must perform to use the Java Authentication and Authorization (JAAS) Provider in a Java 2 Platform, Enterprise Edition (J2EE) environment under Oracle Application Server Containers for J2EE (OC4J). It also has a high-level overview of J2EE and OC4J deployment descriptors. This chapter contains these topics:

Notes: This chapter does not describe how to configure OC4J as a whole. See the *Oracle Application Server Containers for J2EE User's Guide* for those instructions.

To use the LDAP provider, you must install Oracle Application Server and Oracle Internet Directory (OID). For details, see the Oracle Application Server 10g Installation Guide.

Before using your JAAS-based application, you must configure the JAAS Provider components. This chapter discusses configuring JAAS in an OC4J and J2EE environment.

LDAP-Based Provider Environment Settings

The JAAS LDAP-based provider depends on the OID client library `ldapjclnt9.jar` in the directory:

```
[ORACLE_HOME]/jlib
```

The OID client library depends on a native library (for example, `libldapjclnt9.so` in Solaris) in the directory:

```
[ORACLE_HOME]/lib
```

These dependencies affect how you launch the JAZN Admintool, especially when you are using the LDAP-based provider. Before launching the Admintool, you must:

- Ensure that your classpath contains:

```
[ORACLE_HOME]/jlib/ldapjclnt9.jar
```

- Ensure that the operating-system-specific environment variable controlling loading of dynamic libraries (for example, `LD_LIBRARY_PATH` in Solaris) includes:

  ```
  [ORACLE_HOME]/lib
  ```

When you manage OC4J with the Enterprise Manager, it sets these two variables automatically.

J2EE Deployment Descriptors

J2EE provides the following XML deployment descriptors that have security implications:

Table 3–1 J2EE Deployment Descriptors

Filename	Security Tags
`web.xml`	For Web applications, servlets, and Gasps, module-level security roles, security constraints, and authorization constraints
`ejb-jar.xml`	For EJBs, module-level and method-level security roles, security constraints, and authorization constraints
`application.xml` (the file contained in an application's EAR file)	For applications, application-level descriptors for multiple modules

Note: For a full discussion of these descriptors, see the *Oracle Application Server Containers for J2EE User's Guide*; this chapter discusses only the security-related aspects of these descriptors.

OC4J Deployment Descriptors

OC4J provides the following container-specific XML deployment descriptor files that have security implications:

Table 3–2 OC4J Configuration Files

File	Security-related Tags
The global `application.xml`	mappings, user manager, `<jazn>`, `<security-role-mapping>`, `<jazn-web-app>` embedded in `<jazn>`.
`orion-web.xml`	`<jazn-web-app>`
`orion-application.xml`	`<jazn>` tag, `<jazn-web-app>` embedded in `<jazn>`.

JAAS Provider Configuration Files

The JAAS provider stores configuration information in various files; sample configuration files are provided with the product. You edit the JAAS Provider configuration files using the JAZN Admintool or manually, using a text editor.

Caution: If you edit these configuration files manually and you are not running OC4J standalone, you must run `dcmctl updateconfig` to propagate your changes throughout the cluster.

You configure the JAAS Provider in all of the files listed in Table 3–2, as well as in the following JAAS-specific files:

- `jazn.xml`

  The JAAS Provider configuration file; this specifies the default configuration for the JAAS provider, including whether the provider is LDAP-based (uses OID as the data store) or XML-based (uses `jazn-data.xml` as the data store).

- `jazn-data.xml`

  The XML-based provider stores user, role, and policy information in this file; you edit it with the JAZN Admintool.

Note: If you use the LDAP-based provider, you use DAS to manage users and groups. See the Oracle Internet Directory Administrator's Guide. for details.

- `java2.policy`

  The standard Java 2 policy file granting permissions to codebases. This file is located in `$ORACLE_HOME/j2ee/home/config`.

Specifying JAAS as the Policy Provider (Optional)

If you use the JVM shipped with Oracle Application Server 10*g*, the Oracle JAAS Provider is automatically specified as the JAAS policy provider. If you use another JVM, you must explicitly specify `oracle.security.jazn.spi.PolicyProvider` as the policy provider. By default, the JVM uses the Sun JAAS provider.

To specify the JAAS Provider as the policy provider:

1. Add the following information to the end of the `$JAVA_HOME/jre/lib/security/java.security` file:

```
auth.policy.provider=oracle.security.jazn.spi.PolicyProvider
login.configuration.provider=oracle.security.jazn.spi.LoginConfigProvider
```

Locating jazn.xml

The file `jazn.xml` is the configuration file for both the XML-based and LDAP-based JAAS providers. The JAAS Provider must locate a valid `jazn.xml` file before it can begin running.

When the JAAS provider starts up, it searches for `jazn.xml` in order through the directories specified by:

1. `oracle.security.jazn.config` (system property)
2. `java.security.auth.policy` (system property)
3. *`$J2EE_HOME`*`/config` (*`$J2EE_HOME`* is specified by the system property `oracle.j2ee.home`)
4. *`$ORACLE_HOME`*`/j2ee/home/config` (*`$ORACLE_HOME`* is specified by the system property `oracle.home`)
5. `./config`

The JAAS provider stops searching after locating a `jazn.xml` file. If no file is found, you receive the error message "`JAZN has not been properly configured.`"

The <jazn> Tag

You use the `<jazn>` tag to configure the JAAS Provider. The `<jazn>` tag can appear in any of the following locations

- The application's `orion-application.xml`
- The global `application.xml`
- `jazn.xml`

A sample `orion-application.xml` file with all attributes and property names specified is provided in "Specifying auth-method in orion-application.xml" on page 3-13.

The tag supports different attributes depending on whether you are using the XML-based Provider or the LDAP-based provider. This section discusses the two separately, in the following sections:

- The <jazn> Tag and the XML-Based Provider
- The <jazn> Tag and the LDAP-Based Provider
- The <property> Subelement Of <jazn>

Note: You cannot edit the `<jazn>` tag using Enterprise Manager.

The <jazn> Tag and the XML-Based Provider

When you are using the XML-based Provider, the `<jazn>` tag supports the attributes shown in Table 3–3.

Table 3–3 (XML-Based Provider) The <jazn> Tag In orion-application.xml

Attribute	Value (default is bold)	Example
`provider`	**`XML`** or `LDAP`	`provider="XML"`
`location`	(Required) Path to file containing provider data. This can be an absolute path, or a path relative to the `jazn.xml` file. The JAAS Provider first looks for `jazn-data.xml` in the directory containing `jazn.xml`.	`location="./jazn-data.xml"`

Table 3–3 (XML-Based Provider) The <jazn> Tag In orion-application.xml

Attribute	Value (default is bold)	Example

`persistence`	`NONE`
	Do not persist (write) changes to `jazn-data.xml`.
	`ALL`
	Persist changes after every modification.
	`VM_EXIT`
	Persist changes when JVM exits.
		`persistence="ALL"`

`default-realm`	The realm used whenever an authentication or authorization request does not specify a realm explicitly. This attribute is not needed if you have configured only one realm in the repository.	`default-realm="`*`myrealm`*`"`
`config`	If a `config` attribute appears, the JAAS provider reads all provider properties from the file specified in the pathname. This attribute cannot be combined with any other attribute; it must appear alone.	`config="./jazn.xml"`

See: "Specifying Authentication (auth-method)" on page 3-11 for information on the `<jazn-web-app>` element and its attributes `auth-method`, `runas-mode`, and `doasprivileged-mode`.

The <jazn> Tag and the LDAP-Based Provider

You configure your application to use LDAP-based provider by adding an entry to the `orion-application.xml` file similar to the following example:

```
<jazn provider="LDAP"/>
```

This assumes that the OC4J instance has been properly associated with OID using either the installer or Enterprise Manager.

When you associate an OC4J instance with an Oracle Application Server Infrastructure (including the Oracle Internet Directory (OID)), your application can leverage the LDAP-based provider for central management of users. You can specify the use of the LDAP-based provider in several different configuration files (see Chapter 3, "Configuring And Deploying the JAAS Provider" of the Oracle Application Server Containers for J2EE Security Guide)

If you specify the LDAP-based provider globally in the `application.xml` configuration file, then you must set up the following users and groups in OID DAS:

- An `administrators` group
- An `admin` user that is a member of the `administrators` group

You must then grant the following permissions to the `administrators` group using the JAZN Admintool:

- `com.evermind.server.AdministrationPermission ("administration")`
- `com.evermind.server.rmi.RMIPermission("login")`

You can set additional attributes and properties. The `<jazn>` tag in `orion-application.xml` has the attributes shown in Table 3–4.

Table 3–4 (LDAP-Based Provider) The <jazn> Tag in orion-application.xml

Attribute	Value (default is bold)	Example
`provider`	**`XML`** or `LDAP` (this attribute can also be specified	`provider="LDAP"`
`location`	The URL of an LDAP server.	Avoid using. See Note.
`default-realm`	The realm used whenever an authentication or authorization request does not specify a realm explicitly. This attribute is not needed if you have configured only one realm.	`default-realm="us"`
`config`	If a `config` attribute appears, the JAAS provider reads all provider properties from the file specified in the pathname. This attribute cannot be combined with any other attribute; it must appear alone.	`config="`*`configpath`*`"`
`persistence`	**`ALL`** Persist changes after every modification. "The LDAP-based Provider always sets this value to `ALL`.	

Notes:

- If you do not specify the `provider` attribute in the `<jazn>` tag in `orion-application.xml`, you can specify it in `jazn.xml`.
- The `provider` and `location` attributes can be edited using Enterprise Manager; all other attributes must be edited by hand.
- The JAAS LDAP-based provider does not depend on the `location` attribute in the `<jazn>` element. If you do not specify this attribute (it is not specified by default), then the JAAS runtime obtains infrastructure information from configured system settings. The system settings are configured when you use Enterprise Manager to associate a mid-tier to an infrastructure. To take advantage of this feature, we recommend that you do not set the `location` attribute in the `<jazn>` element in any JAAS configuration files, including `jazn.xml` and `orion-application.xml`.

The `<jazn>` tag can contain a `<jazn-web-app>` tag that specifies authentication information.

> **See:** "Specifying Authentication (auth-method)" on page 3-11 for information on the `<jazn-web-app>` element and its attributes `auth-method`, `runas-mode`, and `doasprivileged-mode`.

If you want to permit anonymous, read-only logins to the application, do not assign values to these attributes.

A sample `orion-application.xml` file with all attributes and property names specified is provided in "Specifying auth-method in orion-application.xml" on page 3-13.

The <property> Subelement Of <jazn>

The `<jazn>` tag can contain a `<property>` element. Most of these properties can be set only on a per-VM business, in the global `jazn.xml`. The only exceptions are `ldap.password` and `ldap.user`. The syntax of the `<property>` subelement is:

```
<property name="propname" value="propvalue">
```

Table 3–5 lists the supported properties.

Table 3–5 Values For <property> Element of <jazn> Tag

Property Name	Default	Value
`classpath`		This property, when set, tells the JAAS Provider where to look for third-party classes and JAR files if they cannot be found elsewhere. `classpath="./loginmodules"`
`external.synchronization`	**`false`**	When set to `true`, the XML-based JAAS Provider monitors its data repository (normally `jazn-data.xml`) for external updates. When you add new users outside your application (for example, by using the Admintool), these users will be picked up automatically by OC4J. If you do not set this property, you must stop and restart OC4J in order to make the new users visible. Set to `false` for production environments.
`role.mapping.dynamic`	**`false`**	`role.mapping.dynamic="true"` When set to `true`, the JAAS Provider does authorization checks based on the current `Subject` instead of using static configurations. When set to `false`, the JAAS Provider uses static configurations as the basis for authorization checks.
`jndi.ctx_pool.init_size`	**`5`**	Initial size for JNDI/LDAP connection pool.
`jndi.ctx_pool.inc_size`	**`10`**	Pool increment size for JNDI/LDAP connection pool — number of connections added to pool whenever the supply of connections in the pool is exhausted.
`ldap.cache.*`	See "Configuring the JAAS Provider To Use SSL With Oracle Internet Directory" on page 3-18 and Table 3–9 for details.	
`ldap.connect.max.retry`	**`5`**	Number of times the JAAS Provider attempts to create an LDAP connection before giving up.
`ldap.connect.sleep`	**`5000`**	Number of milliseconds the JAAS Provider waits before retrying a failed LDAP connection attempt.

Table 3–5 Values For <property> Element of <jazn> Tag

Property Name	Default	Value
`ldap.password`		Obfuscated password for the LDAP user name. For example: `{903}oZZYqmGc/iyCaDrD4qs2FHbXf3LAWtMN` See "Password Obfuscation In jazn-data.xml and jazn.xml" on page 10-2 for details on obfuscation.
`ldap.user`		LDAP username or DN. For example: `orcladmin` or cn=`orcladmin`
`ldap.walletloc`		Pathname for the Oracle Wallet.
`ldap.walletpwd`		Obfuscated password for the Oracle Wallet. See "Password Obfuscation In jazn-data.xml and jazn.xml" on page 10-2 for details on obfuscation.

Notes:

- To specify a cleartext `ldap.password` or `ldap.walletpwd` entry, place an exclamation point (`!`) in front of the password value:, as in `!welcome.` The entry will be automatically obfuscated.
- When set to `false`, the XML-based JAAS Provider does not check the file system periodically for external updates to the XML-based data repository. To enable dynamic external synchronization, set the `external.synchronization` property to `true` in the `<jazn>` tag.

Specifying Authentication (auth-method)

You specify the authentication method (`auth-method`) in one of several configuration files, using either the `<jazn-web-app>` or `<login-config>` elements.

Specifying auth-method in web.xml

To specify authentication method at the global level, you use the `<login-config>` element of `web.xml`. For example:

```
<login-config>
   <auth-method>BASIC</auth-method>
</login-config>
```

In `web.xml`, `auth-method` can have the values shown in Table 3–6:

Table 3–6 Values for auth-method in web.xml

Setting	Meaning
`BASIC` (default)	The application uses basic authentication, the standard J2EE authentication.
`FORM`	The application uses form-based authentication.
`DIGEST`	The application uses HTTP DIGEST authentication
`CLIENT-CERT`	The application requires the client to supply its own certificate for use with SSL.

These values can be overridden at the application level by using the `<jazn-web-app>` element in `orion-web.xml` or `orion-application.xml`.

Specifying auth-method in orion-web.xml and orion-application.xml

The `auth-method` supplied in the top-level `<jazn-web-app>` element overrides the `auth-method` in `web.xml`.

There is only one possible value for the `auth-method` in `orion-web.xml` and `orion-application.xml`: `SSO`, meaning that the application uses Oracle Single Sign-On.

A sample entry for `orion-web.xml` would look like:

```
<jazn-web-app
    auth-method="SSO"
/>
```

A sample entry for `orion-application.xml` would look like:

```
<jazn provider="LDAP"
    <jazn-web-app auth-method="SSO"/>
</jazn>
```

Specifying auth-method in orion-application.xml

The `auth-method` supplied in the `<jazn-web-app>` element of the `<jazn>` element overrides the `auth-method` in web.xml. There are two possible values: `SSO`, meaning that the application uses Oracle SSO, and `BASIC`, meaning that the application uses whatever authentication method is specified in `web.xml.` A sample entry would look like:

```
<jazn provider="XML"
   location="jazn-data.xml"
   default-realm="JAZN.com"
   persistence="ALL"

<!-- default values for this application -->
   <jazn-web-app
      auth-method="SSO"
    />
</jazn>
```

Configuring Servlet Authorization (runas-mode and doasprivileged-mode) in <jazn-web-app>

If you want a servlet to be invoked using `subject.doAs()` or `subject.doAsPrivileged()`, you must set the `runas-mode` and `doasprivileged-mode` attributes of the `<jazn-web-app>` element in the `orion-web.xml` or `orion-application.xml` files.

- `subject.doAs()` invokes the servlet using the privileges of a particular *subject*. A subject is defined by an instance of the `javax.security.auth.Subject` class and includes a set of facts regarding a single entity, such as a person. Such facts include identities and security-related attributes, such as passwords and cryptographic keys. The JAAS Provider passes in the `Subject` instance in the method call.

  When the `doAs()` method is used, an `AccessControlContext` instance is retrieved from the current thread (from the server).

- `subject.doAsPrivileged()` uses the privileges of a particular subject without being limited by the access-control restrictions of the server.

  When the `doAsPrivileged()` method is used, the JAAS Provider invokes the method with a null `java.security.AccessControlContext` instance, in order to start the action fresh and execute the servlet without the restrictions of the current server `AccessControlContext` instance.

`runas-mode` and `doasprivileged-mode` control whether the servlet is invoked with `subject.doAsPrivileged()` or `subject.doAs()`. By default, `runas-mode` is set to `false`, which means that neither `subject.doAsPrivileged()` or `subject.doAs()` is invoked.

Note: `runas-mode` is unrelated to the `servlet.runAs` method.

Table 3–7 runas-mode and doasprivileged-mode Settings

If runas-mode is Set To	And doasprivileged-mode Is Set To	Then the servlet is invoked with:
`false` (default)	`true` or `false`	No special privileges
`true`	`true` (default)	`subject.doAsPrivileged()`
`true`	`false`	`subject.doAs()`

Thus, to have your servlet invoked using `subject.doAsPrivileged()` you should have a `<jazn-web-app>` element that looks like this:

```
<jazn-web-app
    auth-method="SSO"
    runas-mode="true"
    doasprivileged-mode="true"
/>
```

Mapping Security Roles In Servlets (run-as)

You can map J2EE security roles to JAAS roles using OC4J groups. This enables your application to run with the privileges of the security role or specific `RealmPrincipal` class. The following tasks pertain to both kinds of privileges.

Note: The `run-as` element is unrelated to `runas-mode`.

If the `run-as` element is specified, then the `<role-name>` maps to a security role already defined for the Web application.

The following steps assume that `sr_manager` has already been defined as a security role in `web.xml` as follows:

```
<security-role>
   <role-name>sr_manager</role-name>
</security-role>
```

To map J2EE security roles to JAAS roles:

1. Specify the `run-as` element within the `<servlet>` tag to run as the specific J2EE security role or specific `RealmPrincipal` class in the `web.xml` file

   For example, to run as the security role `sr_manager`:

   ```
   <servlet>
     <servlet-name>DevGroup</servlet-name>
     <servlet-class>DevGroupServlet</servlet-class>
    <!--  run as security role "sr_manager" -->
       <run-as>
         <role-name>sr_manager</role-name>
       </run-as>
   </servlet>
   ```

2. Define a JAAS `<role>` element in the `jazn-data.xml` file:

   For example, `developer` is defined within a role :

```
<roles>
    <role>
     <name>developer</name>
     <members>
      <member>
   <type>user<type>
   <name>john<name>
       </member>
      </members>
    </role>
   </roles>
```

An XSD schema for `jazn-data.xml` is provided in Appendix B, "JAAS Provider Schemas".

3. Integrate the definitions created in Step 1 and Step 2. Use OC4J groups in the `orion-application.xml` file as follows:

- Map the role-name defined in the web.xml file as a security role (sr_manager).
- Map the role defined in jazn-data.xml as a OC4J group name (developer).

For example, the sr_manager security role is mapped to the group named developer in the JAAS Provider:

```
<security-role-mapping name="sr_manager">
  <group name="developer" />
</security-role-mapping>
```

Because the developer group is mapped to the J2EE security role sr_manager, the user (john in this example) has access to the application resources defined by the sr_manager role.

See Also: ▪

- Java Servlet Specification, available at http://java.sun.com/products/servlet/download.html
- Chapter 6, "Security and J2EE Applications"

Configuring RealmLoginModule

The RealmLoginModule class is the default LoginModule that is configured through the jazn-data.xml file. The RealmLoginModule class authenticates user login credentials before the user can access J2EE applications. Authentication is performed using OC4J container-based authentication (HTTP BASIC, FORM, and so on). You do not need to enable the RealmLoginModule class if your application uses SSO authentication.

See Also: Oracle Application Server 10g Installation Guide for SSO configuration tasks.

You can enable RealmLoginModule either using the JAZN Admintool or by editing jazn-data.xml. For details on using the Admintool, see "Adding and Removing Login Modules" on page 5-7.

Enabling RealmLoginModule Using A Text Editor

Use a text editor to modify the login configuration file jazn-data.xml where needed.

The default configuration for the `RealmLoginModule` class setting in the `jazn-data.xml` file is as follows:

```
<!DOCTYPE jazn-data (View Source for full doctype...)>
 <jazn-data>
    .
    .
    .
<!--  Login Module Data -->
 <jazn-loginconfig>
  <application>
     <name>JAZNUserManager</name>
    <login-modules>
      <login-module>
   <class>oracle.security.jazn.realm.RealmLoginModule</class>
   <control-flag>required</control-flag>
   <options>
     <option>
       <name>addRoles</name>
       <value>true</value>
     </option>
   </options>
       </login-module>
     </login-modules>
   </application>
  </jazn-loginconfig>
 </jazn-data>
```

The `<login-module>` tag supports the following `<option>` values:

Table 3–8 RealmLoginModule Options

Name	Meaning	Default
`debug`	If set to `true`, prints debugging messages.	`false`
`addRoles`	If set to `true`, the `RealmLoginModule adds` all directly granted roles of the user to the Subject after successful authentication.	`true`
`addAllRoles`	If set to `true`, the `RealmLoginModule adds` all directly or indirectly granted roles of the user to the Subject after successful authentication.	`true`
`storePrivateCredentials`	If set to `true`, the `RealmLoginModule adds` all private credentials (for instance, password credentials) to the Subject after successful authentication.	`false`

Table 3–8 RealmLoginModule Options

Name	Meaning	Default
`supportCSIv2`	If set to `true`, the `RealmLoginModule` supports CSIv2.	`false`
`supportNullPassword`	If set to `true`, the `RealmLoginModule` does not check to see if the supplied password is null or empty. If set to `false`, authentication fails if the supplied password is null or empty.	`false`

See Also: The JAAS Provider Javadoc.

"Adding and Removing Login Modules" on page 5-7

Configuring the JAAS Provider To Use SSL With Oracle Internet Directory

This section discusses configuring the JAAS provider to use SSL with OID. For information on how to configure OID to use SSL, see the Oracle Internet Directory Administrator's Guide and *Servlet Developer's Guide*.

There are three ways to use SSL to communicate with OID:

1. NULL authentication—data are encrypted with the Anonymous Diffie-Hellman cipher suite, but no certificates are used for authentication.

   Note: See Table 11–1, "Cipher Suites Supported By OracleSSL" for a list of supported cipher suites.

2. Server-side authentication only (one-way authentication)—server authenticates itself to client with a digital certificate, but client does not authenticate itself.
3. Client and server authentication (two-way authentication)—both client and server authenticate themselves using digital certificates.

For NULL authentication (case 1), add a `<property>` tag to the `<jazn>` tag to specify a protocol (note that you do not specify a wallet location or password, because NULL authentication does not use certificates):

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jazn provider="LDAP" location="ldap://pichan-sun.us.oracle.com:5000"
default-realm="us">
```

```
   <property name="ldap.protocol" value="ssl"/>

</jazn>

<property name="ldap.protocol" value="ssl"/>
```

For either one-way or two-way authentication (cases 2 and 3), use the `<property>` tag to specify protocol, wallet location, and wallet password:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jazn provider="LDAP" location="ldap://pichan-sun.us.oracle.com:5000"
default-realm="us">

   <property name="ldap.protocol" value="ssl"/>
   <property name="ldap.walletloc"
value="/private/oracle/oid/ssl2/testwallet.txt"/>
   <property name="ldap.walletpwd" value="!welcome1"/>

</jazn>
```

Configuring For EJB RMI Client Access

(LDAP-Based Provider Only)

To enable fat client access to EJBs via RMI, you must grant the correct permissions using the JAZN Admintool. You must grant the RMIPermission (login) to your user or role. For example:

```
java -jar jazn.jar -grantperm myrealm -role administrators \
  com.evermind.server.rmi.RMIPermission login
```

Configuring Caching (LDAP-Based Provider Only)

The LDAP-based JAAS Provider supports caching, providing improved performance and scalability. There are three separate caches:

- Policy cache, which stores grantees and permissions
- Realm cache, which stores realms, users and roles, and a role graph.
- Session cache, which stores users and role graphs in an HTTP session object. (This cache is available only to web-based clients with cookies enabled.)

The caching service maintains a global HashMap, which is used to store and retrieve cached objects. A daemon thread runs periodically in the background to invalidate and clean up expired objects in the HashMap. Objects in the cache expire based on a time-to-live algorithm; expiration time can be set with the cache properties, described in Table 3–9.

Note: Only the LDAP-based Provider provides these caches. The XML-based Provider defaults to caching the entire XML document.

Session Cache Details

HttpSession objects persist for the duration of the server-side session. An application can terminate a session explicitly, by invoking HttpSession.invalidate(); a container can terminate a session based on the <session-timeout> value.

See Also: The Oracle HTTP Server Administrator's Guide for more information about session support in OC4J.

The default session timeout for the OC4J server is 20 minutes. You can change this default by editing web.xml and changing the <session-timeout>.

```
<session-config>
  <session-timeout>10 </session-timeout>
</session-config>
```

The JAAS Provider invokes HttpSession.invalidate() to invalidate the session explicitly as appropriate. For example, when WebSSOUtil.logout() is invoked, the JAAS Provider invalidates the session.

Note: Objects stored in an HttpSession instance must implement the java.io.Serializable interface in order to be deployed with the <distributable /> flag in web.xml.

Disabling Caching

Caching is enabled by default. You should disable the caches when performing management and administrative tasks. In particular:

- Disable the policy cache when managing policy. If the policy cache is enabled, calling `Policy.grant()` or `Policy.revoke()` causes an `UnsupportedOperationException.`
- Disable the realm cache when managing realms. This includes adding realms, dropping realms, granting roles, and revoking roles.
- Disable the session cache when you disable HTTP session cookies.

The following is a sample excerpt from `jazn.xml` showing how to disable caching:

```
<jazn provider="LDAP">
   <property name="ldap.user" value="cn=orcladmin"/>
   <property name="ldap.password"
value="{903}3o4PTHbgMzVlzbVfKITIO5Bgio6KK9kD"\/>
   <property name="ldap.cache.session.enable"
value="false" />
   <property name="ldap.cache.realm.enable"
value="false" />
   <property name="ldap.cache.policy.enable"
value="false" />
</jazn>
```

Configuration

The following table describes the cache properties and their default values. You can set these properties only at the virtual machine level, in the `<jazn>` tag in `the` global `jazn.xml` file.

Table 3–9 LDAP Cache Properties

Property	Description	Default
`ldap.cache.policy.enable` (see **Note**)	If set to `true`, enables cache; if set to `false`, disables cache.	`true`
`ldap.cache.realm.enable`	If set to `true`, enables cache; if set to `false`, disables cache.	`true`
`ldap.cache.session.enable`	If set to `true`, enables cache; if set to `false`, disables cache.	`true`
`ldap.cache.initial.capacity`	Initial capacity for the `HashMap`.	`20`
`ldap.cache.load.factor`	Load factor for the `HashMap`.	`.7`

Table 3–9 LDAP Cache Properties

Property	Description	Default
`ldap.cache.purge.initial.delay`	String containing an integer that represents the number of milliseconds the daemon thread waits before starts checking for expired objects.	`3600000`
`ldap.cache.purge.timeout`	The string representation of an integer that represents the number of milliseconds an object remains in cache before being invalidated and removed. It is also the sleep time for the daemon thread between each run looking for expired objects.	`3600000`

Note: `ldap.cache.policy.enable` replaces the deprecated property `ldap.cache.enable`.

A `jazn.xml` file with all caches enabled, a cache size of 100, and a 10000-millisecond timeout would look like:

```
< ?xml version="1.0" encoding="UTF-8" standalone='yes'?>
< !DOCTYPE jazn PUBLIC "JAZN Config"
     "http://xmlns.oracle.com/ias/dtds/jazn.dtd">
< jazn provider="LDAP" location="ldap://example.com:389" >
   < property name="ldap.cache.initial capacity" value="100" />
   < property name="ldap.cache.purget.timeout" value="10000" />
</jazn>
```

Specifying a UserManager In orion-application.xml

Every application, including the top-level default application, has an associated `UserManager`. The `UserManager`'s primary function is to authenticate users who attempt to access web pages and EJBs. The `UserManager` is used to authenticate users connections are made to the application. These are specified using subelements within an `<orion-application>` element that define the configuration. There are three tags that can be used to specify a `UserManager`. They are:.

Table 3–10 UserManager Tags

Tag	Meaning
`<user-manager>`	A user manager implemented by some user-defined class
`<jazn>`	A JAAS user manager.
`<principals>`	A user manager defined in a `principals.xml` file. See "Using the <principals> element and principals.xml" on page 3-24

There may be more than one of these three elements within a single `<orion-application>` element; Which element determines the `UserManager` is determined by the order the elements appear in the table: `<user-manager>` takes precedence over `<jazn>`, which takes precedence over `<principals>`. For example, if both a `<jazn>` and a `<principals>` element are present, the `UserManager` is based on the `<jazn>` element. If multiple elements with the highest-priority tag are present, then the `UserManagers` are chained together as parents. That is, the `UserManager` specified in the first tag becomes the parent of the `UserManager` specified in the second, and so on. The last `UserManager` specified then becomes the `UserManager` of the application. The parent of the first `UserManager` is the `UserManager` associated with the parent application (if any) of the application. The default application does not have a parent application and the parent of its `UserManager` is null.

If no user manager is specified, then the `UserManager` is determined according to the following rules.

- For the default application, a JAAS `UserManager` is created based on `jazn-data.xml` in the directory containing `application.xml`. If no `jazn-data.xml` is present in that directory, one is created. The default realm of the created `jazn-data.xml` is `jazn.com`.
- At deployment time, if the `UserManager` of the parent application is the JAAS `UserManager`, then a JAAS UserManager is created based on `jazn-data.xml`. If necessary, a `jazn-data.xml` file is created in the same way as the previous bullet. A `<jazn>` element is written into the `orion-application.xml` associated with the application.
- At deployment time, if the `UserManager` of the parent application is based on `principals.xml`, then the `UserManager` of the application will be a

principals `UserManager`. If a `principals.xml` file is not present, then an empty file is created. A `<jazn>` element is written into the `orion-application.xml` associated with the application.

- If the `UserManager` of the parent application is user-written, then the parent's `UserManager` will become the `UserManager` of the application.

Using the <principals> element and principals.xml

The `<principals>` element tells OC4J to use the `UserManager` described in a principals file, normally `principals.xml`. A `<principals>` element has one attribute, `<path>`, which specifies a path for the principals file, normally `principals.xml`.

For example,

```
<principals path="myprincipals.xml" />
```

A `principals.xml` file also contains a `<principals>` element; this contains two sub-elements, `<groups>` and `<users>`. The `<groups>` element contains one or more `<group>` elements, and the `<users>` element contains one or more `<user>` elements.

Note: The `XMLUserManager` class is supported for backward compatibility only. Oracle recommends that you use one of the JAAS provider types.

Table 3–11 Elements in principals.xml

Element	Can Contain	Attributes	Description
`<principals>`	`<groups>`, `<users>`	`NA`	Containing element in file
`<groups>`	`<group>`		A list of groups known to this user manager
`<group>`	`<description>`, `<permission>`	`name`	Identifies a single user group; name attribute specifies group name
`<description>`			Not used by JAAS provider, but is displayed in various circumstances.

Table 3–11 Elements in principals.xml

Element	Can Contain	Attributes	Description
`<permission>`		`name`	A `java.security.Permission` that is granted to principals. There are two special values: ■ `administrator`—equivalent to `com.evermind.security.AdministrationPermission()` ■ `rmi:login`— equivalent to `com.evermind.server.rm.RMIPermission("login")`
`<users>`	`<user>`		List of users known to the `UserManager`
`<user>`	`<description>, <group-membership>`		Single user belonging to this group
		`username`	String used to identify the user
		`password`	Cleartext password used to authenticate the user. There is no mechanism for obfuscating this password.
		`deactivated`	Either `true` or `false`. If `true`, then this user will not be found in lookups and will not be able to be authenticated
`<description>`			Arbitrary content that may be displayed in various circumstances
`<group-membership>`		`group`	Name attribute of a <group> which contains this user

Groups in `principals.xml` correspond to roles in the JAAS Provider. The `principals.xml` file does not support any equivalent of the JAAS provider's concept of realms. Permissions granted to groups may be checked explicitly, and OC4J does check for the special permissions listed above. However, group permissions are not integrated with the usual `Permission` checking performed by a `SecurityManager`.

Here is an example `principals.xml` file.

```
<?xml version="1.0" standalone='yes'?>
<!DOCTYPE principals PUBLIC "//Evermind - Orion Principals//"
"http://xmlns.oracle.com/ias/dtds/principals.dtd">
```

```
<principals>
  <groups>
<group name="guests">
    <description>users</description>
  </group>
  <group name="administrators">
    <description>administrators</description>
    <permission name="administration" />
  </group>
  </groups>
  <users>
  <user username="SCOTT" password="TIGER">
  <group-membership group="guests" />
  </user>
  <user username="anonymous" password="">
    <description>The default guest/anonymous user</description>
    <group-membership group="guests" />
  </user>
  <user username="admin" password=""  deactivated="true">
    <description>The default administrator</description>
    <group-membership group="users" />
    <group-membership group="administrators" />
  </user>
  </users>
</principals>
```

4 JAAS Provider Administration Tasks

This chapter describes how to manage the Oracle Application Server Containers for J2EE (OC4J) JAAS Provider.

This chapter contains these topics:

JAAS Provider Management Overview

Managing the JAAS Provider in the J2SE and J2EE environments involves creating and managing realms, users, roles, permissions, and policy. OC4J provides two tools for managing JAAS configuration: the JAAS Provider Admintool and Oracle Enterprise Manager (OEM). You can also manage JAAS programatically.

- Oracle Enterprise Manager
- Oracle Internet Directory Delegated Administrative Service (OID DAS)
- JAZN Admintool, a command-line tool
- JAAS Provider APIs

Note: Configuration tasks differ by whether your application uses an XML-based or LDAP-based JAAS Provider.

Table 4–1 describes which tools can be used in the XML and LDAP provider environments.

Table 4–1 Tools For Managing XML-Based and LDAP-Based Provider Environments

Using This Tool	With XML-Based provider type	With LDAP-Based provider type
Oracle Enterprise Manager	Map security roles.	Map security roles.
OID DAS	Not applicable.	Manage users and groups.
JAZN Admintool	Manage users, roles, and policy.	Manage policy.

Realm and Policy Management

The JAAS Provider supports two types of repository providers, referred to as provider types:

- The XML-based provider type used with an XML file, typically `jazn-data.xml`
- The LDAP-based provider type used with Oracle Internet Directory (OID)

Note: To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide. for details.

OID and `jazn-data.xml` are repositories used to store realm (users and roles) and policy (permissions) information. This section discusses the following topics in relation to the two different provider types:

- Realm and Policy Management Tools
- JAAS Provider Realm Framework
- Realm Management in XML-Based Environments
- Realm Management in LDAP-Based Environments
- JAAS Provider Policy Administration

Realm and Policy Management Tools

Several tools are provided for managing realm and policy information. Table 4–2 describes these tools and indicates the environment in which they operate.

Table 4–2 Realm and Policy Management Tools

Method/Environment	Description	See Also
Oracle Enterprise Manager LDAP-based only	A graphical user interface tool that enables you to manage users, roles, and groups.	"JAAS and Enterprise Manager" on page 8-1
JAZN Admintool Both LDAP and XML-based environments	A command line interface tool that enables administrators to create and manage users, realms, roles, and policies. The JAZN Admintool: ■ Uses the JAAS Provider API to perform functions ■ Can be executed from the operating system command line The JAZN Admintool has the same capabilities and limitations as the JAAS Provider APIs. For example, you cannot create users with the JAZN Admintool if your provider type is LDAP-based Oracle Internet Directory. However, you can create users if your provider type is XML-based.	"Using the JAZN Admintool" on page 5-1

See Also:

The Oracle Application Server 10g Installation Guide for information on installing the provider type you want to use.

JAAS Provider Realm Framework

The J2EE environment defines the concept of user communities. A user community instance is essentially a realm maintained internally by the authorization system.

The API package `oracle.security.jazn.realm` is provided to support realms. This API package is an enhancement to the JAAS policy provider.

Realms can be managed in both provider type environments:

- XML-based

  Provides a lightweight form of storage for realms and JAAS policy

- LDAP-based Oracle Internet Directory

  Provides centralized storage of realms and JAAS policy in a directory

Realm Management in XML-Based Environments

A realm provides user and role management. The XML-based provider offers a lightweight, less restrictive, and faster implementation of realms than does the LDAP-based provider.

XML-Based Realms

You can use the JAAS Provider to create one or more realms for an XML-based environment.

> **See Also:** "Using the JAZN Admintool" on page 4-1 for instructions on creating realms

XML-Based Realm and Policy Information Storage

The XML-based Provider enables you to:

- Create realms, users, and roles
- Grant roles to users and to other roles
- Assign permissions to specific users and roles (principals)

This information is stored in an XML file, typically, `jazn-data.xml`. The following example shows the structure used in a `jazn-data.xml` file to create realms, users, and roles.

```
<!--JAZN Realm Data -->
```

```
<jazn-realm>
     <realm>
          <name>jazn.com</name>
          <users>
               <user>
                    <name>admin</name>
                    <displayName>Realm Administrator</displayName>
                    <description>Administrator for this realm</description>
              <credentials>
                {903}ZcOsWfcw5YRI0Bsq4sNFuLioZgX3a6CF
              </credentials>
               </user>
               <user>
                    <name>anonymous</name>
                    <description>The default guest/anonymous
                             user</description>
               </user>
          </users>
          <roles>
               <role>
                    <name>guests</name>
                    <members>
                         <member>
                              <type>user</type>
                              <name>admin</name>
                         </member>
                         <member>
                              <type>user</type>
                              <name>anonymous</name>
                         </member>
                    </members>
               </role>
               <role>
                    <name>administrators</name>
                    <displayName>Realm Admin Role</displayName>
                    <description>Administrative role for this
                                 realm</description>
                    <members>
                         <member>
                              <type>user</type>
                              <name>admin</name>
                         </member>
                    </members>
               </role>
               <role>
```

```
                        <name>users</name>
                        <members>
                            <member>
                                <type>user</type>
                                <name>admin</name>
                            </member>
                        </members>
                    </role>
                </roles>
            </realm>
        </jazn-realm>
```

See Also: "Sample jazn-data.xml Code" on page A-2 for a completed `jazn-data.xml` file

---

Note: Setting the `<credentials>` element as follows enables you to use clear (readable) passwords in the `jazn-data.xml` file.

- `<credentials clear="true">welcome</credentials>`
- `<credentials>!welcome</credentials>`

This enables the administrator to directly edit `jazn-data.xml` with a text editor. When the file is read and persistence occurs, the password in `jazn-data.xml` is obfuscated and becomes unreadable.

Realm Management in LDAP-Based Environments

A realm provides user and role management. You can manage the data in an LDAP-based realm:

- Internally by creating and managing user information with the JAAS Provider. See Chapter 4, "JAAS Provider Administration Tasks".
- Externally by creating and managing user and role information with Oracle Internet Directory and then integrating it with the JAAS Provider.

LDAP-Based Realm Types

The JAAS Provider supports three types of realms for LDAP-based environments. Each realm provides different user and role management capabilities. Table 4–3 describes these realms.

Table 4–3 Implementation of Realm Types

Realms Type	Description	Use This Realm	See Also
External Realm	▪ Supports external, read-only user and role management ▪ Integrates existing user communities with the JAAS Provider	For non-hosting environments	Figure 4–1 on page 4-9
Identity Management Realm	▪ Created through provisioning tools ▪ Used in hosting environments ▪ Supports external, read-only user and role management	In a hosting environment in which multiple customers or companies subscribe to shared services	Figure 4–2 on page 4-10
Application Realm	▪ Supports external, read-only user management ▪ Supports internal roles management	If you want to use the JAAS Provider role management feature	Figure 4–3 on page 4-11

When you use the LDAP-based JAAS provider with OracleAS Single Sign-On, you must use the Identity Management Realm. The External Realm and Application Realm are not supported when using OracleAS Single Sign-On.

Each realm type consists of:

- A role manager for role management
- A user manager for user management

User and role managers perform their duties internally (through JAAS permissions) or externally (through OID Delegated Administration Service (DAS)).

Note: The JAAS Provider does not supply an internal user manager for creating users. You can create users with DAS or a command line tool such as `ldapadd`.

Figure 4–1 shows a sample LDAP DIT containing an External Realm that is registered as an instance with the JAAS Provider. The realm type is created below a Realms container.

Figure 4–1 Simplified Directory Information Tree for the External Realm

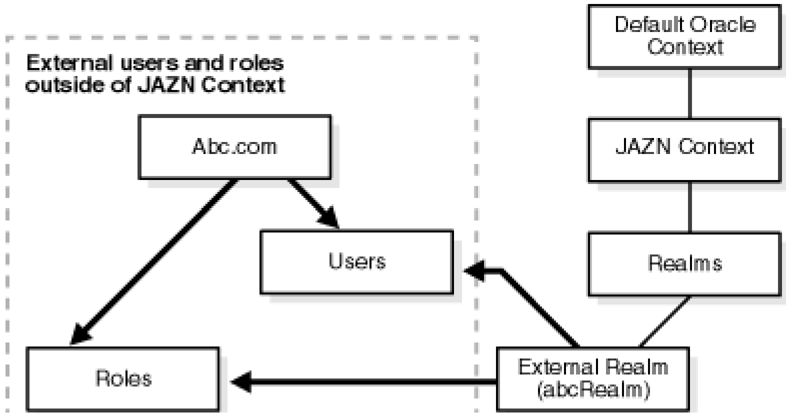

External users and roles outside of JAZN Context
Abc.com
Users
Roles
Default Oracle Context
JAZN Context
Realms
External Realm (abcRealm)

Table 4–4 describes the user and role management responsibilities of the External Realm.

Table 4–4 External Realm Responsibilities

External Realm Name	Role Management	User Management
`abcRealm`	Retrieves external, read-only roles	Retrieves external, read-only users

Figure 4–2 shows a sample LDAP DIT containing an Identity Management Realm that is registered as an instance with the JAAS Provider. The realm type is created below a Realms container.

Figure 4–2 Simplified Directory Information Tree for the Identity Management Realm

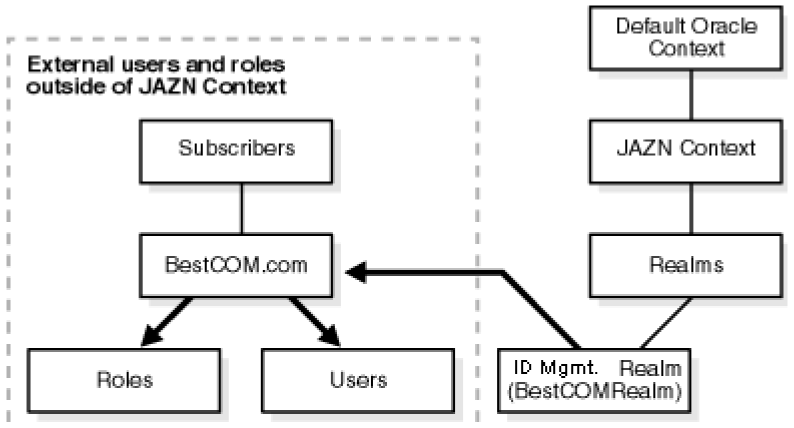

External users and roles outside of JAZN Context
Subscribers
BestCOM.com
Roles
Users
Default Oracle Context
JAZN Context
Realms
ID Mgmt. Realm (BestCOMRealm)

Table 4–5 describes the user and role management responsibilities of the Identity Management Realm.

Table 4–5 Identity Management Realm Responsibilities

Identity Management Realm Name	Role Management	User Management
`BestCOMRealm`	Retrieves external, read-only roles of a subscriber	Retrieves external, read-only users

Figure 4–3 shows a sample LDAP directory information tree (DIT) containing an Application Realm that is registered as an instance with the JAAS Provider. The realm type is created below a Realms container.

Figure 4–3 Simplified Directory Information Tree for the Application Realm

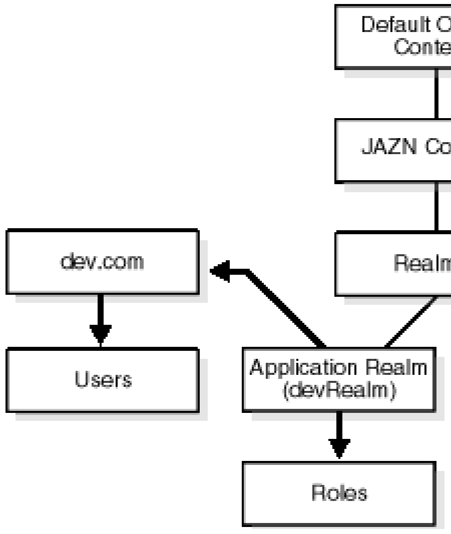


Table 4–6 describes the user and role management responsibilities of the Application Realm.

Table 4–6 Application Realm Responsibilities

Application Realm Name	Role Management	User Management
`devRealm`	Internally creates and manages modifiable roles	Retrieves external, read-only users

LDAP-Based Realm Data Storage

The realm framework provides a means for registering realm instances with the JAAS Provider and managing their information.

A Realms container object is created under the site-wide JAAS context. (For example, see the Realms container in Figure 4–1 on page 4-9.) For each registered realm instance, a corresponding realm entry is created under the Realms container that stores the realm's attributes. This directory hierarchy is known to the JAAS

Provider, which enables the JAAS Provider to create new realm instances in the desirable directory location and find all the registered realms in runtime.

For example, the distinguished name (DN) for a realm called `oracle` can be `"cn=oracle,cn=realms,cn=JAZNContext,cn=site root"`.

Upon successful installation of the JAAS Provider, a default realm instance is installed. Predefined realm properties are configured for starting the default realm. Any realm type must provide concrete implementations for the system-defined Java interfaces `UserManager` and `RoleManager`. During runtime, the JAAS Provider finds all the registered realms and their attributes (name, user manager implementation class, role manager implementation class, and their properties) from the provider type (Oracle Internet Directory) and instantiates the realm's implementation class with the properties for initialization.

Realm Hierarchy As Figure 4–4 illustrates, the JAAS Provider stores its entries within the product container `cn=JAZNContext`. Beneath `cn=JAZNContext` is a `cn=Realms` container, which stores realm entries, and a `cn=Policy` container, which stores global JAAS Provider policies. The `cn=Policy` container in turn stores two types of entries, `cn=Permissions` and `cn=Grantees`.

Note that the JAAS Provider has its own `Groups` and `Users` containers. The `Groups` container contains the group `JAZNAdminGroup`. The `Users` container contains the users that populate these groups.

Figure 4–4 Global JAZNContext Subtree

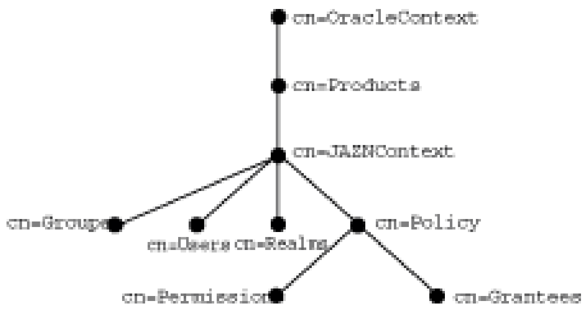
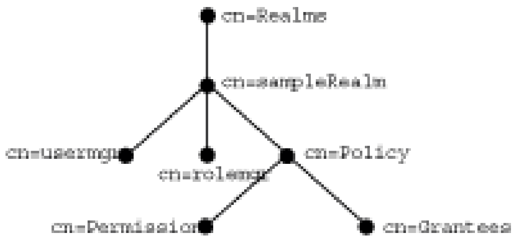


Figure 4–5 shows the directory entries that are placed under the example realm `cn=sampleRealm`. The entry `cn=usermgr` stores information related to user management while the entry `cn=rolemgr` stores information related to role

(group) management. The policy-related entries under `cn=sampleRealm` store realm-specific policies.

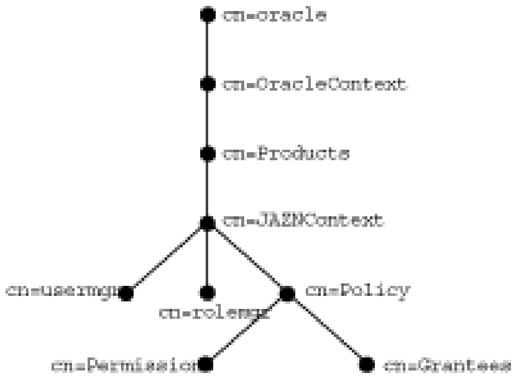
Figure 4–5 A Realm-Specific Subtree



In an identity management-based environment, a subscriber is registered as a realm. Using the subscriber DN, the JAAS Provider locates the subscriber-specific Oracle Context and creates a `cn=JAZNContext` subtree. In this case, the JAAS Provider stores the entries `cn=usermgr` and `cn=rolemgr` and policy-related entries under the subscriber's `JAZNContext`.

In Figure 4–6 `cn=oracle` is a subscriber.

Figure 4–6 Subscriber JAZNContext Subtree



ACLs and JAZN directory entries JAAS Provider directory entries are protected by ACLs at the root of the product subtree. These ACLs grant the group `JAZNAdminGroup` and the JAAS Provider superuser `JAZNAdminUser` full privileges (read, write) for JAAS Provider directory objects. Non-superusers who are not `JAZNAdminGroup` members are denied access to JAAS Provider entries.

Because identity management `JAZNContext` subtrees are mirror images of their site-wide parents, the security measures that they use to protect entries are the same.

LDAP-Based Realm Permissions

A `RealmPermission` class is defined to represent realm permissions. `RealmPermission` extends from `java.security.Permission`. It is used like any regular Java permission. `RealmPermission` has the following characteristics:

- Realm name, also known as target name
- List of actions (permissions applicable to the realm, such as creating a realm, dropping a role, and so on)

See Also:

- The JAAS Provider Javadoc

JAAS Provider Policy Administration

The JAAS Provider implementation of `javax.security.auth.Policy` uses either an LDAP-based Oracle Internet Directory or XML-based provider type for storing policy (authorization rules). The JAAS Provider administrator uses various grant and revoke methods of the `JAZNPolicy` class to create authorization policies for principals.

The policy provider must be administered in a secure manner. There are several ways to administer the JAAS Provider policy:

- Oracle Enterprise Manager (LDAP environments only)
- JAAS Provider Admintool
- Oracle Internet Directory Administration
- AdminPermission Class

See Also: Table 4–2 on page 4-3 for information on Oracle Enterprise Manager and "Using the JAZN Admintool" on page 5-1 for information on the JAZN Admintool

Oracle Internet Directory Administration

For LDAP-based application environments, you manage realm and policy data as Oracle Internet Directory entries through:

- The OID `DAS` and `Oidadmin` administrative tools
- Definition of access control lists in Oracle Internet Directory

Two possible administrative groups can manage the data:

- A JAAS Provider site-wide administrative group that is granted permissions to access and modify the site-wide `JAZNContext` and any identity management-specific `JAZNContext`
- A realm-specific administrative group for each realm instance or administrative user

In hosted application environments, part of the policy data may be partitioned along subscriber boundaries and stored in a subscriber subtree. That policy data cannot be administered by the realm-specific administrative group. The same is true with role information.

With the JAAS Provider policy data (including realm data), only users that belong to `JAZNAdminGroup` have read-access capabilities on provider data.

The LDAP-based environment caches provider policy data; for details, see "Configuring Caching (LDAP-Based Provider Only)" on page 3-18.

See Also: *Oracle Internet Directory Administrator's Guide.*

AdminPermission Class

The `AdminPermission` class can be used in either LDAP-based or XML-based environments.

The `AdminPermission` class represents the right to administer a permission. This enables a grantee (such as a user named `frank`) to further grant and revoke the granted right/permission to other grantees. Instances of this permission class include instances of other permissions. Because this is a permission about permission, it varies slightly from the permission definition, which includes a simple name, actions pair. This variation is resolved by encoding a permission instance as a string and using that as the name of the `AdminPermission` instance. Table 4–7 provides an example:

Table 4–7 ADMIN Permission Example

If User	Then User
`frank` is granted the `AdminPermission` for `java.io.FilePermission("/tmp/*","read,write")`	`frank` can further grant and revoke any permission implied by the embedded permission (that is, `FilePermission` in this instance).

```
 grant Principal com.oracle.security.jazn.JAZNPrincipal "frank"
{
  permission com.oracle.security.jazn.policy.AdminPermission
     "class=java.io.FilePermission, name=\"/tmp/*\", actions=\"read, write\""
};
```

The JAAS Provider does not support recursive embedding of `AdminPermission` (that is, an `AdminPermission` instance embedded within another `AdminPermission` instance). In the initial policy, the user is granted `AdminPermission` to `java.security.AllPermission`, enabling the JAAS Provider user to grant and revoke all permissions to anyone.

A `RoleAdminPermission` class is defined for roles. This means that when role `hr` is granted to `frank`, `frank` is granted both role `hr` and a `RoleAdminPermission` that enables `frank` to further grant and revoke role `hr`.

Policy Partitioning

The JAAS Provider supports policy partitioning among realms (that is, each realm has its own realm-specific policy). This realm-specific policy is administered by the realm-specific administrative group.

Each subscriber is represented by a realm and the subscriber-specific information subtree is stored under a subscriber-specific `JAZNContext`. This subscriber-specific subtree, however, is primarily administered by the JAAS Provider administrative group from the perspective of the LDAP server (Oracle Internet Directory).

JAAS Provider Debug Logging

To turn on JAAS provider debug logging, set the system property `jazn.debug.log.enable` to `true` during Java Virtual Machine (JVM) startup. You do this by modifying the JVM settings for your OC4J instance. In Oracle

Application Server, you normally manage JVM settings via the `<java-options>` element in `opmn.xml`. In standalone mode, you set this property using JVM command-line options. For instance, you might start OC4J standalone with a command line such as:

```
java -Djazn.debug.log.enable=true -jar oc4j.jar
```

Or you can start the Admintool shell in debug mode with the command:

```
java -Djazn.debug.log.enable=true -jar jazn.jar -shell
```

In Oracle Application Server, the debug output is captured by OPMN and written to the log files associated with the OC4J instance.

5 Using the JAZN Admintool

The JAZN Admintool can manage both XML-based and LDAP-based JAAS configurations and data from the command prompt.

Note: The JAZN Admintool manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the *Oracle Internet Directory Administrator's Guide* for details.

The JAZN Admintool is a flexible Java console application, with functions that can be called directly from the command line or through an interactive shell. The shell uses UNIX-derived commands to perform specific JAAS functions. The JAZN Admintool is located in `OC4J_HOME/j2ee/home/jazn.jar.`

This chapter discusses how to perform common administration tasks using the JAZN Admintool. It is divided into the following sections:

- Adding and Removing Users (XML-based Provider Only)
- Checking Passwords (XML-based Provider Only)
- Configuration Operations
- Granting and Revoking Permissions
- Granting and Revoking Roles
- Listing Login Modules
- Listing Permissions
- Listing Permission Information
- Listing Principal Classes
- Listing Realms
- Listing Roles
- Listing Users
- Migrating Principals from the principals.xml File (XML-based Provider Only)
- Setting Passwords (XML-based Provider only)
- Using the JAZN Admintool Shell

Before You Start

When you use the Admintool to manage XML provider data, by default it edits the file `jazn-data.xml` under the `config` directory of the OC4J home instance. The pathname of `jazn-data.xml` is specified in the `<jazn provider="xml" location="`*`pathname`*`">` element in `jazn.xml`. The password for the admin user is set during installation time to the same value as the Oracle Application Server administrator (`ias_admin`) password.

For using the Admintool with the LDAP-based provider, be sure to:

1. Set the correct environment settings as described in "LDAP-Based Provider Environment Settings" on page 3-2.
2. Disable the cache as described in "Disabling Caching" on page 3-20.

Authentication and the JAZN Admintool (XML-based Provider Only)

If you are using the XML-based Provider, you must authenticate yourself to the JAZN Admintool before making administrative changes. You authenticate yourself in one of two ways:

- Supplying the `-user` and `-password` switches, as in:

  ```
  java -jar jazn.jar -user myusername -password mypassword -listrealms
  ```

  Note: If you are using the `-user`, `-password`, or `-clustersupport` options, you must specify them before all other options on the command line.

- Supplying a username and password when prompted by the Admintool, as in:

```
java -jar jazn.jar -listrealms
>RealmLoginModule username: martha
>RealmLoginModule password: mypass
```

In either case you may specify a `LoginModule` for the Admintool in `jazn-data.xml`. If it is not provided, the `RealmLoginModule` is used by default.

Note: The Admintool does not require authentication when used with the LDAP-based provider; if you specify the `-user` and `-password` options when using LDAP, they are ignored.

Specifying an Admintool LoginModule in jazn-data.xml

Your `jazn-data.xml` file can specify which `LoginModule` the Admintool uses to authenticate its users. For example:

```
<application>
  <name>oracle.security.jazn.tools.Admintool</name>
  <login-modules>
    <login-module>
     <class>oracle.security.jazn.realm.RealmLoginModule</class>
      <control-flag>required</control-flag>
      <options>
          <option>
            <name>debug</name>
            <value>false</value>
          </option>
```

```
          <option>
             <name>addAllRoles</name>
             <value>true</value>
          </option>
      </options>
    </login-module>
  </login-modules>
</application>
```

If you try to run the Admintool without specifying a `LoginModule`, the RealmLoginModule with the default options is used.

JAZN Admintool Command-Line Options

The JAZN Admintool provides the following command options, described in greater detail in the following sections. The tool prints error messages if the syntax or parameters are incorrect. You can list all the options and their syntax with the `-help` option, as in:

```
java -jar jazn.jar -help
```

Syntax

The overall syntax for the Admintool is

```
java -jar jazn.jar [-user username -password mypassword
  -clustersupport ORACLE_HOME] [otheroptions]
```

Note: If you are using the `-user`, `-password`, or `-clustersupport` options, you must specify them before all other options on the command line.

This section lists all the Admintool command options.

Admintool Authentication (XML-based Provider Only)

`-user` *`username`* `-password` *`mypassword`*
See "Authentication and the JAZN Admintool (XML-based Provider Only)" on page 5-3.

Clustering Operations

```
-clustersupport oracle_home
```

See "Adding Clustering Support (XML-based Provider Only)" on page 5-7.

Configuration Operations

```
-getconfig
```

See "Configuration Operations" on page 5-12.

Interactive Shell

```
-shell
```

See "Using the JAZN Admintool Shell" on page 5-19.

Login Modules

```
-addloginmodule application_name login_module_name
   control_flag [options]
-listloginmodules [application_name]
-remloginmodule application_name login_module_name
```

See "Adding and Removing Login Modules" on page 5-7 and "Listing Login Modules" on page 5-14.

Migration Operations

```
-convert filename realm
```

See "Migrating Principals from the principals.xml File (XML-based Provider Only)" on page 5-17.

Miscellaneous

```
-help  [<command name>]
```

To display help for a specific command.

Password Management (XML-based Provider only)

```
-checkpasswd realm user [-pw password]
-setpasswd realm user old_pwd new_pwd
```

See "Checking Passwords (XML-based Provider Only)" on page 5-12 and "Setting Passwords (XML-based Provider only)" on page 5-18.

Policy Operations

```
-addperm permission permission_class action target [description]
-addprncpl principlename principle_class parameters [description]
-grantperm {<realm> {-user user|-role <role>} | <principal_class>
         <principal_params>} <permission_class> [<permission_params>] |
-listperms [<realm> {-user <user> |-role <role>} |
         <principal_class> <principal_params> | <permission_name>] |
-listperm permission
-listprncpls
-listprncpl principal_name
-remperm permission
-remprncpl principal_name
-revokeperm {<realm> {-user user|-role <role>} | <principal_class>
         <principal_params>} <permission_class> [<permission_params>] |
```

See "Adding and Removing Policy Permissions (XML-based Provider Only)" on page 5-8, "Adding and Removing Principals (XML-based Provider Only)" on page 5-9, "Granting and Revoking Permissions" on page 5-12, "Listing Permissions" on page 5-14, "Listing Permission Information" on page 5-14, "Listing Principal Classes" on page 5-15, and"Listing Principal Class Information".

Realm Operations

```
-addrealm realm admin {adminpwd adminrole | adminrole
  userbase rolebase realmtype}
-addrole realm role
-adduser realm username password
-grantrole role realm {user|-role to_role}
-listrealms
-listroles [realm [user|-role role]]
-listusers [realm [-role role|-perm permission]]
-remrealm realm
-remrole realm role
-remuser realm user
-revokerole role realm {user|-role from_role}
```

See "Adding and Removing Realms" on page 5-10, "Adding and Removing Roles" on page 5-10, "Adding and Removing Users (XML-based Provider Only)" on page 5-11, "Granting and Revoking Roles" on page 5-13, "Listing Realms" on page 5-16, "Listing Roles" on page 5-16, and "Listing Users" on page 5-17.

Adding Clustering Support (XML-based Provider Only)

-clustersupport *oracle_home*

This option instructs the Admintool to propagate all JAAS configuration changes throughout a cluster. The *oracle_home* argument specifies the absolute pathname of the Oracle home directory. You can combine `-clustersupport` with the `-shell` option.

Notes: If you are using the `-clustersupport` option, you must specify it before all other options on the command line.

The `-clustersupport` option is meaningful only when using the XML-based provider.

For example:

```
java -jar jazn.jar -clustersupport /oracle_home -shell
```

Adding and Removing Login Modules

```
-addloginmodule application_name login_module_name
  control_flag [optionname=value ...]
-remloginmodule application_name login_module_name
```

The `-addloginmodule` option configures a new `LoginModule` for the named application.

The *control_flag* must be one of `required`, `requisite`, `sufficient` or `optional`, as specified in `javax.security.auth.login.Configuration`. See Table 5–1.

Table 5–1 LoginModule Control Flags

Flag	Meaning
`Required`	The `LoginModule` must succeed. Whether or not it succeeds, authentication proceeds down the `LoginModule` list.

Table 5–1 LoginModule Control Flags

Flag	Meaning
`Requisite`	The `LoginModule` must succeed. If it succeeds, authentication continues down the `LoginModule` list. If it fails, control immediately returns to the application (authentication does not continue down the `LoginModule` list).
`Sufficient`	The `LoginModule` is not required to succeed. If it succeeds, control immediately returns to the application and authentication does not proceed down the `LoginModule` list. If it fails, authentication continues down the `LoginModule` list.
`Optional`	The `LoginModule` is not required to succeed. Whether or not it succeeds, authentication proceeds down the `LoginModule` list.

If the `LoginModule` accepts its own options, you specify each option and its value as an *`optionname=value`* pair. Each `LoginModule` has its own individual set of options.

For instance, to add `MyLoginModule` to the application `myapp` as a required module with `debug` set to `true`, type:

```
java -jar jazn.jar -addloginmodule myapp MyLoginModule required debug=true
```

To delete `MyLoginModule` from `myapp`, type:

```
java -jar jazn.jar -remloginmodule myapp MyLoginModule
```

Admintool shell:

```
JAZN:> addloginmodule myapp MyLoginModule required debug=true
JAZN: remloginmodule myapp MyLoginModule
```

Adding and Removing Policy Permissions (XML-based Provider Only)

-**addperm** *permission permission_class action target [description]*
-**remperm** *permission*

The `-addperm` option registers a permission with the JAAS Provider `PermissionClassManager`. The `-remperm` option removes registration for the specified permission class. To supply multiple words in the `permission` or `description` arguments, enclose them in quotation marks ("*three word permission*").

If you add a permission that already exists, the Admintool updates the permission's action and target lists.

For instance, to create permission to drop a realm, type:

```
java -jar jazn.jar -addperm perm1 oracle.security.jazn.realm.RealmPermission
droprealm "permission to drop a realm"
```

To delete the `droprealm` permission, type:

```
java -jar jazn.jar -remperm perm1
```

Admintool shell:

```
JAZN:> addperm perm1 oracle.security.jazn.realm.RealmPermission droprealm -null
"permission to drop a realm"
JAZN: remperm perm1
```

Adding and Removing Principals (XML-based Provider Only)

-**addprncpl** *principlename principle_class parameters [description]*
-**remprncpl** *principal_name*

The `-addprncpl` option registers a principal with the JAAS Provider `PrincipalClassManager`. The `-remprncpl` option removes registration for the specified principal class. To supply multiple words in the `principal_name` and `description` arguments, enclose them in quotation marks ("*three word description*").

If you add a principal that already exists, the Admintool updates the principal's parameter list.

For example, to add the principal `staff`, type:

```
java -jar jazn.jar -addprincpl staff oracle.security.jazn.spi.xml.XMLRealmUser
  "a staff user"
```

Admintool shell:

```
JAZN:> addprincpl staff oracle.security.jazn.spi.xml.XMLRealmUser -null "a staff
```

```
user"
```

Adding and Removing Realms

-**addrealm** *realm admin {adminpwd adminrole | adminrole userbase rolebase realmtype}*
-remrealm *realm*

The `-addrealm` option creates a realm of the specified type with the specified name, and `-remrealm` deletes a realm.

For example, using the XML-based Provider, the administrator `martha` with password `mypass` using role `hr` would add the realm `employees` as follows:

```
java -jar jazn.jar -addrealm employees martha mypass hr
```

Using the LDAP-based Provider, the administrator `martha` using role `hr` would add the realm `employees` to userbase `ub` and rolebase `rb` in an external realm as follows:

```
java -jar jazn.jar -addrealm employees martha hr ub rb external
```

Note: The *`realmtype`* argument is required only when using the LDAP-based Provider. The possible values for *`realmtype`* are:

- `external`
- `application`

In either environment, the administrator would delete `employees` as follows:

```
java -jar jazn.jar -remrealm employees
```

Adding and Removing Roles

-addrole *realm role*
-remrole *realm role*

The `-addrole` option creates a role in the specified realm; the `-remrole` option deletes a role from the realm.

Note: If you are using the LDAP-based provider, `-addrole` and `-remrole` are supported only for application realms; they are not supported for external or identity management realms.

For example, to add the role `roleFoo` to the realm `foo`, type:

```
java -jar jazn.jar -addrole foo fooRole
```

To delete the role from the realm, type:

```
java -jar jazn.jar -remrole foo fooRole
```

Admintool shell: `JAZN:> remrole foo fooRole`

Adding and Removing Users (XML-based Provider Only)

`-adduser` *`realm username password`*
`-remuser` *`realm user`*

The `-adduser` option adds a user to a specified realm; the `-remuser` option deletes a user from the realm. For example, to add the user `martha` to the realm `foo` with the password `mypass`, type:

```
java -jar jazn.jar -adduser foo martha mypass
```

Notes: ■

- To insert a user with no password, end the command line with the `-null` option, as in:
  `jazn -jar jazn.jar -adduser foo martha -null`
- If you are using the LDAP-based provider, these commands will not work.

To delete `martha` from the realm, type:

```
java -jar jazn.jar -remuser foo martha
```

Admintool shell: `JAZN:> adduser foo martha mypass`

Checking Passwords (XML-based Provider Only)

-**checkpasswd** *realm user* [-pw *password*]

The `-checkpasswd` option indicates whether the given user requires a password for authentication.

When you specify `-checkpasswd` alone, the Admintool responds `"A password exists for this principal"` if the user has a password, or `"No password exists for this principal"` if the user has no password.

When you specify `-checkpasswd` together with the `-pw` option, the Admintool responds `"Successful verification of user/password pair"` if the username and password pair are correct, or `"Unsuccessful verification of user/password pair"` if username and/or password is incorrect.

For example, to check whether the user `martha` in realm `foo` uses the password `Hello`, type:

```
java -jar jazn.jar -checkpasswd foo martha -pw Hello
```

Admintool shell: `JAZN:> checkpasswd foo martha -pw Hello`

Configuration Operations

-**getconfig**

The `-getconfig` option displays the current configuration setting in `jazn.xml`.

For example, to check the configuration settings for the realm `foo`, type:

```
java -jar jazn.jar -getconfig
```

Admintool shell: `JAZN:> getconfig foo`

Granting and Revoking Permissions

-**grantperm** *realm* {*-user user*|-role *role* } | *principal_class principal_parameters}* *permission_class* [*permission_parameters]*
-**revokeperm** *realm* {-user user|-role *role*} | *principal_class principal_parameters}* *permission_class* [*permission_parameters]*
-**listperms** *realm* {-user user|-role *role*} | *principal_class*

principal_parameters} *permission_class* [*permission_parameters]*

where *principal_class* is the fully qualified name of a class that implements the principal interface (e.g., `com.sun.security.auth.NTDomainPrincipal`) and *principal_paramters* is a ***single*** String parameter.

The `-grantperm` option grants the specified permission to a user (when called with `-user`) or a role (when called with `-role)` or a principal. The `-revokeperm` option revokes the specified permission from a user or role or principal

A *permission_descriptor* consists of a permission's explicit class name (for example, `oracle.security.jazn.realm.RealmPermission`), its action, and its action and target parameters (for `RealmPermission`, `realmname action`). Note that there may be multiple action and target parameters.

For example, to grant `FilePermission` with target `a.txt` and actions "`read, write`" to user `martha` in realm `foo`, type:

```
java -jar jazn.jar -grantperm foo martha java.io.FilePermission
 a.txt read, write
```

Admintool shell: `JAZN:> grantperm foo martha java.io.FilePermission a.txt read, write`

Granting and Revoking Roles

-grantrole *role realm* {user|-role *to_role*}
-revokerole *role realm* {user|-role *from_role*}

The `-grantrole` option grants the specified role to a user (when called with a user name) or a role (when called with `-role`). The `-revokerole` option revokes the specified role from a user or role.

Note: If you are using the LDAP-based provider, `-grantrole` and `-revokerole` are supported only for application realms; they are not supported for external or identity management realms.

For example, to grant the role `editor` to the user `martha` in realm `foo`, type:

```
java -jar jazn.jar -grantrole editor foo martha
```

Admintool shell: `JAZN:> grantrole editor foo martha`

Listing Login Modules

-**listloginmodules** [*application_name*]

The `-listloginmodules` option displays all `LoginModules` either in the specified *`application_name`* or, if no *`application_name`* is specified, in all applications.

For example, to display all `LoginModules` for the application `myapp`, type:

```
java -jar jazn.jar -listloginmodules myapp
```

Admintool shell: `JAZN:> listloginmodules myapp`

Listing Permissions

-listperms *realm* {-user *user*|-role *role*} | *principal_class* *principal_parameters*} *permission_class* [*permission_parameters*]

The `-listperms` option displays all permissions that match the list criteria. This option lists the following:

- All permissions registered with the JAAS Provider `PermissionClassManager`
- Permissions that are granted to a role when the `-role` option is used.
- Permissions that are grated to a `prinicpal`.

For example, to display all permissions for the user `martha` in realm `foo`, type:

```
java -jar jazn.jar -listperms foo martha
```

Admintool shell: `JAZN:> listperms foo martha`

Listing Permission Information

-**listperm** *permission*

The`-listperm` option displays detailed information about the specified permission, including the permission's display name, class, description, actions, and targets.

For example, to list all information about the permission `perm1`, type:

```
java -jar jazn.jar -listperm perm1
```

Typical output might look like

```
Name:
perm1

Class:
oracle.security.jazn.realm.RealmPermission

Description:
permission to drop realm

Targets:

Actions:
droprealm  <no description available>
```

Admintool shell: `JAZN:> listperm perm1`

Listing Principal Classes

`-listprncpls`

The `-listprncpls` option lists all principal classes registered with the `PrincipalClassManager`.

For example:

```
java -jar jazn.jar -listprncpls
```

Admintool shell: `JAZN:> listprncpls`

Listing Principal Class Information

`-listprncpl` *`principal_name`*

The `-listprncpl` option displays detailed information about the specified principal, including the display name, class, description, and actions.

For example, to list all information about the principal `martha`, type:

```
java -jar jazn.jar -listprncpl martha
```

In our example, the output would be:

```
Name:
martha
Class:
oracle.security.jazn.spi.xml.XMLRealmUser
Description:
a staff user
Parameters:
```

Admintool shell: `JAZN:> listprncpl martha`

Listing Realms

`-listrealms`

The `-listrealms` option displays all realms in the current JAAS environment.

For example, to list all realms, type:

```
java -jar jazn.jar -listrealms
```

Admintool shell: `JAZN:> listrealms`

Listing Roles

-**listroles** [*realm* [*user*|-role *role*]]

The `-listroles` option displays a list of roles that match the list criteria. This option lists:

- All roles in all realms, when called without any parameters
- All roles granted to a user, when called with a realm name and user name
- Roles that are granted the specified *`role`*, when called with a realm name and the option `-role`

For example, to list all roles in realm `foo`, type:

```
java -jar jazn.jar -listroles foo
```

Admintool shell: `JAZN:> listroles foo`

Listing Users

-**listusers** [*realm* [-role *role*|-perm *permission*]]

The `-listusers` option displays a list of users that match the list criteria. This option lists:

- All users in all realms, when called without any parameters
- All users in a realm, when called with a realm name
- Users that are granted a certain role or permission, when called with a realm name and the option `-role` or `-perm`

For example, to list all users in realm `foo`, type:

```
java -jar jazn.jar -listusers foo
```

For example, to list all users in realm `foo` using permission `bar`, type:

```
java -jar jazn.jar -listusers foo -perm bar
```

The Admintool lists users one per line, as in:

```
scott
admin
anonymous
```

Admintool shell: `JAZN:> listusers foo`

Migrating Principals from the principals.xml File (XML-based Provider Only)

-**convert** *filename realm*

The `-convert` option migrates the `principals.xml` file into the specified realm of the current JAAS Provider. The *`filename`* argument specifies the pathname of the input file (typically *ORACLE_HOME*/`j2ee/home/config/principals.xml`).

Note: In previous releases this option was called `-migrate`. The syntax and behavior of `-convert` are identical to those of `-migrate`.

The migration converts `principals.xml` users to JAAS `users` and `principals.xml` groups to JAAS roles. All permissions that were previously granted to a `principals.xml` group are mapped to the JAAS role. Users that were deactivated at the time of migration are not migrated. This ensures that no users can inadvertently gain access through the migration.

An error (either `Javax.naming.AuthenticationException:Invalid username/password` or `javax.naming.NamingException:Lookup Error`) is returned if the input file contains errors.

Before you convert `principals.xml`, you must make sure that you have an administrator user that is authorized to manage realms. To do this:

1. Activate the administrative user in `principals.xml`, which is deactivated by default. Be sure to create a password for the administrator.

   Make sure that the administrator name you used to create the realm is different from the name of the administrator in `principals.xml`. This is necessary because the convert command does not migrate duplicate users, and migrates duplicate roles by overwriting the old one.

2. Create the realm `principals.com` with a dummy user and a dummy role. For example, in the Admintool shell you would type:

   ```
   JAZN> addrealm principals.com u1 welcome r1
   ```
3. Migrate principals.xml to the principals.com realm, as in

   ```
   java -jar jazn.jar -convert config/principals.xml principals.com
   ```
4. Edit `jazn.xml` and change the `<default-realm>` entry to `principals.com`.

5. Stop OC4J and restart it.

Setting Passwords (XML-based Provider only)

-**`setpasswd`** *`realm user old_pwd new_pwd`*

The `-setpasswd` option allows administrators to reset the password of a user given the old password.

For example, to change the user `martha` in realm `foo` from password `mypass` to password `a2d3vn`, type:

```
java -jar jazn.jar -setpasswd foo martha mypass a2d3vn
```

Admintool shell: `JAZN:> setpasswd foo martha mypass a2d3vn`

Using the JAZN Admintool Shell

`-`**`shell`**

The `-shell` option starts a JAZN Admintool shell. The JAZN Admintool shell provides interactive administration of JAAS principals and policies through a UNIX-derived interface.

```
java -jar jazn.jar -user martha -password mypass -shell
JAZN:>
```

The shell responds with the `JAZN:>` prompt. To leave the interface shell, type `exit`.

> **Note:** Multi-word arguments must be enclosed in quotes. For example, `java -jar jazn.jar -user '`*`Oracle DBA`*`' ...`

If you are using the XML-based provider you must supply a username and password to the Admintool; for details see "Authentication and the JAZN Admintool (XML-based Provider Only)" on page 5-3. If you are using the LDAP-based Provider, you do not need to specify the `-user` and `-password` arguments.

Navigating the JAZN Admintool Shell

The Admintool shell supports UNIX-like commands for navigating within a JAZN structure. For a complete discussion of the Admintool directory structure, see "Admintool Shell Directory Structure" on page 5-22. All the Admintool commands support relative and absolute paths.

The Admintool navigation commands are:

add: Creating Provider Data

`add` *`directory_name`* `[`*`other_parameter`*`]`

mkdir *directory_name* [*other_parameter*]
mk *directory_name* [*other_parameter*]

The `add`, `mkdir`, and `mk` commands are synonyms: they create a subdirectory or node in the current directory. For example, if the current directory is the root, then `mk` creates a realm. If the current directory is `/realm/users`, then `mk` creates a user. The effect of `add` depends upon the current directory. Some commands require additional parameters in addition to the name.

cd: Navigating Provider Data

cd *path*

The `cd` command allows users to navigate the directory tree. Relative and absolute path names are supported. To exit a directory, type:

```
cd ..
```

Typing `cd /` returns the user to the root node. An error message is displayed if the specified directory does not exist.

clear: Clearing the Screen

clear

The `clear` command clears the terminal screen by displaying 80 blank lines.

exit: Exiting the JAZN Shell

exit

The `exit` command exits the JAZN shell.

help: Listing JAZN Admintool Shell Commands

help

The `help` command displays a list of all valid commands.

ls: Listing Data

`ls` [*path*]

The `ls` command lists the contents of the current directory or node. For example, if the current directory is the root, then `ls` lists all realms. If the current directory is

`/realm/users`, then `ls` lists all users in the realm. The results of the listing depends on the current directory. The `ls` command can operate with the `*` wildcard.

man: Viewing JAZN Admintool Man Pages

`man` *`command_option`*
`man` *`shell_command`*

The `man` command displays detailed usage information for the specified shell command or JAZN Admintool command option. Where information presented by the `man` page and this document conflict, this document contains the correct usage for the command.

pwd: Displaying The Working Directory

`pwd`

The `pwd` command displays the current location of the user in the directory tree. Undefined values are left blank in this listing.

rm: Removing Provider Data

`rm` *`directory_name`*

The `rm` command removes the directory or node in the current directory. For example, if the current directory is the root, then `rm` removes the specified realm. If the current directory is `/realm/users`, it removes the specified user. The effect of `rm` depends on the current directory. An error message is displayed if the specified directory does not exist.

The `rm` command accepts the `*` wildcard.

set: Updating Values

`set` *`name=value`*

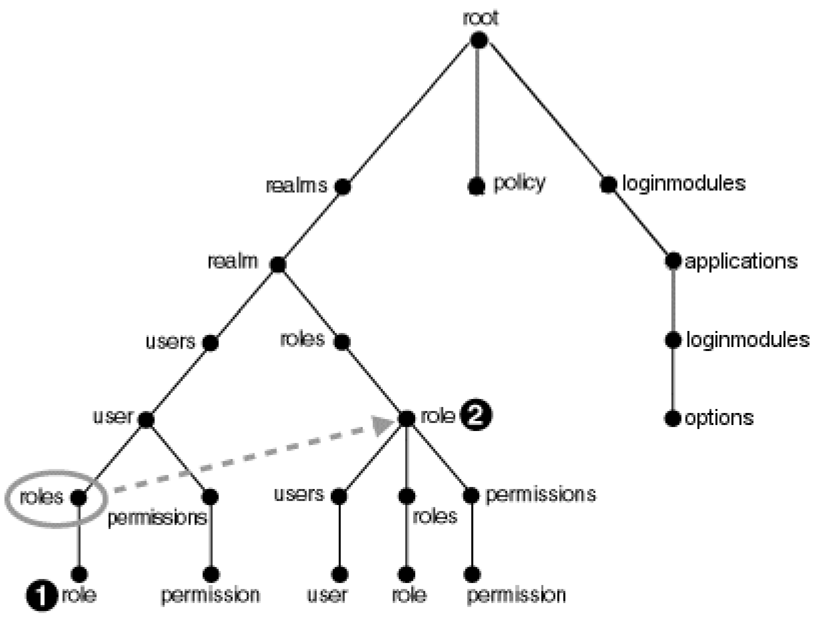
The `set` command updates the value of the specified name. For example, use this command to update the login module class, or a login module control flag, or a login module class option, depending on the working directory.

Admintool Shell Directory Structure

The JAZN Admintool includes a shell called the JAZN shell interface. The JAZN shell is an interactive interface to the JAAS Provider API.

The shell directory structure consists of nodes, where nodes contain subnodes that represent the parent node's properties. Figure 5–1 illustrates the node structure.

Figure 5–1 JAZN Shell Directory Structure

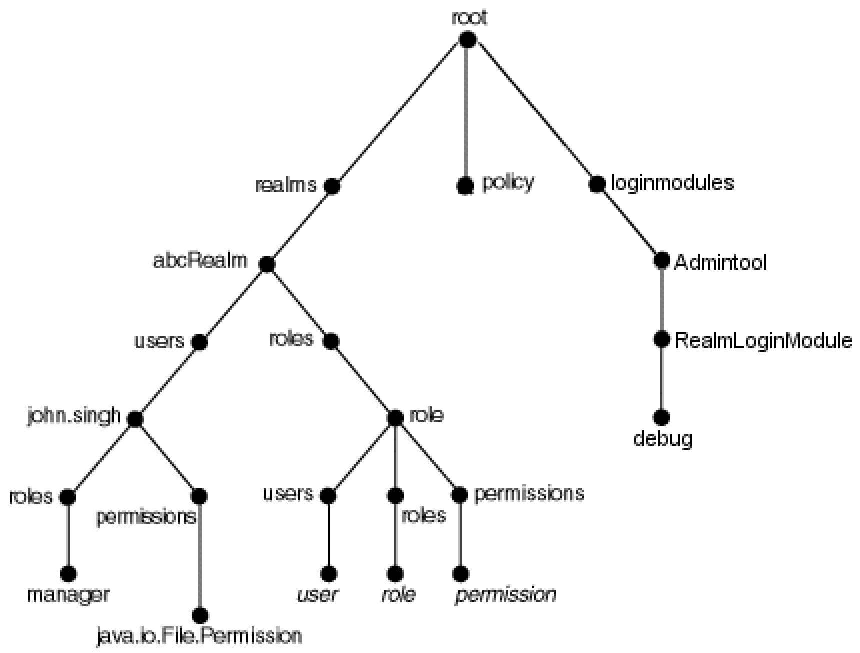


In this structure, the `user` and `role` nodes are linked together. This means that the `roles` link under `user` is the same link as the `roles` link under `realm`. In Unix terms, the `role` at numeral 1 in the diagram is a symbolic link to `role` at numeral 2 in the diagram.

Note: In this release, the policy directory is always empty.

Figure 5–2 shows nodes of the `xmlRealm` created by the `jazn-data.xml` file in "Sample jazn-data.xml Code" on page A-2.

Figure 5–2 Illustrated Shell Directory Structure



6 Security and J2EE Applications

This chapter describes security issues affecting J2EE applications in Oracle Application Server Containers for J2EE (Oracle Application Server Containers for J2EE).

This chapter contains these topics:

Introduction

When the JAAS Provider is integrated with applications developed for the Java 2 Platform, the following Oracle components are available to developers:

- The JAAS Provider, which provides support for storage, retrieval, and administration of realm information (users and roles) and policy information (permissions). The JAAS Provider supports two possible repositories or *provider types*:
  - XML-based Provider Type
  - LDAP-based Oracle Internet Directory (available only with OracleAS Infrastructure installation)
- Login modules, such as the JAAS Provider `RealmLoginModule`

See Also:

- "Provider Types" on page 2-2 for further information about provider types
- Oracle Application Server 10g Security Guide for information on installing Oracle Internet Directory.

Security Considerations During Development and Deployment

The JAAS Provider is designed to work with the J2EE declarative security model. This declarative model requires little or no programming to use JAAS security in your application. Instead, most security decisions are made as part of the deployment process, making it easy to make changes without requiring re-coding. If the declarative model is not sufficient, the JAAS Provider also supports programmatic security in the same manner that JAAS is used in any J2SE environment.

Development

If your application relies on the declarative security model (where J2EE security roles are defined in deployment descriptors, such as `web.xml`), the developer must determine if the application uses application-specific roles. If so, the developer must define these roles so that they can be mapped to the J2EE logical roles during the deployment phase.

If your application uses JAAS programmatically, then the developer must create a JAAS `LoginContext` and explicitly call the `login()` method to invoke a JAAS `LoginModule`.

Deployment

Using the declarative security model, the deployer must make the following security-related decisions:

- Determine the J2EE logical roles that are assumed in the application, then define these roles in the deployment descriptors. For example, an HR application may assume that the J2EE logical role `hr_manager` is running the application; the deployer must define that role.
- Determine the authorization constraints that apply to these roles and define them in the deployment descriptors. For web modules, these constraints typically apply to URL patterns as defined in the J2EE specification. EJB modules typically have constraints at the EJB-method level.
- Decide whether to use an XML flat file or OID (LDAP) as the repository for the JAAS Provider. This also determines which provider, XML-based or LDAP-based, and user manager the application uses.
- Map the security roles (including the application-specific roles, if they exist) to users and groups defined by the OC4J user manager (for instance, `JAZNUserManager`). For example, the J2EE logical role called `hr_manager` may be mapped to a given set of users defined by the OC4J user manager.

For information on making and implementing these decisions, see Chapter 3, "Configuring And Deploying the JAAS Provider"; for a full discussion of deployment, see the *Oracle Application Server Containers for J2EE User's Guide.*

OC4J and the JAAS Provider

Oracle Application Server Containers for J2EE is a J2EE container that accepts HTTP and RMI client connections. These connections permit access to servlets, Java Server Pages (JSPs), and Enterprise JavaBeans (EJBs).

J2EE containers separate business logic from resource and lifecycle management. This enables developers to focus on writing business logic, rather than writing enterprise infrastructure. For example, Java servlets simplify Web development by providing an infrastructure for component, communication, and session management in a Web container integrated with a Web server.

OC4J Integration

The JAAS Provider is integrated with Oracle Application Server Containers for J2EE to enhance application security. This integration provides the following benefits:

- Integration with single sign-on (SSO)
- Fine-grained access control through Java 2 permissions
- `run-as` identity support, delegation support (from servlet to Enterprise JavaBeans)
- Secure file-based storage of user passwords

JAZNUserManager

Another key component of JAAS integration in the J2EE environment is `JAZNUserManager`. `JAZNUserManager` is an implementation of the Oracle Application Server Containers for J2EE `UserManager` interface.

Replacing principals.xml

The OC4J `principals.xml` file is less secure and less flexible than authentication with `JAZNUserManager`. `JAZNUserManager` provides the following:

- Secure storage of obfuscated passwords
- Full role-based access control (RBAC), including hierarchical roles
- Full support for the Java 2 permission model and JAAS
- Secure implementation based on the Java 2 permission model, allowing non-trusted (or partially trusted) code to run in the same JVM as the JAAS Provider

Note: We strongly encourage you to migrate your existing applications from using `principals.xml` to using `JAZNUserManager`. For instructions, see "Migrating Principals from the principals.xml File (XML-based Provider Only)" on page 5-17.

JAZNUserManager Features

In addition to the features mentioned in "Replacing principals.xml" on page 6-4, `JAZNUserManager` provides many other features, including:

- Single Sign-On (SSO) integration with Oracle Application Server Containers for J2EE
- `RealmLoginModule` integration in non-SSO environments
- Support for custom login modules
- Identity propagation
- Location, reading, editing, removal, and management of user and group objects
- Enforcement of security constraints

Authentication Environments

The JAAS Provider integrates with three different login authentication environments in a J2EE application.

- **SSO**

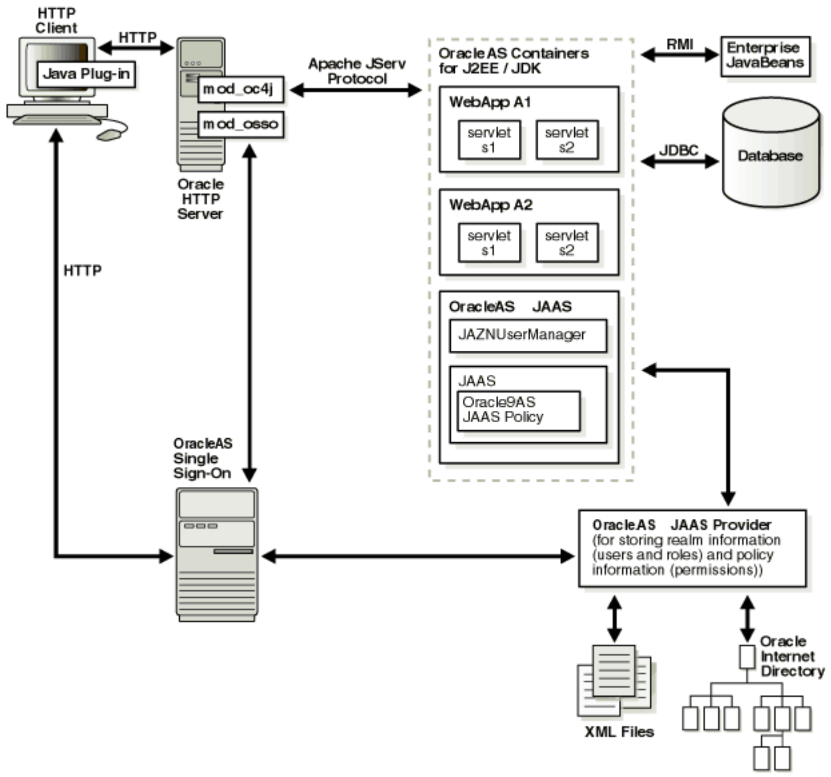
  Uses OracleAS Single Sign-On to authenticate logins
- **SSL**
  - Uses Secure Socket Layers for client certificate-based authentication
  - Uses a login module (for example, `RealmLoginModule`) to authenticate logins
- **Basic Authentication**
  - Prompts user directly for username and password, without going through OracleAS Single Sign-On
  - Uses a login module (for example, `RealmLoginModule`) to authenticate logins

The following sections discuss how the JAAS Provider integrates with each of these authentication types.

Integrating the JAAS Provider with SSO-Enabled Applications

SSO lets a user access multiple accounts and applications with a single set of login credentials. Figure 6–1 shows JAAS integration in an application running in an SSO-enabled J2EE environment.

Figure 6–1 Oracle Component Integration in SSO-Enabled J2EE Environments



SSO-Enabled J2EE Environments: A Typical Scenario

This section describes the responsibilities of Oracle components when an HTTP client request is initiated in an SSO-enabled J2EE environment.

1. An HTTP client attempts to access a Web application, WebApp A1, hosted by Oracle Application Server Containers for J2EE (the Web container for executing servlets). Oracle HTTP Server (using an Apache listener) handles the request.
2. mod_osso/Oracle HTTP Server receives the request and:
   - Determines that WebApp A1 application requires Web-based SSO for authenticating HTTP clients
   - Redirects the HTTP client request to the Web-based SSO OracleAS Single Sign-On (because it has not yet been authenticated).

3. The HTTP client is authenticated by OracleAS Single Sign-On through HTTP or public key infrastructure (PKI) Authentication. OracleAS Single Sign-On then:
   - Validates the user's stored login credentials
   - Sets the SSO cookie (including the user’s distinguished name and realm)
   - Redirects back to the WebApp A1 application (in Oracle Application Server Containers for J2EE)
4. The JAAS Provider retrieves the SSO user.
5. The final step or steps depend on the setting of the `runas-mode` in the `<jazn-web-app>` element.

   If the `runas-mode` is set to `false`, then the following happens:

   a. The target servlet is invoked.

      If the `runas-mode` is set to `true`, then the following happens:

   b. The JAAS Provider invokes the target servlet within a `PrivilegedAction` block through `Subject.doAs()`. The `JAZNUserManager` enforces security constraints.
      – When `Subject.doAs()` is called, JAAS consults the JAAS Provider for permissions associated with the SSO user through the `getPermissions()` method.
      – The JAAS Provider retrieves the permissions associated with the given grantee from the provider type (LDAP-based or XML-based), and updates the policy cache as appropriate. The JAAS Provider then returns the granted set of permissions to the JAAS Provider runtime.
      – The JAAS Provider runtime constructs a new `AccessControlContext` based on the permissions returned from `getPermissions()`.
   c. The servlet's code runs under the `AccessControlContext` of the SSO user.
   d. If the servlet's code attempts to write to a file in the operating system’s file system, then this triggers a call to `SecurityManager.checkPermission()`.
   e. The JVM then:
      – Examines the authorization context associated with the current thread

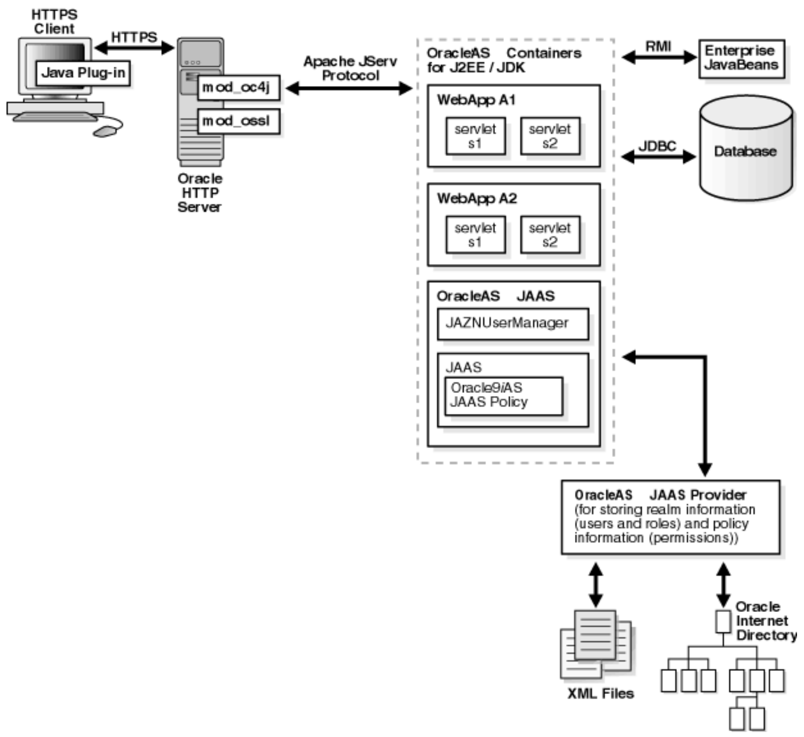
   - Determines that the current subject has the required permissions to write to the file

f. `SecurityManager.checkPermission()` returns safely and the client HTTP request proceeds.

Integrating the JAAS Provider with SSL-Enabled Applications

SSL is an industry standard protocol for managing the security of message transmission on the Internet. Figure 6–2 shows JAAS integration in an application running in an SSL-enabled J2EE environment.

Figure 6–2 Oracle Component Integration In SSL-Enabled J2EE Environments



SSL-Enabled J2EE Environments: A Typical Scenario

This section describes the responsibilities of Oracle components when an HTTP client request is initiated in an SSL-enabled J2EE environment. In this environment, OracleAS Single Sign-On is not used. A login module (for example, `RealmLoginModule`) is used.

1. An HTTP client attempts to access a Web application (named WebApp A1) hosted by Oracle Application Server Containers for J2EE (the Web container for executing servlets). Oracle HTTP Server (using an Apache listener) handles the request.
2. mod_ossl/Oracle HTTP Server receives the request and determines that the WebApp A1 application requires SSL server authentication for HTTP clients.
3. If a server and/or client wallet certificate is configured, the HTTP client is prompted to accept the server certificate of OHS and provide the client certificate.
4. The JAAS Provider retrieves the SSL client certificate.
5. The JAAS Provider retrieves the SSL user from the certificate.
6. The final step or steps depend on the `runas-mode` specified in the `<jazn-web-app>` element.

   If `runas-mode` is set to `false`, then the target servlet is invoked.

   If `runas-mode` is set to `true`, then the following happens:

   a. The JAAS Provider invokes the target servlet within a `PrivilegedAction` block through `Subject.doAs()`. The `JAZNUserManager` enforces security constraints.

      – When `Subject.doAs()` is called, the method consults for permissions associated with the SSL user through the `getPermissions()` method.

      – The JAAS Provider retrieves the permissions associated with the given grantee from the provider type (LDAP-based or XML-based), and updates the policy cache as appropriate. The JAAS Provider then returns the granted set of permissions to the JAAS Provider runtime.

      – The JAAS Provider runtime constructs a new `AccessControlContext` based on the permissions returned from `getPermissions()`.

   b. The servlet's code runs under the `AccessControlContext` of the SSL user.

   c. If the servlet's code attempts to write to a file in the operating system's file system, then this triggers a call to `SecurityManager.checkPermission()`.

d. The JVM then:

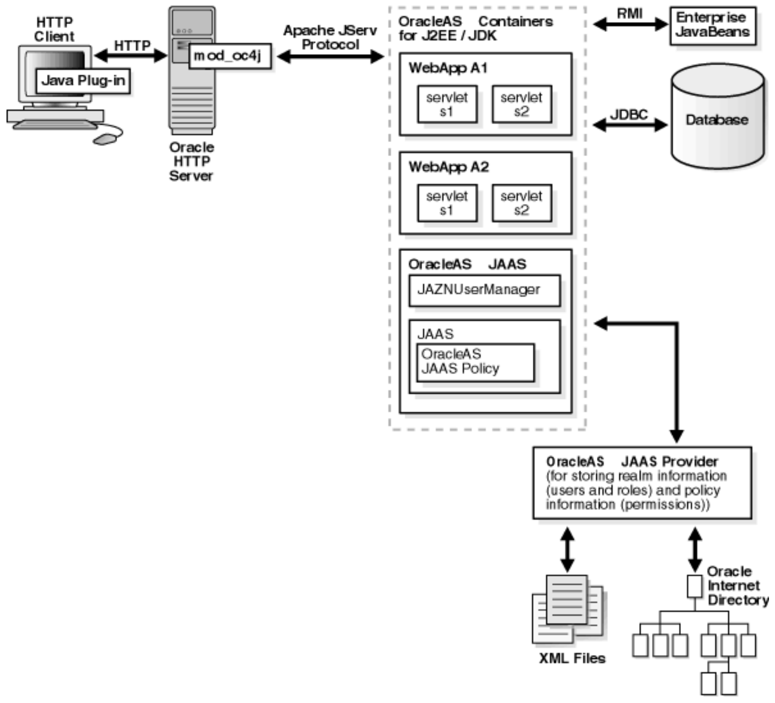
- Examines the authorization context associated with the current thread
- Determines that the current subject has the required permissions to write to the file

e. `SecurityManager.checkPermission()` returns safely and the client HTTP request proceeds.

Integrating the JAAS Provider with Basic Authentication

Basic authentication bypasses OracleAS Single Sign-On. Figure 6–3 shows specific JAAS integration in an application configured for Basic authentication in a J2EE environment.

Figure 6–3 Oracle Component Integration in j2ee Environment



Basic Authentication J2EE Environments: Typical Scenario

This section describes the responsibilities of Oracle components when an HTTP client request is initiated in a J2EE environment configured for Basic authentication. In this environment, OracleAS Single Sign-On is not used. A login module (for example, `RealmLoginModule`) is used.

Note: If you have configured Basic authentication, OC4J invokes the `RealmLoginModule` whenever the user credentials are required. For example, when a request hits a protected page, OC4J will ask the JAAS Provider to authenticate the user, then the `RealmLoginModule` will be invoked to authenticate the user, using the credentials sent by the user via the browser over HTTP.

---

1. An HTTP client attempts to access a Web application (named WebApp A1) hosted by Oracle Application Server Containers for J2EE (the Web container for executing servlets).
2. The JAAS Provider retrieves the user.
3. The final step or steps depend on the setting of the `runas-mode` in the `jazn-web-app` element.

   If the `runas-mode` is set to false, then the following happens:

   a. The target servlet is invoked.

   If the `runas-mode` is set to true, then the following happens:

   a. The JAAS Provider invokes the target servlet's `service()` method within a `PrivilegedAction` block through `Subject.doAs()`. The `JAZNUserManager` enforces security constraints.

      – When `Subject.doAs()` is called, JAAS consults the JAAS Provider for permissions associated with the SSO user through the `getPermissions()` method.

      – The JAAS Provider retrieves the permissions associated with the given grantee from the provider type (LDAP-based or XML-based), and updates the policy cache as appropriate. The JAAS Provider then returns the granted set of permissions to the JAAS Provider runtime.

      – The JAAS Provider runtime constructs a new `AccessControlContext` based on the permissions returned from `getPermissions()`.

b. The servlet's code runs under the `AccessControlContext` of the user.

c. The servlet's code attempts to write to a file in the operating system's file system, triggering a call to `SecurityManager.checkPermission()`.

d. The JVM then:

   – Examines the authorization context associated with the current thread

   – Determines that the current subject has the required permissions to write to the file

e. `SecurityManager.checkPermission()` returns safely and the client HTTP request proceeds.

See Also: Your Sun Java documentation for more information on J2EE by visiting the following URL:

```
http://java.sun.com/j2ee/
```

J2EE and JAAS Provider Role Mapping

Two distinct role types are available to application developers creating JAAS-integrated applications in J2EE environments: J2EE roles and JAAS roles. When these role types are mapped together using Oracle Application Server Containers for J2EE group mappings, users can access an application with a defined set of role permissions for as long as the user is mapped to this role.

This section describes these role types and how which they are mapped together.

- J2EE Security Roles
- JAAS Provider Roles and Users
- OC4J Group Mapping to J2EE Security Roles

J2EE Security Roles

The J2EE development environment includes a portable security roles feature defined in the `web.xml` file for servlets and Java Server Pages (JSPs). Security roles define a set of resource access permissions for an application. Associating a principal (in this case, a JAAS user or role) with a security role assigns the defined access permissions to that principal for as long as they are mapped to the role. For example, an application defines a security role called `sr_developer`:

```
<security-role>
     <role-name>sr_developer</role-name>
</security-role>
```

You also define the access permissions for the `sr_developer` role.

```
 <security-constraint>
    <web-resource-collection>
      <web-resource-name>access to the entire application</web-resource-name>
      <url-pattern>/*</url-pattern>
    </web-resource-collection>
        <!-- authorization -->
    <auth-constraint>
      <role-name>sr_developer</role-name>
    </auth-constraint>
  </security-constraint>
```

JAAS Provider Roles and Users

JAAS roles and users are defined depending on the provider type, LDAP-based or XML-based.

For example, with the XML-based provider type, developer is listed as a role element in the `jazn-data.xml` file:

```
<role>
  <name>developer</name>
  <members>
    <member>
      <type>user<type>
      <name>john<name>
    </member>
  </members>
</role>
```

OC4J Group Mapping to J2EE Security Roles

Oracle Application Server Containers for J2EE (OC4J) enables you to map portable J2EE security roles defined in the J2EE `web.xml` file to groups in an `orion-application.xml` file.

The roles and users defined in your provider environment are mapped to the Oracle Application Server Containers for J2EE `developer` group role in the `orion-application.xml` file.

For example, the `sr_developer` security role is mapped to the group named `developer.`

```
<security-role-mapping name="sr_developer">
```

```
      <group name="developer" />
  </security-role-mapping>
```

Notice that a `<group>` in a `<security-role-mapping>` element corresponds to a role in the JAAS provider. Therefore, this association permits the `developer` group to access the resources allowed for the `sr_developer` security role.

In this paradigm, the user `john` is listed as a member of the `developer` role. Because the `developer` group is mapped to the J2EE security role `sr_developer` in the `orion-application.xml` file, `john` has access to the application resources defined by the `sr_developer` role.

Authentication in the J2EE Environment

Authentication is the process of verifying the identity of a user in a computing system, often as a prerequisite to granting access to resources in a system. User authentication in the J2EE environment is performed by the following:

- OracleAS Single Sign-On (for SSO environments) or the JAAS Provider `RealmLoginModule` or other login module (for non-SSO environments)

  Before HTTP requests can be dispatched to the target servlet, the `JAZNUserManager` gets the authenticated user information (set by `mod_osso`) from the HTTP request object and sets the JAAS subject in Oracle Application Server Containers for J2EE.

- One of the following:
  - `JAZNUserManager`
  - `XMLUserManager`
  - A developer-supplied `UserManager`

Running with an Authenticated Identity

You can choose to configure the `JAZNUserManager` so that a filter enables the target servlet to run with the permissions and roles associated with an authenticated identity or run-as identity. To do this, configure the `jazn-web-app` element.

> **See Also:** "JAZNUserManager" on page 6-4 for further information on options and configuration of the `JAZNUserManager` filter, including the `jazn-web-app` element.

Retrieving Authentication Information

The following `javax.servlet.HttpServletRequest` APIs retrieve authentication information within the servlet:

- `getRemoteUser` for the authenticated username
- `getAuthType` for the authentication scheme
- `getUserPrincipal` for the authenticated principal object
- `getAttribute("java.security.cert.X509certificate")` for the SSL client certificate

Authorization begins with a call to `Subject.doAs()`.

Authorization in the J2EE Environment

Authorization is the process of granting the permissions and privileges entitled to the user. This section discusses authentication within servlets.

If the servlet is configured to permit `doAs()`, the `JAZNUserManager` invokes an authenticated target servlet within a `Subject.doAs()` block to enable JAAS-based authorization in the target servlets.

Authorization is achieved through the following:

- `JAZNUserManager`
- Methods based on the Java 2 Security Model:
  - `Servlet.service()` in the servlet
  - `Subject.doAs()` and `Subject.doAsPrivileged()` in the client
  - `SecurityManager.checkPermission()` in the server

See Also: Configuring Servlet Authorization (runas-mode and doasprivileged-mode) in <jazn-web-app> on page 3-13.

7 Custom LoginModules

This chapter discusses how to write and install a `LoginModule` to be used with the Oracle Application Server Containers for J2EE (OC4J) JAAS Provider. This chapter contains the following sections:

Note: Because the JAAS specification does not cover user management, when you configure your application to use a custom `LoginModule`, the use of the `UserManager` API within your application is effectively disabled. The J2EE API, however, will continue to function within your application.

Custom JAAS LoginModule Integration with OC4J

Because OC4J's support for JAAS fully complies with the JAAS 1.0 specification, users can plug in any JAAS-compliant `LoginModule` implementation, if desired. OC4J includes the `LoginModule`, `RealmLoginModule` that combines J2EE security constraints with either the XML-based or LDAP-based provider types for environments in which OracleAS Single Sign-On (SSO) is not available. When Oracle Internet Directory (OID) is in use, we recommend that you use Oracle Identity Management to integrate with other authentication and identity management systems.

See the *Oracle Identity Management Concepts and Deployment Planning Guide* for details.

A custom JAAS `LoginModule` may be desirable when OracleAS Identity Management is not available and users are defined in an external repository. For those cases, you can configure a `LoginModule` using the XML-based provider type, and the following preliminary questions need to be considered.

1. **Development**. Do you want to take advantage of J2EE security constraints?
2. **Development, packaging, and deployment**. Are you using the login modules that come with J2SE 1.4? Or are you deploying in-house or third-party login modules?

> **Note:** Custom login modules are supported only with the XML-based Provider.

Packaging and Deployment

If you are using one or more of the default login modules provided with J2SE 1.3 and 1.4 (such as the J2SE1.4 `com.sun.security.auth.module.Krb5LoginModule`), then no additional configuration is needed. The OracleAS JAAS Provider can locate the default login modules.

If you are deploying your application with a custom login module, then you must deploy the login module and configure the JAAS Provider properly so that the module can be found at runtime.

Four options are available when packaging and deploying your custom login modules:

- Deploying as Standard Extensions or Optional Packages
- Deploying Within the J2EE Application
- Using the OC4J Classloading Mechanism
- Using the JAAS Provider Classloading Mechanism

The remainder of this section discusses these options in greater detail.

Deploying as Standard Extensions or Optional Packages

If you deploy your login modules as standard extensions, the JAAS Provider will be able to find them. No additional configuration is necessary. Deploying login modules as standard extensions allows multiple applications to share the deployed login modules.

For example, one way to deploy your login modules as standard extensions is to deploy them to the `${java.home}/lib/ext` directory.

> **See Also:**
> `http://java.sun.com/j2se/1.4/docs/guide/extensions`

Deploying Within the J2EE Application

If your login module is used only by a single J2EE application rather than shared among multiple applications, then you can simply package your login module as part of your application, and the JAAS Provider will be able to find it. No additional configuration is necessary.

If a later application needs the same `LoginModule`, you must repackage the login module and any relevant classes with the new application.

If you want to allow multiple applications to share the same `LoginModule` but you cannot deploy the `LoginModule` as an extension, then you can consider using the OC4J classloading mechanism or the JAAS Provider classloading mechanism.

Using the OC4J Classloading Mechanism

The JAAS Provider is integrated with OC4J's classloading architecture. If you configure your application so that the deployed custom login modules are part of your application `classpath`, then the JAAS Provider can locate them.

One way to accomplish this is using the `<library>` element in either of the following files:

- `application.xml`
- `orion-application.xml`

> **See Also:** The Oracle Application Server Containers for J2EE Services Guide for more information about the `<library>` element.

Using the JAAS Provider Classloading Mechanism

If for some reason you cannot configure your application's `classpath`, you can take advantage of the JAAS Provider's classloading mechanism. Deploy your login modules and specify their location using the `classpath` property of the `<jazn>` tag. See Table 3–3, "(XML-Based Provider) The <jazn> Tag In orion-application.xml" for complete information on the `<jazn>` tag properties.

Configuration

You modify the following files to configure your application to take advantage of custom login modules:

- `jazn-data.xml`
- `orion-application.xml`

These files are discussed in greater detail below.

jazn-data.xml

This file serves as the repository for the XML-based provider.

Although many `jazn-data.xml` files can be associated with an OC4J instance, the `jazn-data.xml` specified in the default `jazn.xml` serves as the default repository for the OracleAS JAAS provider.

Note that Oracle supports only the XML-based provider in conjunction with custom login modules.

The following sections discuss these XML elements:

- `<jazn-loginconfig>`
- `<jazn-policy>`

<jazn-loginconfig>

This tag contains information that associates applications with login modules.

Example:

```
<jazn-loginconfig>
  <application>
    <name>sampleLM</name>
    <login-modules>
       <login-module>
```

```
          <class>sample.SampleLoginModule</class>
          <control-flag>required</control-flag>
       </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>
```

This sample fragment associates the application `sampleLM` with the login module `sample.SampleLoginModule`.

<jazn-policy>

This tag contains information that associates grantees with permissions.

Example:

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <class>sample.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
   </grantee>
   <permissions>
     <permission>
       <class>com.evermind.server.rmi.RMIPermission</class>
       <name>login</name>
     </permission>
   </permissions>
  </grant>
</jazn-policy>
```

This sample fragment grants the permission `com.evermind.server.rmi.RMIPermission` with target name `login` to the principal with class `sample.SampleUser` and name `ray`.

Note: Oracle recommends that you manage the contents of `jazn-data.xml` using the JAZN Admintool.

For more information about the JAZN Admintool, see Chapter 5, "Using the JAZN Admintool".

orion-application.xml

This file contains application configuration information specific to OC4J. The following tags are discussed in detail:

- `<jazn>`
- `<security-role-mapping>`
- <library>

<jazn>

For a full discussion of the `<jazn>` tag, see "The <jazn> Tag" on page 3-5.

The following properties are specific to `LoginModule` configuration:

- `role.mapping.dynamic`

  This property, when set to `true`, instructs the JAAS Provider to perform authorization checks based on the current `Subject` instead of based on users and roles defined in the application specific `jazn-data.xml`.

  The `LoginModule` instance(s) must ensure that the appropriate principals (users, roles, or groups) are associated with the `Subject` instance during the commit phase of the authentication process, in order for the principals to be taken into consideration during the authorization process. This association of principals to the `Subject` is typically implemented using the standard JAAS API.

- `classpath`

  This property, when set, tells the JAAS Provider where to look for third-party classes and JAR files if they cannot be found elsewhere. Example:

  ```
  <jazn provider="XML" location="./jazn-data.xml">
     <property name="classpath"
  value="../../shared/lib/sample.jar;../../shared/lib/samplemodule.jar" />
     <property name="role.mapping.dynamic" value="true" />
  </jazn>
  ```

  For details, see Table 3–3, "(XML-Based Provider) The <jazn> Tag In orion-application.xml".

<security-role-mapping>

This optional tag describes static security-role mapping information. For details, see the Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.

<library>

This tag sets the `classpath` associated with your application. Whenever possible, use this tag instead of the `classpath` property in the `<jazn>` tag. Example:

```
<library path="../../shared/lib/sample.jar"/>
<library path="../../shared/lib/samplemodule.jar"/>
```

For details, see the Oracle Application Server Containers for J2EE User's Guide.

Simple Login Module J2EE Integration

Developing a simple `LoginModule` follows the standard development, packaging, and deployment cycle. The following sections discuss each step in the cycle.

Development

Develop a JAAS-compliant `LoginModule` according to the JAAS SPI (see the Javadoc for `javax.security.auth.spi.LoginModule` for more information).

Packaging

Package your `LoginModule` in one of two ways:

- Package your `LoginModule` classes as part of your application's EAR file. For Web applications, include the classes under the `WEB-INF/classes`.
- Package it separately and refer to it using the classloading mechanisms.

Deployment

To deploy your `LoginModule` in the global `jazn-data.xml` file:

1. Register your application's login module within the `<application>` tag.

   The following entry registers the login module `sample.SampleLoginModule` to be used for authenticating users accessing the `sampleLM` application.

```
 <application>
```

```
         <name>sampleLM</name>
         <login-modules>
            <login-module>
               <class>sample.SampleLoginModule</class>
               <control-flag>required</control-flag>
               <options>
                  <option>
                     <name>debug</name>
                     <value>true</value>
                  </option>
               </options>
            </login-module>
         </login-modules>
      </application>
```

2. **Optional**. Grant relevant permissions to your users and roles.

   For example, if the principal `admin` needs EJB access, then you must grant the permission `com.evermind.rmi.RMIPermission` to `admin`.

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>sample.SampleUser</class>
        <name>admin</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>com.evermind.server.rmi.RMIPermission</class>
      <name>login</name>
    </permission>
  </permissions>
</grant>
```

To deploy your `LoginModule` in the application-specific `orion-application.xml` file:

1. Set the `<jazn>` property `role.mapping.dynamic` to `true`:

```
<jazn provider="XML" location="./jazn-data.xml" >
  <property name="role.mapping.dynamic" value="true" />
</jazn>
```

2. Create appropriate `<security-role-mapping>` entries.

```
<security-role-mapping name="sr_developer">
  <user name="developer" />
</security-role-mapping>
<security-role-mapping name="sr_manager">
  <group name="managers" />
</security-role-mapping>
```

8 JAAS and Enterprise Manager

The JAAS LDAP-based Provider stores information in Oracle Internet Directory (OID). This chapter describes how to use Oracle Enterprise Manager to manage data in the Oracle Application Server Containers for J2EE (OC4J) JAAS Provider.

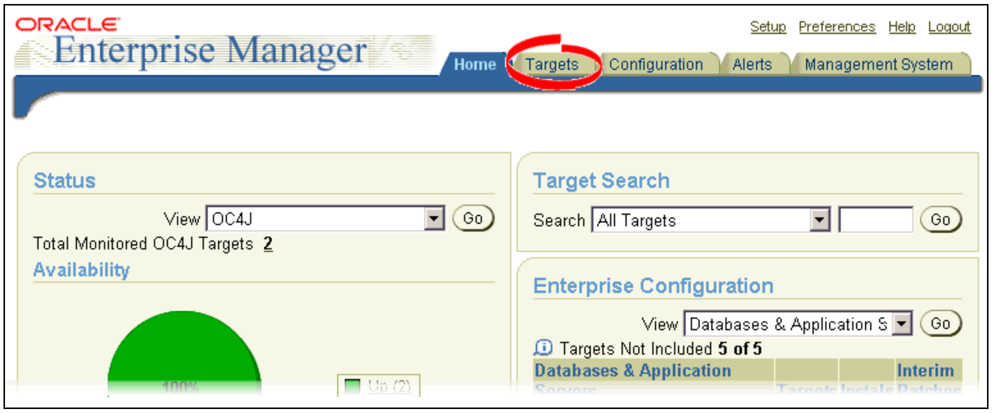
This chapter contains these topics:

Startup

Use this procedure to access the Oracle Enterprise Manager for OC4J Home page.

1. From the Enterprise Manager home, click **Targets**.

Figure 8–1 Enterprise Manager Home Tab

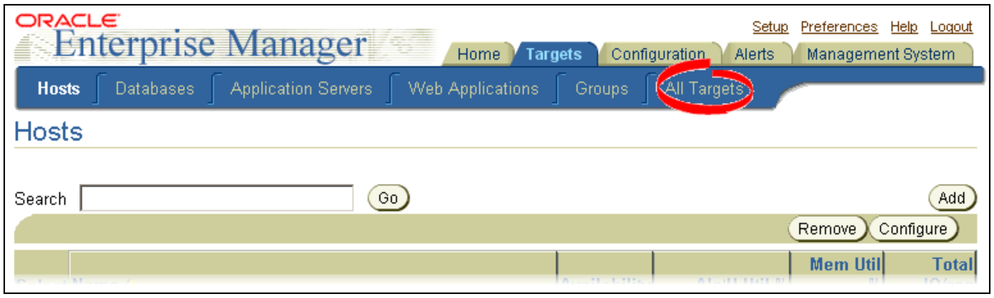


Total Monitored OC4J Targets 2

Targets Not Included 5 of 5

2. From the Targets tab, click **All Targets**.

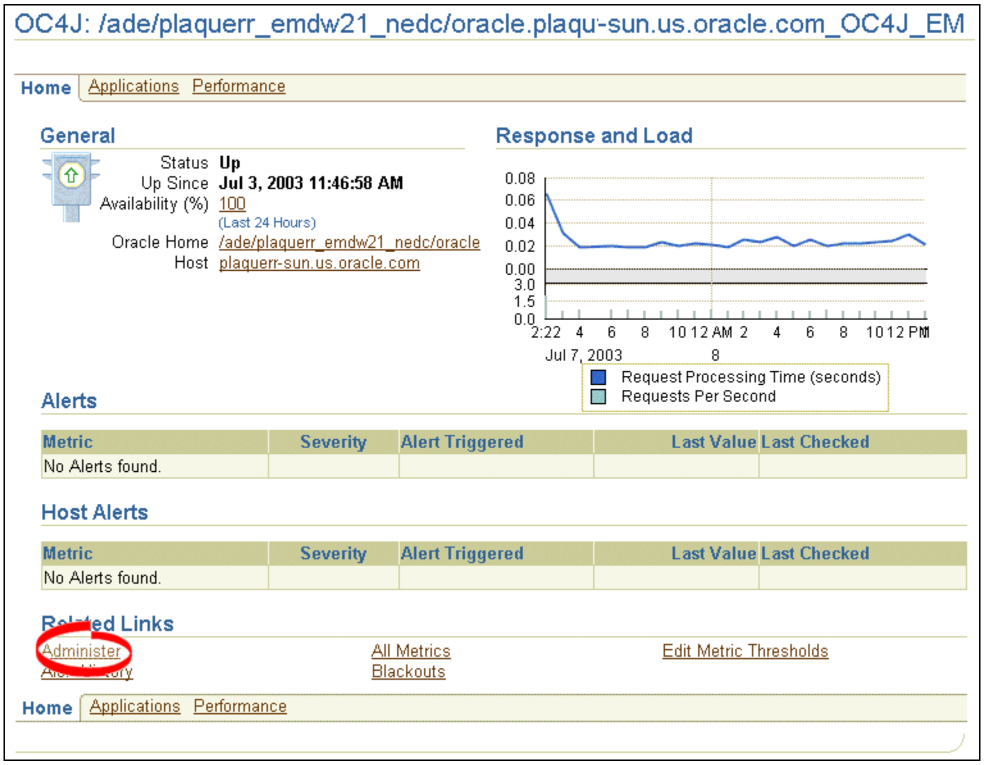
Figure 8–2 Enterprise Manager Targets Tab



3. From the All Targets page, select your specific OC4J instance.

4. From the Home page of the OC4J instance, click **Administer** in the Related Links area near the bottom of the page.

Figure 8–3 OC4J Instance Home Page



OC4J: /ade/plaquerr_emdw21_nedc/oracle.plaqu-sun.us.oracle.com_OC4J_EM

Home | Applications Performance

General

Status	**Up**
Up Since	**Jul 3, 2003 11:46:58 AM**
Availability (%)	100 (Last 24 Hours)
Oracle Home	/ade/plaquerr_emdw21_nedc/oracle
Host	plaquerr-sun.us.oracle.com

Response and Load

0.08
0.06
0.04
0.02
0.00
3.0
1.5
0.0
2:22 4 6 8 10 12 AM 2 4 6 8 10 12 PM
Jul 7, 2003 8
Request Processing Time (seconds)
Requests Per Second

Alerts

Metric	Severity	Alert Triggered	Last Value	Last Checked
No Alerts found.				

Host Alerts

Metric	Severity	Alert Triggered	Last Value	Last Checked
No Alerts found.				

Related Links

Administer	All Metrics	Edit Metric Thresholds
Alert History	Blackouts	

Home | Applications Performance

Note: You may be required to supply your username and password to log into the administer page.

---

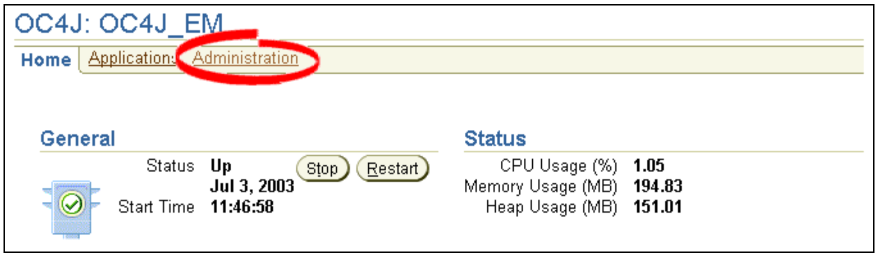
5. From the Enterprise Manager for Oracle Application Server OC4J Home page, select one of the following options:

   - To edit the global application security settings, click **Administration**. Continue with "Editing Global Security Settings" on page 8-4.
   - To edit an individual module's security settings, click **Application**. Continue with "Editing Individual Security Settings" on page 8-5.

Editing Global Security Settings

1. From the Enterprise Manager for Oracle Application Server OC4J Home page, click **Administration**.

Figure 8–4 Oracle Enterprise Manager for Oracle Application Server OC4J Home Page

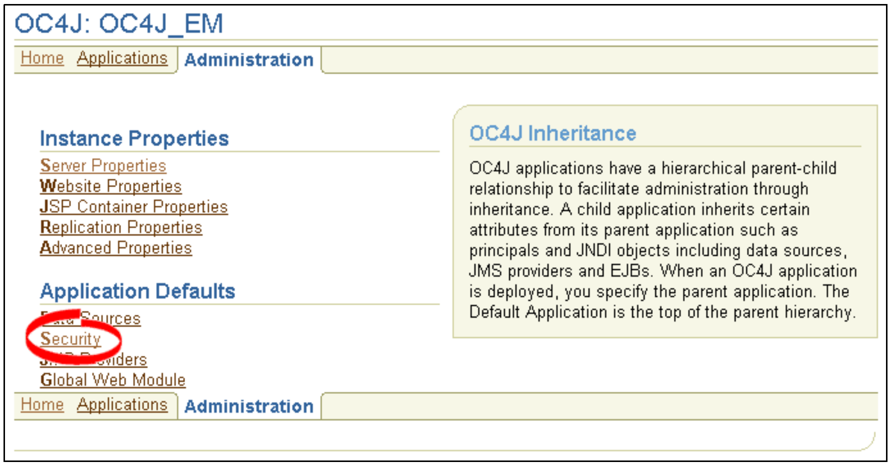


Status	Up
Start Time	Jul 3, 2003 11:46:58

CPU Usage (%)	1.05
Memory Usage (MB)	194.83
Heap Usage (MB)	151.01

2. From the Administration page, click **Security**.

Figure 8–5 Oracle Enterprise Manager for Oracle Application Server Administration



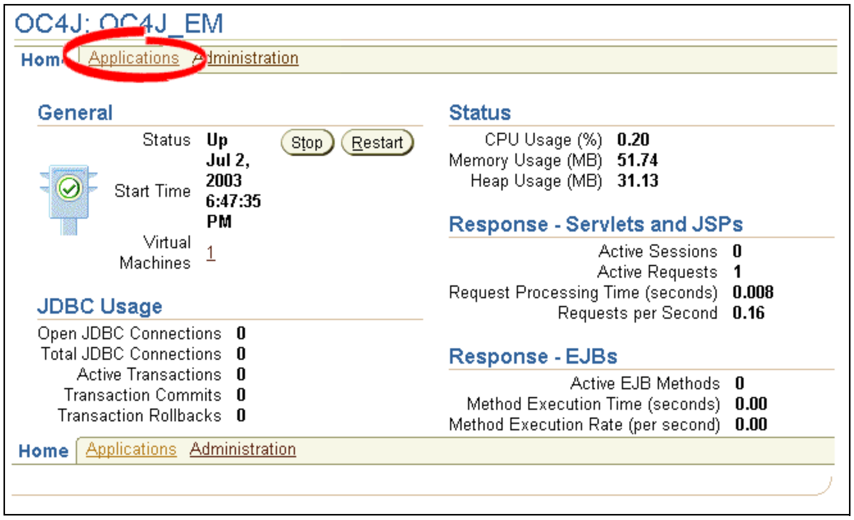
OC4J applications have a hierarchical parent-child relationship to facilitate administration through inheritance. A child application inherits certain attributes from its parent application such as principals and JNDI objects including data sources, JMS providers and EJBs. When an OC4J application is deployed, you specify the parent application. The Default Application is the top of the parent hierarchy.

The security page appears (see Figure 8–10 on page 8-9).

Editing Individual Security Settings

1. From the Enterprise Manager for Oracle Application Server OC4J Home page, click **Applications**.

Figure 8–6 Oracle Enterprise Manager for Oracle Application Server OC4J Home Page



OC4J: OC4J_EM

Home | Applications | Administration

General

Status **Up** Stop Restart
Start Time **Jul 2, 2003 6:47:35 PM**
Virtual Machines 1

JDBC Usage

Open JDBC Connections	**0**
Total JDBC Connections	**0**
Active Transactions	**0**
Transaction Commits	**0**
Transaction Rollbacks	**0**

Status

CPU Usage (%)	**0.20**
Memory Usage (MB)	**51.74**
Heap Usage (MB)	**31.13**

Response - Servlets and JSPs

Active Sessions	**0**
Active Requests	**1**
Request Processing Time (seconds)	**0.008**
Requests per Second	**0.16**

Response - EJBs

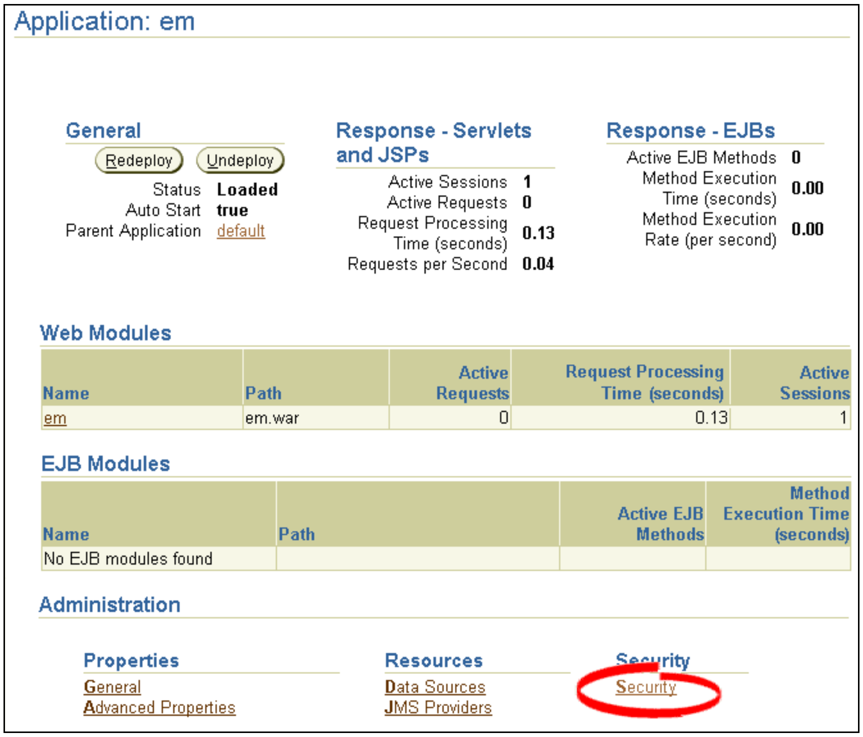
Active EJB Methods	**0**
Method Execution Time (seconds)	**0.00**
Method Execution Rate (per second)	**0.00**

Home | Applications | Administration

2. Select an deployed application or click **Default**.

3. From the module's Application page, click **Security**.

Figure 8–7 Oracle Enterprise Manager for Oracle Application Server Application Page



Application: em

General

Status	**Loaded**
Auto Start	**true**
Parent Application	default

Response - Servlets and JSPs

Active Sessions	**1**
Active Requests	**0**
Request Processing Time (seconds)	**0.13**
Requests per Second	**0.04**

Response - EJBs

Active EJB Methods	**0**
Method Execution Time (seconds)	**0.00**
Method Execution Rate (per second)	**0.00**

Web Modules

Name	Path	Active Requests	Request Processing Time (seconds)	Active Sessions
em	em.war	0	0.13	1

EJB Modules

Name	Path	Active EJB Methods	Method Execution Time (seconds)
No EJB modules found			

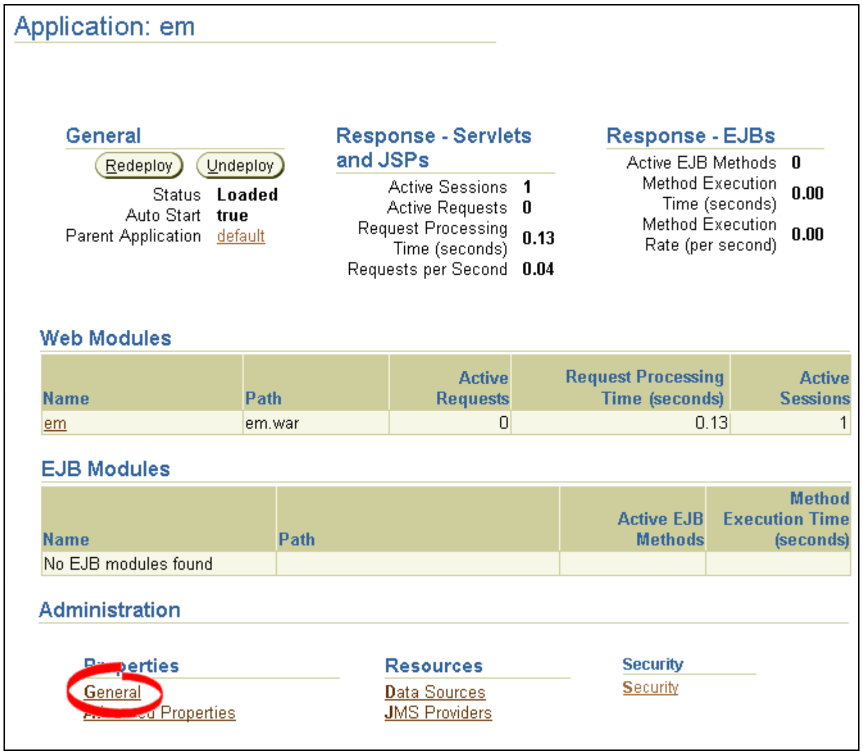
Administration

The Security page appears (see Figure 8–10 on page 8-9).

Selecting a UserManager

1. From a module's Application page (see Figure 8–7 on page 8-6), click **General** in the Administration Properties area.

Figure 8–8 Oracle Enterprise Manager for Oracle Application Server Application Page



Application: em

General

Redeploy | Undeploy

Status	**Loaded**
Auto Start	**true**
Parent Application	default

Response - Servlets and JSPs

Active Sessions	**1**
Active Requests	**0**
Request Processing Time (seconds)	**0.13**
Requests per Second	**0.04**

Response - EJBs

Active EJB Methods	**0**
Method Execution Time (seconds)	**0.00**
Method Execution Rate (per second)	**0.00**

Web Modules

Name	Path	Active Requests	Request Processing Time (seconds)	Active Sessions
em	em.war	0	0.13	1

EJB Modules

Name	Path	Active EJB Methods	Method Execution Time (seconds)
No EJB modules found			

Administration

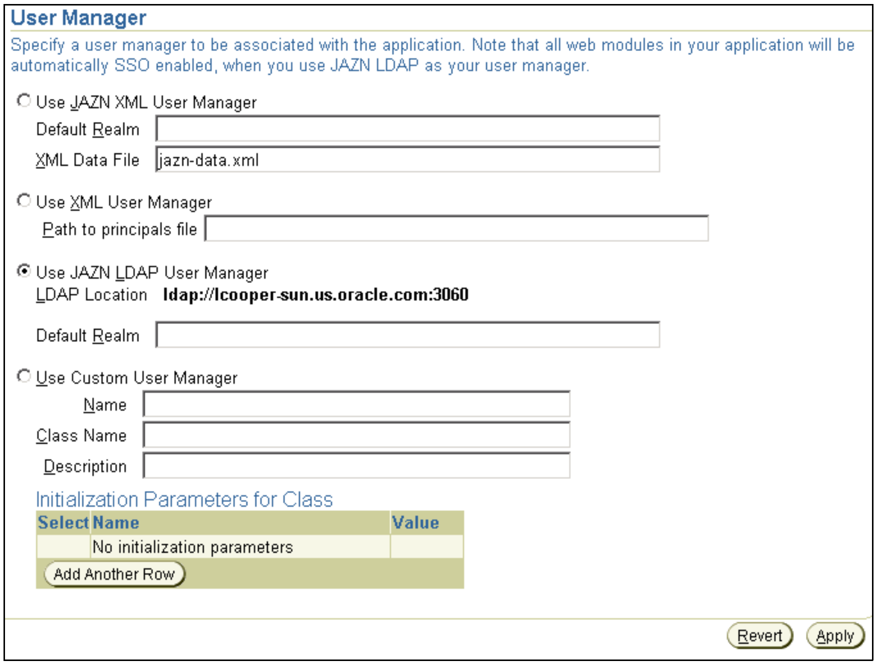
Properties
General
Advanced Properties

Resources
Data Sources
JMS Providers

Security
Security

2. On the Properties screen, scroll down to the **User Manager** area.

Figure 8–9 User Manager area of Properties Page



User Manager

Specify a user manager to be associated with the application. Note that all web modules in your application will be automatically SSO enabled, when you use JAZN LDAP as your user manager.

○ Use JAZN XML User Manager

Default Realm	
XML Data File	jazn-data.xml

○ Use XML User Manager

Path to principals file	

[✓ ● Use JAZN LDAP User Manager]

LDAP Location	ldap://lcooper-sun.us.oracle.com:3060
Default Realm	

○ Use Custom User Manager

Name	
Class Name	
Description	

Initialization Parameters for Class

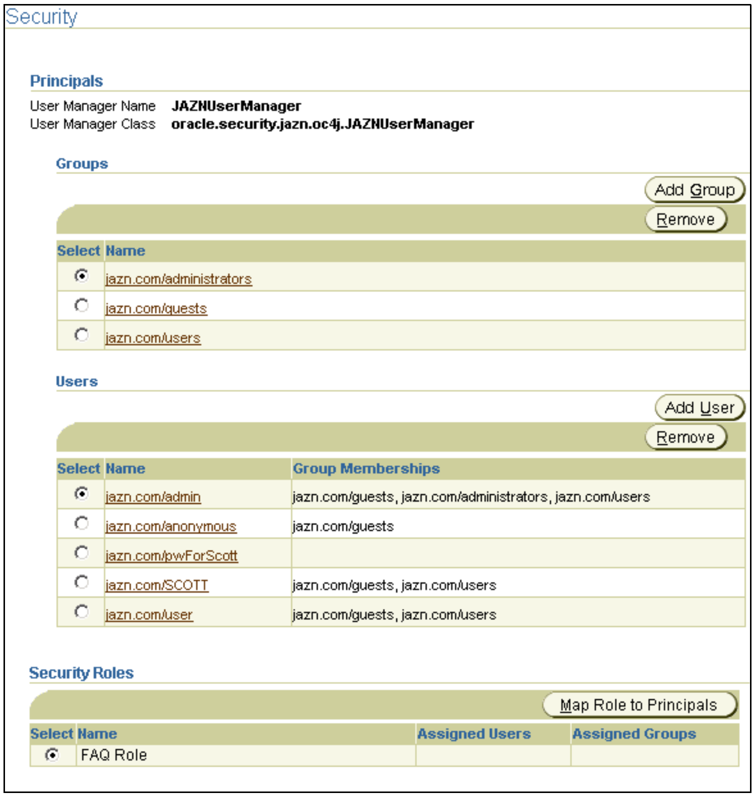
Select	Name	Value
	No initialization parameters	

3. Click the user manager you want to use, and fill in the appropriate pathname and realm information.

Mapping Security Roles

1. Navigate to the **Security** page as discussed in "Startup" on page 8-2 and click **Map Role To Principals** in the Security Roles area.

Figure 8–10 Security Page



Security

Principals

User Manager Name **JAZNUserManager**
User Manager Class **oracle.security.jazn.oc4j.JAZNUserManager**

Groups

Select	Name
●	jazn.com/administrators
○	jazn.com/guests
○	jazn.com/users

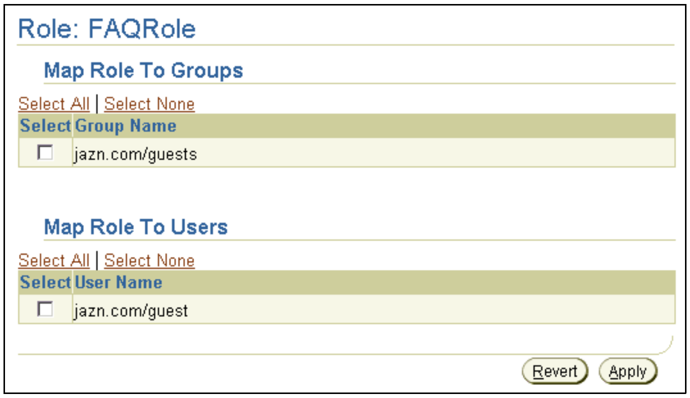
Users

Select	Name	Group Memberships
●	jazn.com/admin	jazn.com/guests, jazn.com/administrators, jazn.com/users
○	jazn.com/anonymous	jazn.com/guests
○	jazn.com/pwForScott	
○	jazn.com/SCOTT	jazn.com/guests, jazn.com/users
○	jazn.com/user	jazn.com/guests, jazn.com/users

Security Roles

Select	Name	Assigned Users	Assigned Groups
●	FAQ Role		

2. Select the group and/or user to map to the role and click **Apply**.



Role: FAQRole

Map Role To Groups

Select All | Select None

Select	Group Name
☐	jazn.com/guests

Map Role To Users

Select All | Select None

Select	User Name
☐	jazn.com/guest

Figure 8–11 Security: Map Role Screen

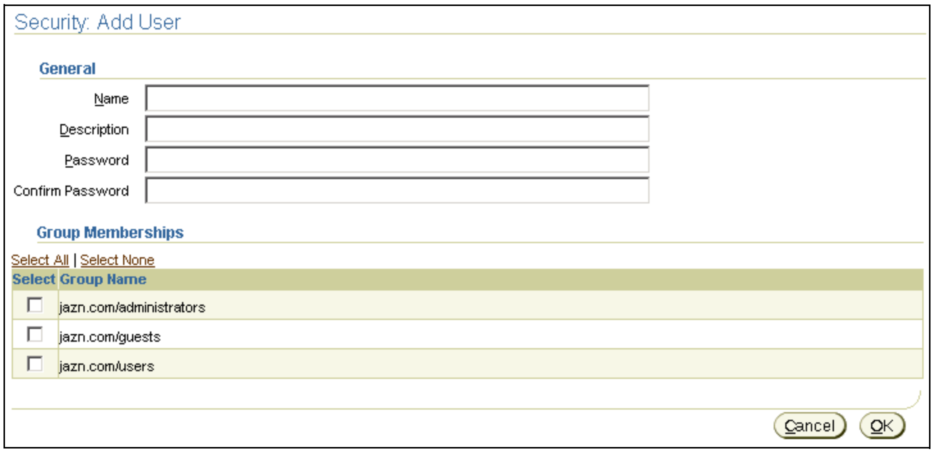
Creating Users

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide for details.

---

1. From the Security page (see Figure 8–10 on page 8-9), click **Add User** in the Users area. The **Security: Add User** screen appears.

Figure 8–12 Security: Add User Screen


Security: Add User

General

Name	
Description	
Password	
Confirm Password	

Group Memberships

Select All | Select None

Select	Group Name
☐	jazn.com/administrators
☐	jazn.com/guests
☐	jazn.com/users

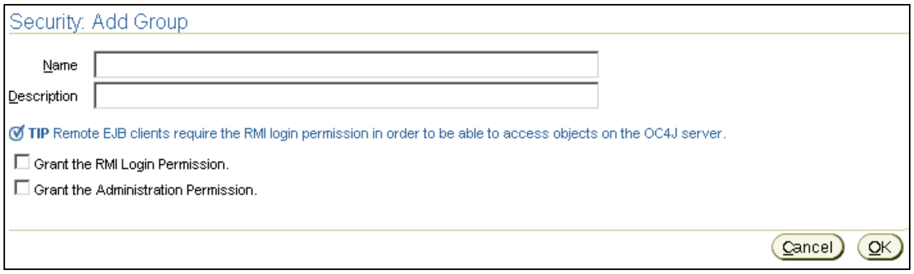
2. Fill in the **Name**, **Description**, **Password**, and **Confirm Password** fields and place checks beside any groups the user should be a member of. Click **OK**.

Creating Groups

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide for details.

1. From the Security page (see Figure 8–10 on page 8-9), click **Add Group** in the Groups area. The **Security: Add Group** screen appears.

Figure 8–13 Security: Add Group Screen


Security: Add Group

Name

Description

TIP Remote EJB clients require the RMI login permission in order to be able to access objects on the OC4J server.

☐ Grant the RMI Login Permission.

☐ Grant the Administration Permission.

2. Fill in the **Name** and **Description** fields and place checks beside any permissions you want to grant the group. Click **OK**.

Deleting Users Or Groups

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide for details.

---

1. From the Security page (see Figure 8–10 on page 8-9), select the user or group from the appropriate list.
2. Click **Remove**.
3. A confirmation screen appears asking whether you want to remove the specified user or group. Click **Yes**.

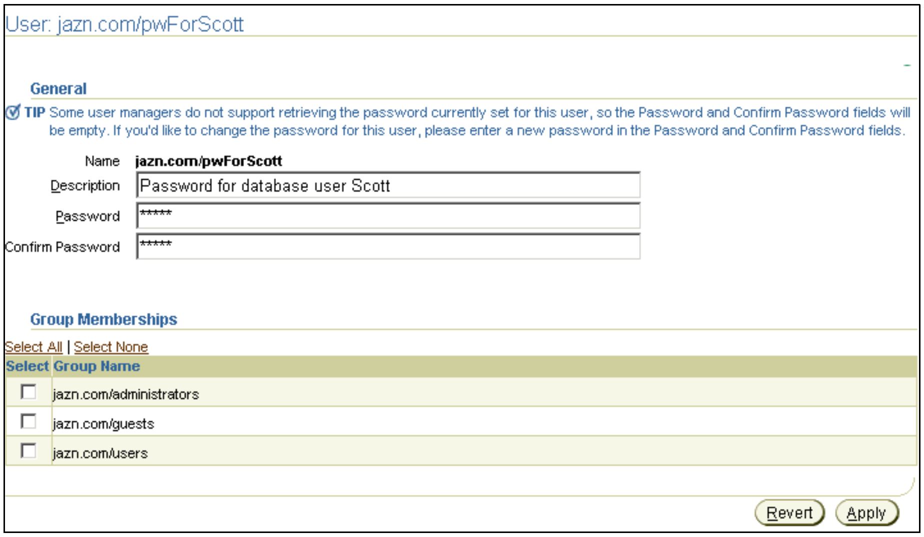
Editing Users

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide for details.

---

1. From the Security page (see Figure 8–10 on page 8-9), select a user.

Figure 8–14 User Screen



User: jazn.com/pwForScott

General

TIP Some user managers do not support retrieving the password currently set for this user, so the Password and Confirm Password fields will be empty. If you'd like to change the password for this user, please enter a new password in the Password and Confirm Password fields.

Name	jazn.com/pwForScott
Description	Password for database user Scott
Password	*****
Confirm Password	*****

Group Memberships

Select All | Select None

Select	Group Name
☐	jazn.com/administrators
☐	jazn.com/guests
☐	jazn.com/users

2. Type the new description or password into the appropriate text box. (To help avoid typographical errors, you must type the password twice.)
3. Click **Apply**.

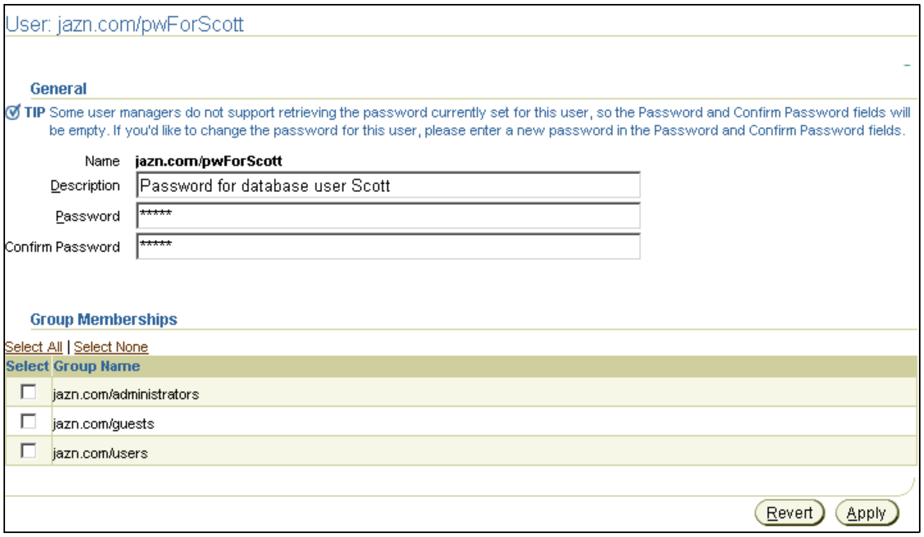
Assigning Users To Groups

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator's Guide for details.

---

1. From the Security page (see Figure 8–10 on page 8-9), select the user from the **Users** list. The **User** screen appears.

Figure 8–15 User Screen



User: jazn.com/pwForScott

General

TIP Some user managers do not support retrieving the password currently set for this user, so the Password and Confirm Password fields will be empty. If you'd like to change the password for this user, please enter a new password in the Password and Confirm Password fields.

Name	jazn.com/pwForScott
Description	Password for database user Scott
Password	*****
Confirm Password	*****

Group Memberships

Select All | Select None

Select	Group Name
☐	jazn.com/administrators
☐	jazn.com/guests
☐	jazn.com/users

2. To add the selected user to a group, click the group’s checkbox.
3. Click **Apply**.

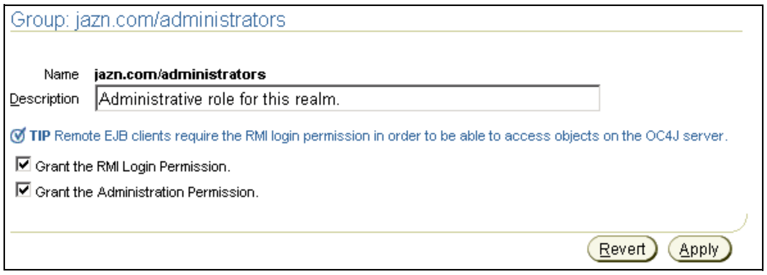
Granting Permissions To Groups

Note: Enterprise Manager manages only XML-based roles and users. To manage LDAP-based users and roles, use the Delegated Administration Service (DAS); see the Oracle Internet Directory Administrator’s Guide for details.

---

1. From the Security page (see Figure 8–10 on page 8-9), select the group from the **Groups** list. The **Group** screen appears.

Figure 8–16 Group Screen



Group: jazn.com/administrators

Name **jazn.com/administrators**

Description Administrative role for this realm.

TIP Remote EJB clients require the RMI login permission in order to be able to access objects on the OC4J server.

☑ Grant the RMI Login Permission.

☑ Grant the Administration Permission.

2. To grant the group RMI or Administration permission, click the appropriate checkbox.
3. Click **Apply**.

Part II
Other Technologies

This part discusses Java technologies other than JAAS that affect application security.

This part contains the following chapters:

9 Java 2 Security

This chapter discusses Java 2 Security features. It contains the following sections:

- Introduction
- JAAS Provider Permission Classes
- Creating a Java 2 Policy File
- The Java 2 Security Manager

Introduction

The Java 2 Security Model is fundamental to the JAAS Provider. The Java 2 Security Model enables configuration of security at all levels of restriction. This provides developers and administrators with increased control over many aspects of enterprise applet, component, servlet, and application security.

> **See Also:** For a tutorial on Java 2 Security, see `http://java.sun.com/docs/books/tutorial/security1.2/index.html`. For full information on Java 2 Security, see `http://java.sun.com/security`.

Permissions

Permissions are the basis of the Java 2 Security Model. All Java classes (whether run locally or downloaded remotely) are subject to a configured security policy that defines the set of permissions available for those classes. Each permission represents a specific access to a particular resource. Table 9–1 identifies the elements that comprise a Java permission instance.

Table 9–1 Java Permission Instance Elements

Element	**Description**	**Example**
Class name	The permission class	`java.io.FilePermission`
Target	The target name (resource) to which this permission applies	Directory `/home/*`
Actions	The actions associated with this target	Read, write, and execute permissions on directory `/home/*`

Protection Domains

Each Java class, when loaded, is associated with a protection domain. Protection domains can be configured for all levels of restriction (from complete restriction on resources to full access to all resources). Each protection domain is assigned a group of permissions based on a configured security policy at Java virtual machine (JVM) startup.

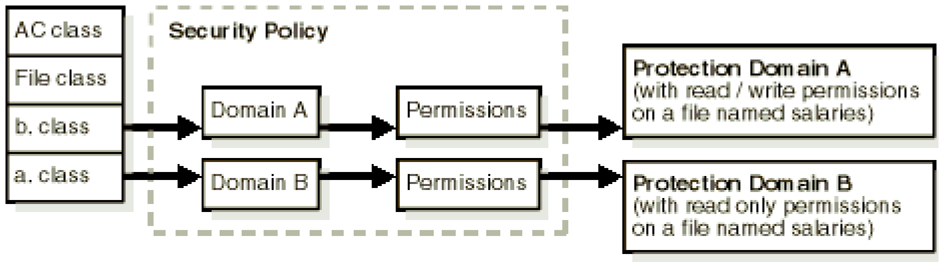
At runtime, the authorization check is done by stack introspection. This consists of reviewing the runtime stack and checking permissions based on the protection domains associated with the classes on the stack. This is typically triggered by a call to either:

- `SecurityManager.checkPermission()`
- `AccessController.checkPermission()`

The permission set in effect is defined as the intersection of all permission sets assigned to protection domains at the moment of the security check.

Figure 9–1 shows the basic model for authorization checking at runtime.

Figure 9–1 Java 2 Security Model



See Also:

- Chapter 4, "JAAS Provider Administration Tasks"
- Sun Java documentation at the URL

  `http://java.sun.com/security/`

JAAS Provider Permission Classes

Table 9–2 lists the permission classes furnished by the JAAS Provider. These classes allow applications to control access to resources. For information about the classes discussed, see the JAAS Provider Javadoc.

Table 9–2 JAAS Provider Permission Classes

Permission	Part of Package	Description
`AdminPermission`	`oracle.security.jazn.policy`	Represents the right to administer a permission (that is, grant or revoke another user's permission assignment).
`RoleAdminPermission`	`oracle.security.jazn.policy`	The grantee of this permission is granted the right to further grant/revoke the target role.
`JAZNPermission`	`oracle.security.jazn`	For authorization permissions. `JAZNPermission` contains a name (also called a target name), but no actions list; you either have or do not have the named permission.
`RealmPermission`	`oracle.security.jazn.realm`	Represents permission actions for a realm (such as `createRealm`, `dropRealm`, and so on). `RealmPermission` extends from `java.security.Permission`, and is used like any regular Java permission.

Creating a Java 2 Policy File

The Java 2 policy file grants permissions to trusted code or applications that you run. This enables code or applications to access Oracle support for JAAS or JDK APIs requiring specific access privileges.

A preconfigured Java 2 policy (`java2.policy`) is provided in *`ORACLE_HOME`*`/j2ee/home/config`.

You need to modify the Java 2 policy file to grant permissions to trusted code or applications.

For example, the following section of a `java2.policy` file grants `java.security.AllPermission` to the trusted `jazn.jar`.

```
/* grant the JAZN library AllPermission */
grant codebase "file:${oracle.home}/j2ee/home/jazn.jar" {
    permission java.security.AllPermission;
};
```

The following example grants specific permissions to all applications running in the `$ORACLE_HOME/appdemo` directory.

```
/* Assuming you are running your application demo in $ORACLE_HOME/appdemo/, */
/* Grant JAZN permissions to the demo to run JAZN APIs*/
grant codebase "file:/${oracle.ons.oraclehome}/appdemo/-" {
   permission oracle.security.jazn.JAZNPermission "getPolicy";
   permission oracle.security.jazn.JAZNPermission "getRealmManager";
   permission oracle.security.jazn.policy.AdminPermission
"oracle.security.jazn.realm.RealmPermission$*$createRealm,dropRealm,
  createRole, dropRole,modifyRealmMetaData";
```

The Java 2 Security Manager

The JAAS Provider checks permissions only when a `SecurityManager` has been installed. You specify a `SecurityManager` in one of two ways:

- Calling `System.setSecurityManager()`
- Setting the system property `java.security.manager` when starting OC4J

You can use either mechanism to install either the default `SecurityManager` or a custom `UserManager`.

Note: You set system properties by using the `-D` command-line option in Enterprise Manager; see the Advanced Configuration chapter in the fOracle Application Server Containers for J2EE User's Guide for details.

The permissions granted to particular classes by the default `SecurityManager` are determined by reading a *policy file*. The default policy file is supplied as part of J2EE. You can specify a policy file explicitly using the system property `java.security.policy`, as in

```
-Djava.security.policy=policyfilepath
```

Within an Oracle9*i* Application Server installation, OC4J instances run by default with no `SecurityManager`. If you choose to install a `SecurityManager`, you must specify one that does not interfere with normal OC4J functions. You can find a sample policy file at *`ORACLE_HOME`*`/j2ee/home/config/java2.policy`. The sample file grants "`AllPermission`" to most OC4J JARs. A typical block in this file looks like:

```
grant codebase "file:${oracle.home}/j2ee/home/ejb.jar" {
   permission java.security.AllPermission;
};
```

Note the use of "`${oracle.home}`" to specify the location of *`ORACLE_HOME`*. You can set `oracle.home` by specifying the system property:

```
-Doracle.home=oraclehomepathname
```

Path canonicalization follows the rules of `java.io.File`. On UNIX, the path cannot contain any symbolic links. If you do not specify a canonical path, then the default `SecurityManager` will not apply the `codebase` specification in the policy file.

You may need to grant additional permissions to your application code and to classes generated by OC4J.   The sample `java2.policy` file contains at the bottom a block that was required to run a demo with Java 2 security enabled. The required permissions will depend on the details of your application and the required codebase will depend on the details of your installation.

Using PrintingSecurityManager To Debug Java 2 Policy

In order to simplify the determination of what permissions need to be granted, Oracle supplies a custom `SecurityManager`, `PrintingSecurityManager`, that never throws a `SecurityException`. Instead, whenever a security exception would normally be thrown, `PrintingSecurityManager` prints messages specifying what exceptions the default `SecurityManager` would have thrown. To determine what permissions must be granted to your application, start OC4J using `PrintingSecurityManager` and execute your application.

When you run an application with the default `SecurityManager`, the application terminates when the first `SecurityException` is thrown; after you correct the first problem, you must execute the application again and again to track down all the causes of the exception. By using `PrintingSecurityManager`, you can create one large list of all the `SecurityExceptions`.

To install the `PrintingSecurityManager,` you must specify it on the command line that starts OC4J, as in

```
-Djava.security.manager=oracle.oc4j.security.PrintingSecurityManager
-Djava.security.policy=yourpolicypath
```

You must grant sufficient permissions (normally `java.security.AllPermission`)to `oc4j.jar`, which contains `PrintingSecurityManager`. If you have granted insufficient permissions, `PrintingSecurityManager` goes into an infinite loop as certain operations it performs triggers permission checks that fail. This eventually causes a `StackOverflowError`.

`PrintingSecurityManager` prints two kinds of messages on `System.out`. The first is

```
SecurityManager would throw java.security.AccessControlException: access denied
  (java.io.FilePermission path read)
```

where *`path`* is the actual path of your class. This message means that the default `SecurityManager` would have thrown the specified exception. Whenever the `PrintingSecurityManager` generates such a message, it tries to determine what codesource would need to be granted the indicated `Permission`. If the `PrintingSecurityManager` succeeds in determining the needed permission, it prints a diagnostic message that looks like:

```
(file:path/yourclass) needs (java.io.FilePermission path read)
```

`PrintingSecurityManager` cannot completely reproduce the actions of the default `SecurityManager`; this means that in some cases this message is an incorrect guess.

10 Password Management

This chapter discusses managing passwords within XML files. It contains the following sections:

Introduction

Many OC4J components require passwords for authentication. Embedding these passwords into deployment and configuration files poses a security risk, especially if the permissions on the files allow them to be read by any user. To avoid this problem, OC4J provides two solutions:

- *password obfuscation*, which replaces passwords stored in cleartext files with an encrypted version of the password. This is discussed in "Password Obfuscation In jazn-data.xml and jazn.xml".
- *password indirection*, which replaces cleartext passwords with information necessary to look up the password in another location. This is discussed in "Creating An Indirect Password".

Password Obfuscation In jazn-data.xml and jazn.xml

The JAAS configuration files, `jazn.xml` and `jazn-data.xml`, contain user names and passwords for JAAS authorization. To protect these files, OC4J uses password obfuscation.

Whenever you update jazn.xml or `jazn-data.xml`, OC4J reads the file, then rewrites it with obfuscated (encrypted) versions of all passwords. In all other OC4J configuration files, you can avoid exposing password cleartext by using password indirection, as "Creating An Indirect Password" explains below.

The JAAS Provider does not obfuscate passwords in `orion-application.xml`. This means that you should not embed passwords within a `<jazn>` element that is stored in `orion-application.xml`.

If you are using the LDAP-based provider, you should create a separate `jazn.xml` file that contains a `<jazn>` element defining your application; this file does not contain any user or group data. This `<jazn>` element looks like:

```
<jazn provider="LDAP" location="yourlocation">
    <property name="ldap.name" value="cn=orcladmin" />
    <property name="ldap.password" value="!welcome1" />
</jazn>
```

You then create a `<jazn>` element in `orion-application.xml` that points to the `jazn.xml` file using the `config` attribute, as in:

```
<jazn config="./jazn.xml" />
```

JAZN automatically obfuscates the password stored in this separate `jazn.xml` file the first time it reads this file.

Hand-editing jazn-data.xml

If you prefer, you can directly edit `jazn-data.xml` with a text editor. The next time OC4J reads `jazn-data.xml`, it will rewrite the file with all passwords obfuscated and unreadable.

Setting the `clear` attribute of the `<credentials>` element to `true` enables you to use clear (human-readable) passwords in the `jazn-data.xml` file.

```
<credentials clear="true">welcome</credentials>
<credentials>!welcome</credentials>
```

Creating An Indirect Password

The following OC4J XML configuration and deployment files support password indirection in one or more entities:

- `data-sources.xml`—`password` attribute of `<data-source>` element
- `ra.xml` — `<res-password>` element
- `rmi.xml`— `password` attribute of `<cluster>` element
- `application.xml`— `password` attributes of `<resource-provider>` and `<commit-coordinator>` elements
- `jms.xml`— `<password>` element
- `internal-settings.xml`— `<sep-property>` element, attributes `name=" keystore-password"` and `name=" truststore-password"`

To make any of these passwords indirect, replace the literal password string with a string containing "`->`" followed by either the username or by the realm and username separated by a slash ("`/`").

Note: To begin a literal (non-indirect) password with the string "`->`", precede the password by "`->!`". For instance, you would represent the direct password "`->silly`" as "`->!->silly`".

Indirect Password Examples

- `<data-source password="->Scott">`— Use `JaznUserManager` to look up `Scott` in the `JaznUserManager,` and use the password stored there.
- `<res-password="->customers/Scott">`— Use `JaznUserManager` to look up `Scott` in the `customers` realm, and use the password stored there.
- `<cluster password="martha">`—The literal string "`martha`" is the password; the password is not indirect.

Specifying a UserManager In orion-application.xml

The `<password-manager>` element specifies the `UserManager` that the global application uses to look up indirect passwords. (See "Creating An Indirect Password" on page 10-3.) If this element is omitted, the `UserManager` of the global application is used for authentication and authorization of indirect passwords. The `<jazn>` element within a `<password-manager>` element can be different from the `<jazn>` element at the top level.

For example, you can use an LDAP-based `UserManager` for the regular `UserManager`, but use an XML-based `UserManager` to authenticate indirect passwords. This is the only way to use indirect passwords in LDAP.

For full details, see"Specifying a UserManager In orion-application.xml" on page 3-22.

Note: It is possible to use pluggable UserManagers as password managers. However, if you use `XMLUserManager` as your password manager, principals.xml will not have passwords obfuscated.

11 Oracle HTTPS for Client Connections

This chapter describes the Oracle Application Server Containers for J2EE (Oracle Application Server Containers for J2EE) implementation of HTTPS that provides SSL functionality to client HTTP connections. The following topics are included:

Introduction

This chapter discusses how to use the Secure Sockets Layer protocol to communicate securely between networked applications. It begins by discussing fundamental SSL concepts, then continues with information about using Oracle HTTPS and JSSE.

Notes:

- Secure communication between a client and Oracle HTTP Server is independent of secure communication between Oracle HTTP Server and OC4J. (Also note that the secure AJP protocol used between Oracle HTTP Server and OC4J is not visible to the end user.) This section covers only secure communication between OC4J and the client.
- OC4J standalone supports SSL communication directly between a client and OC4J, using HTTPS. This is discussed in "Configuring OC4J Standalone for SSL" on page 11-25.

Overview of SSL Keys and Certificates

In SSL communication between two entities, such as companies or individuals, the server has a *public key* and an associated *private key*. Each key is a number, with the private key of an entity being kept secret by that entity, and the public key of an entity being publicized to any other parties with which secure communication might be necessary. The security of the data exchanged is guaranteed by keeping the private key secret, and by the complex encryption algorithm. This system is known as *asymmetric encryption*, because the key used to encrypt data is not the same as the key used to decrypt data.

Asymmetric encryption has a performance cost due to its complexity. A much faster system is *symmetric encryption*, where the same key is used to encrypt and decrypt data. But the weakness of symmetric encryption is that the same key has to be known by both parties, and if anyone intercepts the exchange of the key, then the communication becomes insecure.

SSL uses both asymmetric and symmetric encryption to communicate. An asymmetric key (*PKI public key*) is used to encode a symmetric encryption key (the *bulk encryption key*); the bulk encryption key is then used to encrypt subsequent communication. After both sides agree on the bulk encryption key, faster communication is possible without losing security and reliability.

When an SSL session is negotiated, the following steps take place:

1. The server sends the client its public key.
2. The client creates a bulk encryption key, often a 128 bit RC4 key, using a specified encryption suite.

3. The client encrypts the bulk key with the server's public key, and sends the encrypted bulk key to the server.
4. The server decrypts the bulk encryption key using the server's private key.

This set of operations is called *key exchange*. After key exchange has taken place, the client and the server use the bulk encryption key to encrypt all exchanged data.

Note: It is possible, but rare, for the client to have its own private and public keys as well.

In SSL the public key of the server is sent to the client in a data structure known as an X.509 certificate. This certificate, created by a *certificate authority* (CA), contains a public key, information concerning the owner of the certificate, and optionally some digital rights of the owner. Certificates are digitally signed by the CA which created them using that CA's digital certificate public key.

In SSL, the CA's signature is checked by the receiving process to ensure that it is on the *approved list* of CA signatures. This check is sometimes performed by analysis of certificate chains. This occurs if the receiving process does not have the signing CA's public key on the approved list. In that case the receiving process checks to see if the signer of the CA's certificate is on the approved list or the signer of the signer, and so on. This chain of certificate, signer of certificate, signer of signer of certificate, and so on is a *certificate chain*. The highest certificate in the chain (the original signer) is called the *root certificate* of the certificate chain.

The root certificate is often on the approved list of the receiving process. Certificates in the approve list are called *trust points* or trusted certificates. A root certificate can be signed by a CA or can be *self-signed*, meaning that the digital signature that verifies the root certificate is encrypted through the private key that corresponds with the public key that the certificate contains, rather than through the private key of a higher CA.

Functionally, a certificate acts as a container for public keys and associated signatures. A single certificate file can contain one or multiple chained certificates, up to an entire chain. Private keys are normally kept separately to prevent them from being inadvertently revealed, although they can be included in a separate section of the certificate file for convenient portability between applications.

A *keystore* is used to store certificates, including the certificates of all trusted parties, for use by a program. Through its keystore, an entity such as OC4J (for example) can authenticate other parties as well as authenticate itself to other parties. Oracle HTTP Server has what is called a *wallet* for the same purpose. Sun's SSL implementation introduces the notion of a *truststore*, which is a keystore file that includes the trusted certificate authorities that a client will implicitly accept during an SSL handshake.

In Java, a keystore is a `java.security.KeyStore` instance that you can create and manipulate using the `keytool` utility that is provided with the Sun Microsystems JDK. The underlying physical manifestation of this object is a file. Go to the following site for information about `keytool`:

`http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html`

Creating Keys and Certificates With OC4J and Oracle HTTP Server

The steps for using keys and certificates for SSL communication in OC4J are as follows. These are server-level steps, typically executed prior to deployment of an application that will require secure communication, perhaps when you first set up an OracleAS instance.

1. Use `keytool` to generate a private key, public key, and unsigned certificate.You can place this information into either a new keystore or an existing keystore.
2. Obtain a signature for the certificate, using either of the following two approaches.

   You can generate your own signature:

   a. Use `keytool` to "self-sign" the certificate. This is appropriate if your clients will trust you as, in effect, your own certificate authority.

   Alternatively, you can obtain a signature from a recognized certificate authority:

   a. Using the certificate from Step 1, use `keytool` to generate a *certificate request*, which is a request to have the certificate signed by a certificate authority.
   b. Submit the certificate request to a certificate authority.
   c. Receive the signature from the certificate authority and import it into the keystore, again using `keytool`. In the keystore, the signature will be matched with the associated certificate.

Note: Oracle Application Server includes OracleAS Certificate Authority (OCA). This allows customers to create and issue certificates for themselves and their users, although these certificates would likely be unrecognized outside a customer's organization without prior arrangements. See the *Oracle Application Server 10g Security Guide* for information about OCA.

The process for requesting and receiving signatures is up to the particular certificate authority you use. Because that is outside the scope and control of OracleAS, the OracleAS documentation does not cover it. You can go to the Web site of any certificate authority for information. (Any browser should have a list of trusted certificate authorities.) Here are the Web addresses for VeriSign, Inc. and Thawte, for example:

```
http://www.verisign.com/
```

```
http://www.thawte.com/
```

For SSL communication between OC4J and Oracle HTTP Server, you would also execute the preceding steps for Oracle HTTP Server, but using a wallet and Oracle Wallet Manager instead of a keystore and the `keytool` utility. See the *Oracle Application Server 10g Security Guide* for information about wallets and the Oracle Wallet Manager.

In addition, you would execute the following steps as appropriate.

If the OC4J certificate is signed by an entity that Oracle HTTP Server does not yet trust:

3. From OC4J, use `keytool` to export the OC4J certificate. This places the certificate into a file that is accessible to Oracle HTTP Server.
4. From Oracle HTTP Server, use Oracle Wallet Manager to import the OC4J certificate.

If the Oracle HTTP Server certificate is signed by an entity that OC4J does not yet trust, and if OC4J is in a mode of operation that requires *client authentication* (as discussed in "Requesting Client Authentication" on page 11-7):

5. From Oracle HTTP Server, use Oracle Wallet Manager to export the Oracle HTTP Server certificate. This places the certificate into a file that is accessible to OC4J.
6. From OC4J, use `keytool` to import the Oracle HTTP Server certificate.

During communications over SSL between Oracle HTTP Server and OC4J, all data on the communications channel between the two is encrypted. The following steps are executed:

1. The OC4J certificate chain is authenticated to Oracle HTTP Server during establishment of the encrypted channel.
2. Optionally, if OC4J is in client-authentication mode, Oracle HTTP Server is authenticated to OC4J. This also occurs during establishment of the encrypted channel.
3. The bulk encryption key is securely exchanged using the PKI public key, and is then used for the encryption of further communications on the channel.

Example: Creating an SSL Certificate and Generating Your Own Signature

This example corresponds to Step 2 above, in the mode where you generate your own signature by using `keytool` to self-sign the certificate.

First, create a keystore with an RSA private/public keypair, using the `keytool` command. This example (in which `%` is the system prompt) uses the RSA keypair algorithm to generate a keystore to reside in a file named `mykeystore`, which has a password of `123456` and is valid for 21 days:

```
% keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456 -validity 21
```

Note the following:

- The `keystore` option specifies the name of the file in which the keys are stored.
- The `storepass` option sets the password for protecting the keystore.
- The `validity` option sets the number of days for which the certificate is valid.

The `keytool` prompts you for more information, as follows:

```
What is your first and last name?
  [Unknown]: Test User
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  Oracle
What is the name of your City or Locality?
  [Unknown]:  Redwood Shores
What is the name of your State or Province?
  [Unknown]:  CA
```

```
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB> correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Note: To determine your two-letter country code, use the ISO country code list at the following URL:

`http://www.bcpl.net/~jspath/isocodes.html.`

The `mykeystore` file is created in the current directory. The default alias of the key is `mykey`.

Requesting Client Authentication

OC4J supports a *client authentication* mode in which the server explicitly requests authentication from the client before the server will communicate with the client. In an OracleAS environment, Oracle HTTP Server acts as the client to OC4J.

For client authentication, Oracle HTTP Server must have its own certificate and authenticate itself by sending a certificate and a certificate chain that ends with a root certificate. OC4J can be configured to accept only root certificates from a specified list in establishing a chain of trust back to a client.

A certificate that OC4J trusts is called a *trust point*. In the certificate chain from Oracle HTTP Server, the trust point is the first certificate that OC4J encounters that matches one in its own keystore. There are three ways to establish trust:

- The client certificate is in the keystore.
- One of the intermediate CA certificates in the certificate chain from Oracle HTTP Server is in the keystore.
- The root CA certificate in the certificate chain from Oracle HTTP Server is in the keystore.

OC4J verifies that the entire certificate chain up to and including the trust point is valid to prevent any forged certificates.

If you request client authentication with the `needs-client-auth` attribute, perform the following steps. See "OC4J Configuration Steps for SSL" on page 11-23 for how to configure this attribute.

1. Decide which of the certificates in the chain from Oracle HTTP Server is to be your trust point. Ensure that you either have control over the issuance of certificates using this trust point or that you trust the certificate authority as an issuer.
2. Import the intermediate or root certificate in the server keystore as a trust point for authentication of the client certificate.

> **Note:** If you do not want OC4J to accept certain trust points, make sure these trust points are not in the keystore.

3. Execute the steps to create the client certificate (documented in "Creating Keys and Certificates With OC4J and Oracle HTTP Server" on page 11-4). The client certificate includes the intermediate or root certificate that is installed in the server. If you wish to trust another certificate authority, obtain a certificate from that authority.
4. Save the certificate in a file on Oracle HTTP Server.
5. Provide the certificate for the Oracle HTTP Server initiation of the secure AJP connection.

During secure communication between the client and OC4J, the following functionality is executed:

- The link (all communications) between the two is encrypted.
- OC4J is authenticated to the client. A "secret key" is securely exchanged and used for the encryption of the link.
- Optionally, if OC4J is in client-authentication mode, the client is authenticated to OC4J.

See Also:

- Oracle Application Server 10g Security Guide *e* and *Servlet Developer's Guide* for information about Oracle Wallet Manager, PKI, and security fundamentals.
- Documentation for JSSE and the `java.net` packages at `http://www.java.sun.com`

Oracle HTTPS And Clients

HTTPS is vital to securing client-server interactions. For many server applications, HTTPS is handled by the Web server. However, any application that acts as a client, such as servlets that initiate connections to other Web servers, needs its own HTTPS implementation to make requests and to receive information securely from the server. Java application developers who are familiar with either the HTTP package, `HTTPClient`, or who are familiar with the Sun Microsystems, Inc., `java.net` package can easily use Oracle HTTPS to secure client interactions with a server.

Oracle HTTPS extends the `HTTPConnection` class of the `HTTPClient` package, which provides a complete HTTP client library. To support client HTTPS connections, several methods have been added to the `HTTPConnection` class that use the OracleSSL class, `OracleSSLCredential`.

Note: Oracle `HTTPClient` supports two different SSL implementations: the Java Secure Socket Extension (JSSE) and OracleSSL. This documentation discusses the two implementations separately.

HTTPConnection Class

The `HTTPConnection` class is used to create new connections that use HTTP, with or without SSL. To provide support for PKI (Public Key Infrastructure) digital certificates and wallets, the methods described in "Oracle HTTPS Example" on page 11-17 have been added to this class.

See Also: The `HTTPClient` Javadoc.

OracleSSLCredential Class (OracleSSL Only)

Security credentials are used to authenticate the server and the client to each other. Oracle HTTPS uses the Oracle Java SSL package, `OracleSSLCredential`, to load user certificates and trustpoints from base64 or DER-encoded certificates. (DER, part of the X.690 ASN.1 standard, stands for Distinguished Encoding Rules.)

The API for Oracle Java SSL requires that security credentials be passed to the HTTP connection before the connection is established. The `OracleSSLCredential` class is used to store these security credentials. Typically, a wallet generated by Oracle Wallet Manager is used to populate the `OracleSSLCredential` object. Alternatively, individual certificates can be added by using an `OracleSSLCredential` class API. After the credentials are complete, they are passed to the connection with the `setCredentials` method.

Overview of Oracle HTTPS Features

Oracle HTTPS supports HTTP 1.0 and HTTP 1.1 connections between a client and a server. To provide SSL functionality, new methods have been added to the `HTTPConnection` class of this package. These methods are used in conjunction with Oracle Java SSL to support cipher suite selection, security credential management with Oracle Wallet Manager, security-aware applications, and other features that are described in the following sections. Oracle HTTPS uses the Oracle Java SSL class, `OracleSSLCredential`, and it extends the `HTTPConnection` class of the `HTTPClient` package. `HTTPClient` supports two SSL implementations, OracleSSL and JSSE.

In addition to the functionality included in the `HTTPClient` package, Oracle HTTPS supports the following:

- Multiple cryptographic algorithms
- Certificate and key management with Oracle Wallet Manager
- Limited support for the `java.net.URL` framework
- Both the OracleSSL and JSSE SSL implementations

In addition, Oracle HTTPS uses the `HTTPClient` package to support

- HTTP tunneling through proxies
- HTTP proxy authentication

The following sections describe Oracle HTTPS features in detail:

- SSL Cipher Suites
- SSL Cipher Suites Supported by OracleSSL
- SSL Cipher Suites Supported by JSSE
- Security-Aware Applications Support
- java.net.URL Framework Support

SSL Cipher Suites

Before data can flow through an SSL connection, both sides of the connection must negotiate common algorithms to be used for data transmission. A set of such algorithms combined to provide a mix of security features is called a *cipher suite*. Selecting a particular cipher suite lets the participants in an SSL connection establish the appropriate level for their communications.

`HTTPClient` supports two different SSL implementations, each of which supports different cipher suites. These are discussed below.

Choosing a Cipher Suite

In general, you should prefer:

- RSA to Diffie-Hellman, because RSA defeats many security attacks.
- 3DES or RC4 128 to other encryption methods, because 3DES and RC4 128 have strong keys
- SHA1 digest to MD5, because SHA1 produces a stronger digest.

SSL Cipher Suites Supported by OracleSSL

OracleSSL supports the cipher suites listed in Table 11–1. Note that with NULL encryption, SSL is only used for authentication and data integrity purposes.

Table 11–1 Cipher Suites Supported By OracleSSL

Cipher Suite	Authentication	Encryption	Hash Function (Digest)
`SSL_RSA_WITH_3DES_EDE_CBC_SHA`	RSA	3DES EDE CBC	SHA1
`SSL_RSA_WITH_RC4_128_SHA`	RSA	RC4 128	SHA1
`SSL_RSA_WITH_RC4_128_MD5`	RSA	RC4 128	MD5
`SSL_RSA_WITH_DES_CBC_SHA`	RSA	DES CBC	SHA1
`SSL_RSA_EXPORT_WITH_RC4_40_MD5`	RSA	RC4 40	MD5
`SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`	RSA	DES40 CBC	SHA1
`SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`	DH anon	3DES EDE CBC	SHA1
`SSL_DH_anon_WITH_RC4_128_MD5`	DH anon	RC4 128	MD5
`SSL_DH_anon_WITH_DES_CBC_SHA`	DH anon	DES CBC	SHA1
`SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`	DH anon	RC4 40	MD5
`SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`	DH anon	DES40 CBC	SHA1
`SSL_RSA_WITH_NULL_SHA`	RSA	NULL	SHA1
`SSL_RSA_WITH_NULL_MD5`	RSA	NULL	MD5

SSL Cipher Suites Supported by JSSE

JSSE supports the cipher suites listed in Table 11–1. Note that with NULL encryption, SSL is only used for authentication and data integrity purposes.

Table 11–2 Cipher Suites Supported By JSSE

Cipher Suite	Authentication	Encryption	Hash Function (Digest)
`SSL_RSA_WITH_3DES_EDE_CBC_SHA`	RSA	3DES EDE CBC	SHA1
`SSL_RSA_WITH_RC4_128_SHA`	RSA	RC4 128	SHA1
`SSL_RSA_WITH_RC4_128_MD5`	RSA	RC4 128	MD5
`SSL_RSA_WITH_DES_CBC_SHA`	RSA	DES CBC	SHA1
`SSL_RSA_EXPORT_WITH_RC4_40_MD5`	RSA	RC4 40	MD5
`SSL_DH_anon_WITH_3DES_EDE_CBC_SHA`	DH anon	3DES EDE CBC	SHA1
`SSL_DH_anon_WITH_RC4_128_MD5`	DH anon	RC4 128	MD5
`SSL_DH_anon_WITH_DES_CBC_SHA`	DH anon	DES CBC	SHA1
`SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`	DH anon	RC4 40	MD5
`SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`	DH anon	DES40 CBC	SHA1
`SSL_RSA_WITH_NULL_SHA`	RSA	NULL	SHA1
`SSL_RSA_WITH_NULL_MD5`	RSA	NULL	MD5
`SSL_DHE_DSS_WITH_DES_CBC_SHA`	DH	DES CBC	SHA1
`SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`	DH	3DES EDE CBC	SHA1
`SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`	DH	DES40 CBC	SHA1

Access Information About Established SSL Connections

Users can access information about established SSL connections using the `getSSLSession` method of Oracle HTTPS. After a connection is established, users can retrieve the cipher suite used for the connection, the peer certificate chain, and other information about the current connection.

Security-Aware Applications Support

Oracle HTTPS uses Oracle Java SSL to provide security-aware applications support. When security-aware applications do not set trust points, Oracle Java SSL allows them to perform their own validation letting the handshake complete successfully only if a complete certificate chain is sent by the peer. When applications authenticate to the trustpoint level, they are responsible for authenticating individual certificates below the trustpoint.

After the handshake is complete, the application must obtain the SSL session information and perform any additional validation for the connection.

Security-unaware applications that need the trust point check must ensure that trust points are set in the HTTPS infrastructure.

See Also: *Oracle Advanced Security Administrator's Guide* for information about Oracle Java SSL.

java.net.URL Framework Support

The `HTTPClient` package provides basic support for the `java.net.URL` framework with the `HTTPClient.HttpUrlConnection` class. However, many of the Oracle HTTPS features are supported through system properties only.

Features that are only supported through system properties are

- cipher suites selection option
- confidentiality only option
- server authentication option
- mutual authentication option
- security credential management with Oracle Wallet Manager

> **Note:** If the `java.net.URL` framework is used, then set the `java.protocol.handler.pkgs` system property to select the `HTTPClient` package as a replacement for the JDK client as follows:
>
> ```
> java.protocol.handler.pkgs=HTTPClient
> ```

> **See Also:**
>
> - "Specifying Default System Properties" on page 11-15 for information about configuring your client to use JSSE.
> - Documentation for the `java.net.URL` framework at
>
>   `http://java.sun.com`

Specifying Default System Properties

For many users of HTTPS it is desirable to specify some default properties in a non-programmatic way. The best way to accomplish this is through Java system properties which are accessible through the `java.lang.System` class. These properties are the only way for users of the `java.net.URL` framework to set security credential information. Oracle HTTPS recognizes the following properties:

- javax.net.ssl.KeyStore
- javax.net.ssl.KeyStorePassword
- Oracle.ssl.defaultCipherSuites (OracleSSL only)

The following sections describe how to set these properties.

javax.net.ssl.KeyStore

This property can be set to point to the text wallet file exported from Oracle Wallet Manager that contains the credentials that are to be used for a specific connection. For example:

```
javax.net.ssl.KeyStore=/etc/ORACLE/WALLETS/Default/default.txt
```

where *`default.txt`* is the name of the text wallet file that contains the credentials.

If no other credentials have been set for the HTTPS connection, then the file indicated by this property is opened when a handshake first occurs. If any errors occur while reading this file, then the connection fails and an `IOException` is thrown.

If you do not set this property, the application is responsible for verifying that the certificate chain contains a certificate that can be trusted. However, `HTTPClient` using Oracle SSL does verify that all of the certificates in the certificate chain, from the user certificate to the root CA, have been sent by the server and that all of these certificates contain valid signatures.

javax.net.ssl.KeyStorePassword

This property can be set to the password that is necessary to open the wallet file. For example:

```
javax.net.ssl.KeyStorePassword=welcome1
```

where *`welcome1`* is the password that is necessary to open the wallet file.

Potential Security Risk with Storing Passwords in System Properties

Storing the wallet file password as a Java system property can result in a security risk in some environments. To avoid this risk, use one of the following alternatives:

- If mutual authentication is not required for the application, use a text wallet that contains no private key. No password is needed to open these wallets.
- If a password is necessary, then do not store it in a clear text file. Instead, load the property dynamically before the `HTTPConnection` is started by using `System.setProperty()`. Unset the property after the handshake is completed.

Oracle.ssl.defaultCipherSuites (OracleSSL only)

This property can be set to a comma-delimited list of cipher suites. For example:

```
Oracle.ssl.defaultCipherSuites=
            SSL_RSA_WITH_DES_CBC_SHA,\
            SSL_RSA_EXPORT_WITH_RC4_40_MD5,\
            SSL_RSA_WITH_RC4_128_MD5
```

The cipher suites that you set this property to are used as the default cipher suites for new HTTPS connections.

> **See Also:** Table 11–1 on page 11-12 for a complete list of the cipher suites that are supported by OracleSSL.

Oracle HTTPS Example

The following is a simple program that uses Oracle HTTPS, `HTTPClient`, and OracleSSL to connect to a Web server, send a `GET` request, and fetch a Web page. The complete code for this program is presented here followed by sections that explain how Oracle HTTPS is used to set up secure connections.

```
import HTTPClient.HTTPConnection;
import HTTPClient.HTTPResponse;
import oracle.security.ssl.OracleSSLCredential;
import java.io.IOException;

public class HTTPSConnectionExample
{
    public static void main(String[] args)
    {
        if(args.length < 4)
        {
            System.out.println(
            "Usage: java HTTPSConnectionTest [host] [port] " +
            "[wallet] [password]");
            System.exit(-1);
        }

        String hostname = args[0].toLowerCase();
        int port = Integer.decode(args[1]).intValue();
        String walletPath = args[2];
        String password = args[3];

        HTTPConnection httpsConnection = null;
        OracleSSLCredential credential = null;

        try
```

```
{
    httpsConnection = new HTTPConnection("https", hostname, port);
}
catch(IOException e)
{
    System.out.println("HTTPS Protocol not supported");
    System.exit(-1);
}

try
{
    credential = new OracleSSLCredential();
    credential.setWallet(walletPath, password);
}
catch(IOException e)
{
    System.out.println("Could not open wallet");
    System.exit(-1);
}
httpsConnection.setSSLCredential(credential);

try
{
    httpsConnection.connect();
}
catch (IOException e)
{
    System.out.println("Could not establish connection");
    e.printStackTrace();
    System.exit(-1);
}

javax.security.cert.X509Certificate[] peerCerts = null;
try
{
    peerCerts =
      (httpsConnection.getSSLSession()).getPeerCertificateChain();
}
catch(javax.net.ssl.SSLPeerUnverifiedException e)
{
    System.err.println("Unable to obtain peer credentials");
    System.exit(-1);
}

String peerCertDN =
                peerCerts[peerCerts.length -1].getSubjectDN().getName();
            peerCertDN = peerCertDN.toLowerCase();
            if(peerCertDN.lastIndexOf("cn="+hostname) == -1)
            {
                System.out.println("Certificate for " + hostname + " is issued to 
"
                    + peerCertDN);
                System.out.println("Aborting connection");
                System.exit(-1);
            }

            try
            {
                HTTPResponse rsp = httpsConnection.Get("/");
                System.out.println("Server Response: ");
                System.out.println(rsp);
            }
            catch(Exception e)
            {
                System.out.println("Exception occured during Get");
                e.printStackTrace();
                System.exit(-1);
            }
        }
}
```

Initializing SSL Credentials In OracleSSL

This program example uses a wallet created by Oracle Wallet Manager to set up credential information. First the credentials are created and the wallet is loaded using

```
credential = new OracleSSLCredential();
credential.setWallet(walletPath, password);
```

After the credentials are created, they are passed to `HTTPSConnection` using

```
httpsConnection.setSSLCredential(credential);
```

The private key, user certificate, and trust points located in the wallet can now be used for the connection.

Verifying Connection Information

Although SSL verifies that the certificate chain presented by the server is valid and contains at least one certificate trusted by the client, that does not prevent impersonation by malicious third parties. An HTTPS standard that addresses this problem requires that HTTPS servers have certificates issued to their host name. Then it is the responsibility of the client to perform this validation after the SSL connection is established.

To perform this validation in this sample program, `HTTPSConnectionExample` establishes a connection to the server without transferring any data using the following:

```
httpsConnection.connect();
```

After the connection is established, the connection information, in this case the server certificate chain, is obtained with the following:

```
peerCerts = (httpsConnection.getSSLSession()).getPeerCertificateChain();
```

Finally the server certificate's common name is obtained with the following:

```
String peerCertDN = peerCerts[peerCerts.length -1].getSubjectDN().getName();
peerCertDN = peerCertDN.toLowerCase();
```

If the certificate name is not the same as the host name used to connect to the server, then the connection is aborted with the following:

```
if(peerCertDN.lastIndexOf("cn="+hostname) == -1)
{
    System.out.println("Certificate for " + hostname + " is issued to " +
        peerCertDN);
    System.out.println("Aborting connection");
        System.exit(-1);
}
```

Transferring Data Using HTTPS

It is important to verify the connection information before data is transferred from the client or from the server. The data transfer is performed in the same way for HTTPS as it is for HTTP. In this sample program a `GET` request is made to the server using the following:

```
HTTPResponse rsp = httpsConnection.Get("/");
```

Using HTTPClient with JSSE

OracleAS supports HTTPS client connections using the Java Secure Socket Extension (JSSE). A client can configure `HTTPClient` to use JSSE as the underlying SSL provider.

Notes:

- The JSSE SSL implementation is not thread-safe; if you need to use SSL in a threaded application, use OracleSSL.
- For full information on JSSE, see the Sun documentation at `http://java.sun.com/products/jsse/`.

`HTTPClient` still uses OracleSSL as the default provider, but the developer can easily change this by setting the `SSLSocketFactory` on the `HTTPConnection` class. This following code snippet demonstrates how a client could configure HTTPClient to use JSSE for SSL communication.

```
public void obtainHTTPSConnectionUsingJSSE() throws Exception
{
// set the trust store to the location of the client's trust store file
      // this value specifies the certificate authorities the client accepts
   System.setProperty("javax.net.ssl.trustStore", KEYSTORE_FILE);
   // creates the HTTPS URL
   URL testURL = new URL("https://" + HOSTNAME + ":" + HTTPS_PORTNUM);
   // call SSLSocketFactory.getDefault() to obtain the default JSSE
implementation
  // of an SSLSocketFactory
   SSLSocketFactory socketFactory =
(SSLSocketFactory)SSLSocketFactory.getDefault();
   HTTPConnection connection = new HTTPConnection(testURL);

   // configure HTTPClient to use JSSE as the underlying
   // SSL provider
   connection.setSSLSocketFactory(socketFactory);
   // call connect to setup SSL handshake
   try
   {
       connection.connect();
   }
   catch (IOException e)
   {
       e.printStackTrace();    }

   HTTPResponse response = connection.Get("/index.html");
}
```

Configuring HTTPClient To Use JSSE

The steps required to use JSSE with `HTTPClient` are as follows:

1. Create a truststore using the keytool.

Notes: ■For details of using the keytool, see `http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html`J

- JSSE's implementation of SSL has some subtle differences from Oracle's implementation. Unlike in OracleSSL, if no truststore is set, the JDK default truststore will be used. This default will accept known certificate authorities, such as Verisign and Thawte. Many self-signed certificates will be rejected by this default.

---

2. Set the truststore property. A client wishing to use JSSE must specify the client truststore location in `javax.net.ssl.trustStore`. Unlike OracleSSL, the client does not need to set the `javax.net.ssl.keyStore` property.
3. Obtain the JSSE `SSLSocketFactory` by calling `SSLSocketFactory.getDefault().`
4. Create an `HTTPClient` connection.
5. Configure the `HTTPClient` connection to use the JSSE implementation of SSL. `HTTPClient` can be configured to use JSSE in one of two ways:
    1. (**Per-connection**) The client calls `HTTPConnection.setSSLSocketFactory(SSLSocketFactory factory)`
    2. (E**ntire VM**) The client calls the static method: `HttpConnection.setDefaultSSLSocketFactory(SSLSocketFactory factory).` This static method must be called before instantiating any `HTTPConnection` instances.
6. Call `HTTPConnection.connect()` before sending any HTTPS data. This allows the connection to verify the SSL handshaking that must occur between client and server before any data can be encrypted and sent.

7. Use the `HTTPConnection` instance normally. At this point, the client is set up to use `HTTPClient` with JSSE. There is no additional configuration necessary and basic usage is the same.

Configuring Oracle HTTP Server and OC4J for SSL

For secure communication between Oracle HTTP Server and OC4J, configuration steps are required at each end, as detailed in the following sections:

- Oracle HTTP Server Configuration Steps for SSL
- OC4J Configuration Steps for SSL

Oracle HTTP Server Configuration Steps for SSL

In Oracle HTTP Server, verify proper SSL settings in `mod_oc4j.conf` for secure communication. SSL must be enabled, with a wallet file and password specified, as follows:

```
Oc4jEnableSSL on
Oc4jSSLWalletFile wallet_path
Oc4jSSLWalletPassword pwd
```

The *`wallet_path`* value is a directory path to the wallet file, without a file name. (The wallet file name is already known.) The *`pwd`* value is the wallet password.

For more information about the `mod_oc4j.conf` file, see *Oracle HTTP Server Administrator's Guide*.

OC4J Configuration Steps for SSL

In the `default-web-site.xml` file (or other Web site XML file, as appropriate), you must specify appropriate SSL settings under the `<web-site>` element.

1. Turn on the `secure` flag to specify secure communication, as follows:

   ```
   <web-site ... secure="true" ... >
      ...
   </web-site>
   ```

   Setting `secure="true"` specifies that the AJP protocol should use an SSL socket.

2. Use the `<ssl-config>` sub-element and its `keystore` and `keystore-password` attributes to specify the path and password for the keystore, as follows:

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd" />
   </web-site>
   ```

   The `<ssl-config>` element is required whenever the `secure` flag is set to `"true"`.

   The *`path_and_file`* value can indicate either an absolute or relative directory path and includes the file name. A relative path is relative to the location of the Web site XML file.

3. Optionally, to specify that client authentication is required, turn on the `needs-client-auth` flag. This is an attribute of the `<ssl-config>` element.

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd"
                  needs-client-auth="true" />
   </web-site>
   ```

   This sets up a mode where OC4J will accept or reject a client entity, such as Oracle HTTP Server, for secure communication depending on its identity. The `needs-client-auth` flag instructs OC4J to request the client certificate chain upon connection. If OC4J recognizes the root certificate of the client, then the client is accepted.

   The keystore that is specified in the `<ssl-config>` element must contain the certificates of any clients that are authorized to connect to OC4J through secure AJP and SSL.

Here is an example that sets up secure communication with client authentication:

```
<web-site display-name="OC4J Web Site" protocol="ajp13" secure="true" >
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
               needs-client-auth="true" />
</web-site>
```

Only the portions in bold are specific to security. The protocol value is always `"ajp13"` for communication through Oracle HTTP Server, whether or not you use secure communication. A protocol value of `ajp13` with `secure="false"` indicates AJP protocol, while `ajp13` with `secure="true"` indicates secure AJP protocol.

For more information about elements and attributes of the `<web-site>` and `<ssl-config>` elements, see the *Oracle Application Server Containers for J2EE Servlet Developer's Guide.*

Configuring OC4J Standalone for SSL

For secure communication between a client and OC4J, configuration is required on OC4J standalone. You are required to provide a certificate on the client-side only if you configure client-authentication.

In the `default-web-site.xml` file of OC4J (or other Web site XML file, as appropriate), you must specify appropriate SSL settings under the `<web-site>` element.

1. Turn on the `secure` flag to specify secure communication, as follows:

   ```
   <web-site ... protocol="http" secure="true" ... >
      ...
   </web-site>
   ```

   Setting `secure="true"` specifies that the HTTP protocol is to use an SSL socket.

2. Use the `<ssl-config>` sub-element and its `keystore` and `keystore-password` attributes to specify the directory path and password for the keystore, as follows:

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd" />
   </web-site>
   ```

   The `<ssl-config>` element is required whenever the `secure` flag is set to `"true"`.

   The *`path_and_file`* value can indicate either an absolute or relative directory path and includes the file name.

Note: You can hide the password through password indirection. See Oracle Application Server Containers for J2EE Security Guide for a description of password indirection.

---

3. Optionally, turn on the `needs-client-auth` flag, an attribute of the `<ssl-config>` element, to specify that client authentication is required, as follows:

   ```
   <web-site ... secure="true" ... >
      ...
      <ssl-config keystore="path_and_file" keystore-password="pwd"
                  needs-client-auth="true" />
   </web-site>
   ```

   This step sets up a mode where OC4J accepts or rejects a client entity for secure communication, depending on its identity. The `needs-client-auth` attribute instructs OC4J to request the client certificate chain upon connection. If the root certificate of the client is recognized, then the client is accepted.

   The keystore specified in the `<ssl-config>` element must contain the certificates of any clients that are authorized to connect to OC4J through HTTPS.

4. Optionally, specify each application in the Web site as shared. The `shared` attribute of the `<web-app>` element indicates whether multiple bindings (different Web sites, or ports, and context roots) can be shared. Supported values are `"true"` and `"false"` (default).

   Sharing implies the sharing of everything that makes up a Web application, including sessions, servlet instances, and context values. A typical use for this mode is to share a Web application between an HTTP site and an HTTPS site at the same context path, when SSL is required for some but not all of the communications. Performance is improved by encrypting only sensitive information, rather than all information.

   If an HTTPS Web application is marked as shared, then instead of using the SSL certificate to track the session, the cookie is used to track the session. This is beneficial in that the SSL certificate uses 50K to store each certificate when tracking it, which sometimes results in an "out of memory" problem for the session before the session times out. This could possibly make the Web application less secure, but might be necessary to work around issues such as SSL session timeouts not being properly supported in some browsers.

5. Optionally, set the cookie domain if `shared` is true and the default ports are not used. When the client interacts with a Web server over separate ports, the cookie believes that each separate port denotes a separate Web site. If you use the default ports of 80 for HTTP and 443 for HTTPS, the client recognizes these as two different ports of the same Web site and creates only a single cookie. However, if you use non-default ports, the client does not recognize these ports as part of the same Web site and will create separate cookies for each port, unless you specify the cookie domain.

   Cookie domains track the client's communication across multiple servers within a DNS domain. If you use non-default ports for a shared environment with HTTP and HTTPS, set the `cookie-domain` attribute in the `<session-tracking>` element in the `orion-web.xml` file for the application. The cookie-domain attribute contains the DNS domain with at least two components of the domain name provided.

   ```
   <session-tracking cookie-domain=".oracle.com" />
   ```

Example 11–1 HTTPS Communication With Client Authentication

The following configures a Web site for HTTPS secure communication with client authentication:

```
<web-site display-name="OC4J Web Site" protocol="http" secure="true" >
   <default-web-app application="default" name="defaultWebApp" />
   <access-log path="../log/default-web-access.log" />
   <ssl-config keystore="../keystore" keystore-password="welcome"
              needs-client-auth="true" />
</web-site>
```

Only the portions in bold are specific to security. The protocol value is always `"http"` for HTTP communication, whether or not you use secure communication. A protocol value of `http` with `secure="false"` indicates HTTP protocol; `http` with `secure="true"` indicates HTTPS protocol.

Then, configures the news application to accept both HTTP and HTTPS connections:

```
<web-app application="news" name="news-web" root="/news" shared="true" />
```

This Web site uses the default port numbers for HTTP and HTTPS communication. If it did not, you would also add the `cookie-domain` attribute.

```
<session-tracking cookie-domain=".oracle.com" />
```

For more information about elements and attributes of the `<web-site>`, `<web-app>`, and `<session-tracking>` elements, see the XML Appendix in the *Oracle Application Server Containers for J2EE Servlet Developer's Guide.*

Example 11–2 Creating an SSL Certificate and Configuring HTTPS

The following example uses `keytool` to create a test certificate and shows all of the XML configuration necessary for HTTPS to work. To create a valid certificate for use in production environments, see the `keytool` documentation.

1. Install the correct JDK

   Ensure that JDK 1.3.x is installed. This is required for SSL with OC4J. Set the JAVA_HOME to the JDK 1.3 directory. Ensure that the JDK 1.3.x `JAVA_HOME/bin` is at the beginning of your path. This may be achieved by doing the following:

   <u>UNIX</u>

   ```
   $ PATH=/usr/opt/java130/bin:$PATH
   $ export $PATH
   $ java -version
   java version "1.3.0"
   ```

   <u>Windows</u>

   ```
   set PATH=d:\jdk131\bin;%PATH%
   ```

   Ensure that this JDK version is set as the current version in your Windows registry. In the Windows Registry Editor under `HKEY_LOCAL_MACHINE/SOFTWARE/JavaSoft/Java Development Kit`, set '`CurrentVersion`' to 1.3 (or later).

2. Request a certificate

   a. Change directory to `ORACLE_HOME/j2ee`

   b. Create a keystore with an RSA private/public keypair using the `keytool` command. In our example, we generate a keystore to reside in a file named '`mykeystore`', which has a password of '`123456`' and is valid for 21 days, using the 'RSA' key pair generation algorithm with the following syntax:

   ```
   keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456
   -validity 21
   ```

   Where:

- the `keystore` option sets the filename where the keys are stored
- the `storepass` option sets the password for protecting the keystore
- the `validity` option sets number of days the certificate is valid

The `keytool` prompts you for more information, as follows:

```
keytool -genkey -keyalg "RSA" -keystore mykeystore -storepass 123456
-validity 21

What is your first and last name?
  [Unknown]:  Test User
What is the name of your organizational unit?
  [Unknown]:  Support
What is the name of your organization?
  [Unknown]:  Oracle
What is the name of your City or Locality?
  [Unknown]:  Redwood Shores
What is the name of your State or Province?
  [Unknown]:  CA
What is the two-letter country code for this unit?
  [Unknown]:  US
Is <CN=Test User, OU=Support, O=Oracle, L=Reading, ST=Berkshire, C=GB>
correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Note: To determine your 'two-letter country code', use the ISO country code list at the following URL: `http://www.bcpl.net/~jspath/isocodes.html.`

The `mykeystore` file is created in the current directory. The default alias of the key is `mykey`.

3. If you do not have a `secure-web-site.xml` file, then copy the `default-web-site.xml` to `$ORACLE_HOME/j2ee/home/config/secure-web-site.xml.`
4. Edit `secure-web-site.xml` with the following elements:
   a. Add `secure="true"` to the `<web-site>` element, as follows:

```
<web-site port="8888" display-name="Default OracleAS Containers for J2EE
```

```
Web Site" secure="true">
```

b. Add the following new line inside the `<web-site>` element to define the keystore and the password.

```
<ssl-config keystore="<Your-Keystore>"
keystore-password="<Your-Password>" />
```

Where `<Your-Keystore>` is the full path to the keystore and `<Your-Password>` is the keystore password. In our example, this is as follows:

```
<!-- Enable SSL -->
<ssl-config keystore="../../keystore" keystore-password="123456"/>
```

Note: The keystore path is relative to where the XML file resides.

---

c. Change the web-site port number, to use an available port. For example, the default for SSL ports is 443, so change the Web site port attribute to port="4443". To use the default of 443, you have to be a super user.

d. Now save the changes to `secure-web-site.xml`.

5. If you did not have the `secure-web-site.xml` file, then edit `server.xml` to point to the `secure-web-site.xml` file.

a. Uncomment or add the following line in the file `server.xml` so that the `secure-web-site`.xml file is read.

```
<web-site path="./secure-web-site.xml" />
```

Note: Even on Windows, you use a forward slash and not a back slash in the XML files.

---

b. Save the changes to `server.xml`.

6. Stop and re-start OC4J to initialize the secure-web-site.xml file additions. Test the SSL port by accessing the site in a browser on the SSL port. If successful, you will be asked to accept the certificate, because it is not signed by an accepted authority.

When completed, OC4J listens for SSL requests on one port and non-SSL requests on another. You can disable either SSL requests or non-SSL requests, by commenting out the appropriate `*web-site.xml` in the `server.xml` configuration file.

```
<web-site path="./secure-web-site.xml" /> - comment out this to remove SSL
<default-site path="./default-web-site.xml" /> - comment out this to
      remove non-SSL
```

Requesting Client Authentication with OC4J Standalone

OC4J supports a "client-authentication" mode in which the server explicitly requests authentication from the client before the server will communicate with the client. In this case, the client must have its own certificate. The client authenticates itself by sending a certificate and a certificate chain that ends with a root certificate. OC4J can be configured to accept only root certificates from a specified list in establishing a chain of trust back to the client.

A certificate that OC4J trusts is called a trust point. This is the first certificate that OC4J encounters in the chain from the client that matches one in its own keystore. There are three ways to configure trust:

- The client certificate is in the keystore.
- One of the intermediate certificate authority certificates in the client's chain is in the keystore.
- The root certificate authority certificate in the client's chain is in the keystore.

OC4J verifies that the entire certificate chain up to and including the trust point is valid to prevent any forged certificates.

If you request client authentication with the `needs-client-auth` attribute, perform the following:

1. Decide which of the certificates in the client's chain is to be your trust point. Ensure that you either have control of the issue of certificates using this trust point or that you trust the certificate authority as an issuer.
2. Import the intermediate or root certificate in the server keystore as a trust point for authentication of the client certificate.
3. If you do not want OC4J to have access to certain trust points, make sure that these trust points are not in the keystore.
4. Execute the preceding steps to create the client certificate, which includes the intermediate or root certificate installed in the server. If you wish to trust another certificate authority, obtain a certificate from that authority.

5. Save the certificate in a file on the client.
6. Provide the certificate on the client initiation of the HTTPS connection.
   a. If the client is a browser, set the certificate in the client browser security area.
   b. If the client is a Java client, you must programmatically present the client certificate and the certificate chain when initiating the HTTPS connection.

HTTPS Common Problems and Solutions

The following errors may occur when using SSL certificates:

Keytool Error: java.security.cert.CertificateException: Unsupported encoding

Cause: You cannot allow trailing whitespace in the keytool.

Action: Delete all trailing whitespace. If the error still occurs, add a new line in your certificate reply file.

Keytool Error: KeyPairGenerator not available

Cause: You are probably using a keytool from an older JDK.

Action: Use the keytool from the latest JDK on your system. To ensure that you are using the latest JDK, specify the full path for this JDK.

Keytool Error: Failed to establish chain from reply

Cause: The keytool cannot locate the root CA certificates in your keystore; thus, the keytool cannot build the certificate chain from your server key to the trusted root certificate authority.

Action: Execute the following:

```
keytool -keystore keystore -import -alias cacert -file cacert.cer (keytool
-keystore keystore -import -alias intercert -file inter.cer)
```

If you use an intermediate CA keytool, then execute the following:

```
keystore keystore -genkey -keyalg RSA -alias serverkey keytool -keystore
keystore -certreq -file my.host.com.csr
```

Get the certificate from the Certificate Signing Request, then execute the following:

```
keytool -keystore keystore -import -file my.host.com.cer -alias serverkey
```

No available certificate corresponds to the SSL cipher suites which are enabled

Cause: Something is wrong with your certificate.

IllegalArgumentException: Mixing secure and non-secure sites on the same ip + port

Cause: You cannot configure SSL and non-SSL web-sites to listen on the same port and IP address.

Action: Check to see that different ports are assigned within `secure-web-site.xml` and `default-web-site.xml` files.

Keytool does not work on HP-UX

Cause: On HP-UX, it has been reported that the 'keytool' does not work with the RSA option.

Action: Generate the key on another platform and FTP it to the HP-UX server.

12 EJB Security

This chapter discusses security issues affecting EJBs. It discusses the following topics:

- EJB JNDI Security Properties
- Configuring Security

For full information about EJBs, see the *Oracle Application Server Containers for J2EE Enterprise JavaBeans Developer's Guide.*

EJB JNDI Security Properties

There are two JNDI properties that are specific to security. You can either set these within the jndi.properties file or within your EJB implementation.

JNDI Properties in jndi.properties

If setting the JNDI properties within the `jndi.properties` file, set the properties as follows. Make sure that this `jndi.properties` file is accessible from the `CLASSPATH`.

When you access EJBs in a *remote* container, you must pass valid credentials to this container. Stand-alone clients define their credentials in the `jndi.properties` file deployed with the client's code.

```
java.naming.security.principal=<username>
java.naming.security.credentials=<password>
```

JNDI Properties Within Implementation

Set the properties with the same values, only with different syntax. For example, JavaBeans running within the container pass their credentials within the `InitialContext`, which is created to look up the remote EJBs.

For instance, to pass JNDI security properties within the `Hashtable` environment, set these as shown below:

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
      "com.evermind.server.ApplicationClientInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "guest");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
Context ic = new InitialContext (env);
Object homeObject = ic.lookup("java:comp/env/employeeBean");

// Narrow the reference to a TemplateHome.
EmployeeHome empHome =
   (EmployeeHome) PortableRemoteObject.narrow(homeObject,
   EmployeeHome.class);
```

Configuring Security

EJB security involves two realms: granting permissions if you download into a browser and configuring your application for authentication and authorization. This section covers the following:

- Granting Permissions in Browser
- Authenticating and Authorizing EJB Applications
- Specifying Credentials in EJB Clients

Granting Permissions in Browser

If you download the EJB application as a client where the security manager is active, you must grant the following permissions before you can execute:

```
permission java.net.SocketPermission "*:*", "connect,resolve";
permission java.lang.RuntimePermission "createClassLoader";
permission java.lang.RuntimePermission "getClassLoader";
permission java.util.PropertyPermission "*", "read";
permission java.util.PropertyPermission "LoadBalanceOnLookup",
  "read,write";
```

Authenticating and Authorizing EJB Applications

For EJB authentication and authorization, you define the principals under which each method executes by configuring of the EJB deployment descriptor. The container enforces that the user who is trying to execute the method is the same as defined within the deployment descriptor.

The EJB deployment descriptor enables you to define security roles under which each method is allowed to execute. These methods are mapped to users or groups in the OC4J-specific deployment descriptor. The users and groups are defined within your designated security user managers, which uses either the JAZN or XML user manager. For a full description of security user managers, see the *Oracle Application Server Containers for J2EE User's Guide* and *Oracle Application Server Containers for J2EE Services Guide.*

For authentication and authorization, this section focuses on XML configuration within the EJB deployment descriptors. EJB authorization is specified within the EJB and OC4J-specific deployment descriptors. You can manage the authorization piece of your security within the deployment descriptors, as follows:

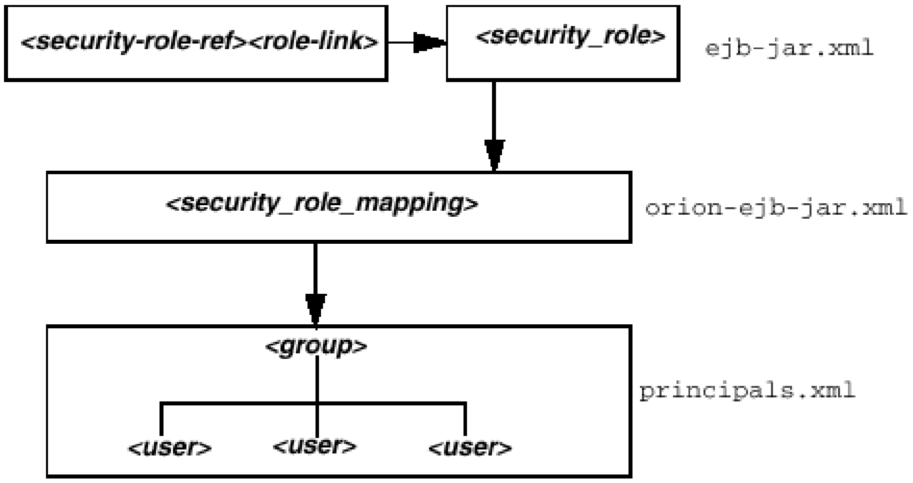
- The EJB deployment descriptor describes access rules using logical roles.

- The OC4J-specific deployment descriptor maps the logical roles to concrete users and groups, which are defined either the JAZN or XML user managers.

Users and groups are identities known by the container. Roles are the *logical* identities each application uses to indicate access rights to its different objects. The username/passwords can be digital certificates and, in the case of SSL, private key pairs.

Thus, the definition and mapping of roles is demonstrated in Figure 12–1.

Figure 12–1 Role Mapping



Defining users, groups, and roles are discussed in the following sections:

- Specifying Users and Groups
- Specifying Logical Roles in the EJB Deployment Descriptor
- Specifying Unchecked Security for EJB Methods
- Specifying the runAs Security Identity
- Mapping Logical Roles to Users and Groups
- Specifying a Default Role Mapping for Undefined Methods
- Specifying Users and Groups by the Client

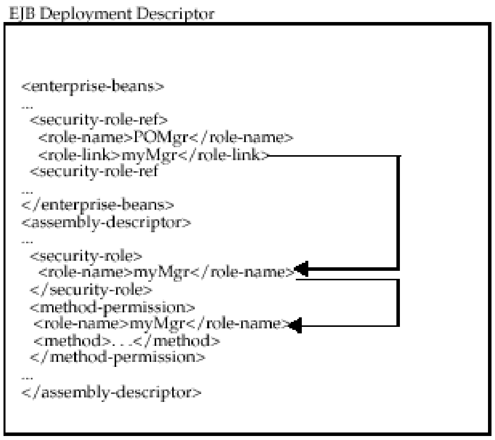
Specifying Users and Groups

OC4J supports the definition of users and groups—either shared by all deployed applications or specific to given applications. You define shared or application-specific users and groups within either the JAZN or XML user managers. See the *Oracle Application Server Containers for J2EE User's Guide* and *Oracle Application Server Containers for J2EE Services Guide.* for directions.

Specifying Logical Roles in the EJB Deployment Descriptor

As shown in Figure 12–2, you can use a logical name for a role within your bean implementation, and map this logical name to the correct database role or user. The mapping of the logical name to a database role is specified in the OC4J-specific deployment descriptor. See "Mapping Logical Roles to Users and Groups" on page 12-10 for more information.

Figure 12–2 Security Mapping



```
<enterprise-beans>
...
 <security-role-ref>
   <role-name>POMgr</role-name>
   <role-link>myMgr</role-link>
 <security-role-ref
...
</enterprise-beans>
<assembly-descriptor>
...
 <security-role>
   <role-name>myMgr</role-name>
 </security-role>
 <method-permission>
  <role-name>myMgr</role-name>
  <method>. . .</method>
 </method-permission>
...
</assembly-descriptor>
```

If you use a logical name for a database role within your bean implementation for methods such as `isCallerInRole`, you can map the logical name to an actual database role by doing the following:

1. Declare the logical name within the `<enterprise-beans>` section `<security-role-ref>` element. For example, to define a role used within the purchase order example, you may have checked, within the bean's implementation, to see if the caller had authorization to sign a purchase order. Thus, the caller would have to be signed in under a correct role. In order for the

bean to not need to be aware of database roles, you can check `isCallerInRole` on a logical name, such as `POMgr`, because only purchase order managers can sign off on the order. Thus, you would define the logical security role, `POMgr` within the `<security-role-ref><role-name>` element within the `<enterprise-beans>` section, as follows:

```
<enterprise-beans>
...
  <security-role-ref>
   <role-name>POMgr</role-name>
   <role-link>myMgr</role-link>
  </security-role-ref>
</enterprise-beans>
```

The `<role-link>` element within the `<security-role-ref>` element can be the actual database role, which is defined further within the `<assembly-descriptor>` section. Alternatively, it can be another logical name, which is still defined more in the `<assembly-descriptor>` section and is mapped to an actual database role within the Oracle-specific deployment descriptor.

Note: The `<security-role-ref>` element is not required. You only specify it when using security context methods within your bean.

2. Define the role and the methods that it applies to. In the purchase order example, any method executed within the `PurchaseOrder` bean must have authorized itself as `myMgr`. Note that `PurchaseOrder` is the name declared in the `<entity | session><ejb-name>` element.

   Thus, the following defines the role as `myMgr`, the EJB as `PurchaseOrder`, and all methods by denoting the'*' symbol.

   Note: The `myMgr` role in the `<security-role>` element is the same as the `<role-link>` element within the `<enterprise-beans>` section. This ties the logical name of `POMgr` to the `myMgr` definition.

```
<assembly-descriptor>
 <security-role>
  <description>Role needed purchase order authorization</description>
  <role-name>myMgr</role-name>
 </security-role>
 <method-permission>
  <role-name>myMgr</role-name>
  <method>
   <ejb-name>PurchaseOrder</ejb-name>
   <method-name>*</method-name>
  </method>
 </method-permission>
...
</assembly-descriptor>
```

After performing both steps, you can refer to `POMgr` within the bean's implementation and the container translates `POMgr` to `myMgr`.

Note: If you define different roles within the `<method-permission>` element for methods in the same EJB, the resulting permission is a union of all the method permissions defined for the methods of this bean.

The `<method-permission><method>` element is used to specify the security role for one or more methods within an interface or implementation. According to the EJB specification, this definition can be of one of the following forms:

1. Defining all methods within a bean by specifying the bean name and using the'*' character to denote all methods within the bean, as follows:

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
  <ejb-name>EJBNAME</ejb-name>
  <method-name>*</method-name>
  </method>
</method-permission>
```

2. Defining a specific method that is uniquely identified within the bean. Use the appropriate interface name and method name, as follows:

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
  <ejb-name>myBean</ejb-name>
  <method-name>myMethodInMyBean</method-name>
  </method>
</method-permission>
```

> **Note:** If there are multiple methods with the same overloaded name, the element of this style refers to all the methods with the overloaded name.

3. Defining a method with a specific signature among many overloaded versions, as follows:

```
<method-permission>
  <role-name>myMgr</role-name>
  <method>
      <ejb-name>myBean</ejb-name>
   <method-name>myMethod</method-name>
   <method-params>
      <method-param>javax.lang.String</method-param>
      <method-param>javax.lang.String</method-param>
        </method-params>
  </method>
</method-permission>
```

The parameters are the fully-qualified Java types of the method's input parameters. If the method has no input arguments, the `<method-params>` element contains no elements. Arrays are specified by the array element's type, followed by one or more pair of square brackets, such as `int`[ ][ ].

Specifying Unchecked Security for EJB Methods

If you want certain methods to not be checked for security roles, you define these methods as unchecked, as follows:

```
<method-permission>
  <unchecked/>
  <method>
     <ejb-name>EJBNAME</ejb-name>
     <method-name>*</method-name>
  </method>
</method-permission>
```

Instead of a `<role-name>` element defined, you define an `<unchecked/>` element. When executing any methods in the `EJBNAME` bean, the container does not check for security. Unchecked methods always override any other role definitions.

Specifying the runAs Security Identity

You can specify that all methods of an EJB execute under a specific identity. That is, the container does not check different roles for permission to run specific methods; instead, the container executes all of the EJB methods under the specified security identity. You can specify a particular role or the caller's identity as the security identity.

Specify the runAs security identity in the `<security-identity>` element, which is contained in the `<enterprise-beans>` section. The following XML demonstrates that the `POMgr` is the role under which all the entity bean methods execute.

```
<enterprise-beans>
 <entity>
 ...
  <security-identity>
     <run-as>
        <role-name>POMgr</role-name>
     </run-as>
  </security-identity>
...
 </entity>
</enterprise-beans>
```

Alternatively, the following XML example demonstrates how to specify that all methods of the bean execute under the identity of the caller:

```
<enterprise-beans>
 <entity>
 ...
  <security-identity>
     <use-caller-identity/>
  </security-identity>
...
 </entity>
</enterprise-beans>
```

Mapping Logical Roles to Users and Groups

You can use logical roles or actual users and groups in the EJB deployment descriptor. However, if you use logical roles, you must map them to the actual users and groups defined either in the JAZN or XML User Managers.

Map the logical roles defined in the application deployment descriptors to JAZN or XML User Manager users or groups through the `<security-role-mapping>` element in the OC4J-specific deployment descriptor.

- The `name` attribute of this element defines the logical role that is to be mapped.
- The `group` or `user` element maps the logical role to a group or user name. This group or user must be defined in the JAZN or XML User Manager configuration. See *Oracle Application Server Containers for J2EE User's Guide* and *Oracle Application Server Containers for J2EE Services Guide* for a description of the JAZN and XML User Managers.

Example 12–1 Mapping Logical Role to Actual Role

This example maps the logical role `POMGR` to the `managers` group in the `orion-ejb-jar.xml` file. Any user that can log in as part of this group is considered to have the `POMGR` role; thus, it can execute the methods of `PurchaseOrderBean`.

```
<security-role-mapping name="POMGR">
 <group name="managers" />
</security-role-mapping>
```

Note: You can map a logical role to a single group or to several groups.

To map this role to a specific user, do the following:

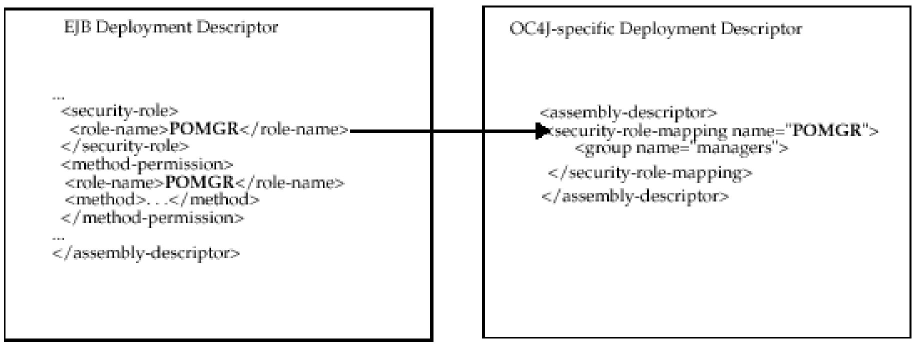
```
<security-role-mapping name="POMGR">
 <user name="guest" />
</security-role-mapping>
```

Lastly, you can map a role to a specific user within a specific group, as follows:

```
<security-role-mapping name="POMGR">
 <group name="managers" />
 <user name="guest" />
</security-role-mapping>
```

As shown in Figure 12–3, the logical role name for `POMGR` defined in the EJB deployment descriptor is mapped to `managers` within the OC4J-specific deployment descriptor in the `<security-role-mapping>` element.

Figure 12–3 Security Mapping


EJB Deployment Descriptor
OC4J-specific Deployment Descriptor

```
...
 <security-role>
   <role-name>POMGR</role-name>
 </security-role>
 <method-permission>
  <role-name>POMGR</role-name>
  <method>. . .</method>
 </method-permission>
...
</assembly-descriptor>
```

```
<assembly-descriptor>
 <security-role-mapping name="POMGR">
    <group name="managers">
 </security-role-mapping>
</assembly-descriptor>
```

Notice that the `<role-name>` in the EJB deployment descriptor is the same as the name attribute in the `<security-role-mapping>` element in the OC4J-specific deployment descriptor. This is what identifies the mapping.

Specifying a Default Role Mapping for Undefined Methods

If any methods have not been associated with a role mapping, they are mapped to the default security role through the `<default-method-access>` element in the `orion-ejb-jar.xml` file. The following is the automatic mapping for any insecure methods:

```
<default-method-access>
```

```
   <security-role-mapping name="&lt;default-ejb-caller-role&gt;"
                          impliesAll="true" />
   </security-role-mapping>
</default-method-access>
```

The default role is `<default-ejb-caller-role>` and is defined in the `name` attribute. You can replace this string with any name for the default role. The `impliesAll` attribute indicates whether any security role checking occurs for these methods. This attribute defaults to true, which states that no security role checking occurs for these methods. If you set this attribute to false, the container will check for this default role on these methods.

If the `impliesAll` attribute is false, you must map the default role defined in the `name` attribute to a JAZN or XML user or group through the `<user>` and `<group>` elements. The following example shows how all methods not associated with a method permission are mapped to the "`others`" group.

```
<default-method-access>
   <security-role-mapping name="default-role" impliesAll="false" />
      <group name="others" />
   </security-role-mapping>
</default-method-access>
```

Specifying Users and Groups by the Client

In order for the client to access methods that are protected by users and groups, the client must provide the correct user or group name with a password that the JAZN or XML User Manager recognizes. And the user or group must be the same one as designated in the security role for the intended method. See "Specifying Credentials in EJB Clients" on page 12-12 for more information.

Specifying Credentials in EJB Clients

When you access EJBs in a *remote* container, you must pass valid credentials to this container.

- Stand-alone clients define their credentials in the `jndi.properties` file deployed with the EAR file.
- Servlets or JavaBeans running within the container pass their credentials within the `InitialContext`, which is created to look up the remote EJBs.

Credentials in JNDI Properties

Indicate the username (principal) and password (credentials) to use when looking up remote EJBs in the `jndi.properties` file.

For example, if you want to access remote EJBs as `POMGR/welcome`, define the following properties. The `factory.initial` property indicates that you will use the Oracle JNDI implementation:

```
java.naming.security.principal=POMGR
java.naming.security.credentials=welcome
java.naming.factory.initial=
   com.evermind.server.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://myhost/ejbsamples
```

In your application program, authenticate and access the remote EJBs, as shown below:

```
InitialContext ic = new InitialContext();
CustomerHome =
(CustomerHome)ic.lookup("java:comp/env/purchaseOrderBean");
```

Credentials in the InitialContext

To access remote EJBs from a servlet or JavaBean, pass the credentials in the `InitialContext` object, as follows:

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
   "com.evermind.server.ApplicationClientInitialContextFactory");
env.put(Context.SECURITY_PRINCIPAL, "POMGR");
env.put(Context.SECURITY_CREDENTIALS, "welcome");
Context ic = new InitialContext (env);
CustomerHome =
   (CustomerHome)ic.lookup("java:comp/env/purchaseOrderBean")
```

13 J2EE Connector Architecture Security

This chapter describes the security issues affecting the J2EE Connector Architecture in an Oracle Application Server Containers for J2EE (OC4J) application. For full information on the J2EE Connector Architecture, see the *Oracle Application Server Containers for J2EE Services Guide*. This chapter covers the following topics:

- Deploying Resource Adapters
- Specifying Container-Managed or Component-Managed Sign-On
- Authentication in Container-Managed Sign-On

Deploying Resource Adapters

This section discusses deployment descriptors, deploying standalone resource adapters, and deploying embedded resource adapters.

Oracle Application Server Containers for J2EE supports three deployment descriptors: `ra.xml`, `oc4j-ra.xml`, and `oc4j-connectors.xml`. The `ra.xml` descriptor is normally supplied with the resource adapter. Whenever you deploy a resource adapter within an EAR file, Oracle Application Server Containers for J2EE generates `oc4j-connectors.xml` and `oc4j-ra.xml`. You should manually edit the second file.

The oc4j-ra.xml Descriptor

The `oc4j-ra.xml` descriptor provides Oracle Application Server Containers for J2EE-specific deployment information (Java Naming and Directory Interface (JNDI) path name and connector properties) for resource adapters. For each resource adapter, `oc4j-ra.xml` contains one or more `<connector-factory>` elements specifying a JNDI name corresponding to a set of configuration parameter values. Oracle Application Server Containers for J2EE binds each connection into the proper JNDI namespace location as a `ConnectionFactory` instance.

A `<connector-factory>` element can contain an optional `<security-config>` element that describes how to supply user names and passwords to the EIS.

The <security-config> Element

The `<security-config>` element specifies the user name and password for container-managed sign-ons.

There are two ways of supplying this information in the `<security-config>` element of the `oc4j-ra.xml` file:

- Specifying mapping subelements explicitly (in the `<principal-mapping-entries>` subelement)
- Specifying the name of a user-created mapping class that either implements `oracle.j2ee.connector.PrincipalMapping` or inherits from `oracle.j2ee.AbstractPrincipalMapping` (in the `<principal-mapping-interface>` subelement)

Authentication issues are discussed in detail in "Authentication in Container-Managed Sign-On" on page 13-6. This section discusses only the syntax for the `<security-config>` element.

A `<security-config>` element contains either a `<principal-mapping-entries>` element, specifying user names and passwords explicitly; a `<principal-mapping-interface>` element, specifying the name of the mapping class; or a `<jaas-module>` element, specifying the JAAS module to be used for authentication.

```
<security-config>
  <principal-mapping-entries>                               // 1
     <default-mapping>                                      // 2
        <res-user>username</res-user>                      // 3
        <res-password>password</res-password>              // 4
     </default-mapping>
     <principal-mapping-entry>                              // 5
        <initiating-user>iuname</initiating-user> // 6
        <res-user>username</res-user>
        <res-password>password</res-password>
     </principal-mapping-entry>
  </principal-mapping-entries>

  <principal-mapping-interface>                           // 7
    <impl-class>classname</impl-class>                   // 8
    <property name="propname"
              value="propvalue" />                       // 9
  </principal-mapping-interface>

  <jaas-module>                                           // 10
      <jaas-application-name>                             // 11
         appname
      </jaas-application-name>
  </jaas-module>
</security-config>
```

1. `<principal-mapping-entries>`: Ppovides a declarative specification for resource mapping. This element begins with an optional `<default-mapping>` element; it continues with one or more `<principal-mapping-entry>` elements.
2. `<default-mapping>`: specifies the user name and password for the default resource principal.
3. `<res-user>`: specifies user name.
4. `<res-password>`: specifies password.

Note: This element supports password indirection. For more information, refer to "Creating An Indirect Password" on page 10-3.

5. `<principal-mapping-entry>`: specifies a mapping from a single initiating principal to a resource principal and password.
6. `<initiating-user>`: specifies the initiating principal.
7. `<principal-mapping-interface>`: specifies information necessary to employ user-created classes to provide mappings.
8. `<impl-class>`: specifies the name of the user-provided `PrincipalMapping` implementation.
9. `<property name="`*`propname`*`" value="`*`propvalue`*`">`: specifies information specific to your `PrincipalMapping` implementation: for instance, the path of the principal mapping file, or LDAP server connection information. (This element is optional and it can be repeated.)
10. `<jaas-module>`: specifies the JAAS module that is used for authentication. It has only one element, `<jaas-application-name>`.
11. `<jaas-application-name>`: specifies the name of the JAAS module that is used for authentication.

The oc4j-connectors.xml Descriptor

The `oc4j-connectors.xml` descriptor configures the resource adapters that are deployed by `oc4j-ra.xml`. The `oc4j-connectors.xml` descriptor lists the standalone resource adapters that are deployed in this Oracle Application Server Containers for J2EE instance, as well as the resource adapters that are embedded within an application. This descriptor contains, for each individual connector, a `connector>` element that specifies the name and path name for the connector. Each `<connector>` element contains a `<security-permission>` element that defines the permissions granted to each resource adapter. The syntax is:

`<security-permission enabled="`*`booleanvalue`*`">`

This element specifies the permissions to be granted to each resource adapter. Each `<security-permission>` contains a `<security-permission-spec>` that conforms to the Java 2 Security policy file syntax.

Oracle Application Server Containers for J2EE automatically generates a `<security-permission>` element in `oc4j-connectors.xml` for each

`<security-permission>` element in `ra.xml`. Each generated element has the `enabled` attribute set to `false`. Setting the `enabled` attribute to `true` grants the named permission.

Example:

```
<oc4j-connectors>
  <connector name="myEIS" path="eis.rar">
  . . .
    <security-permission>
      <security-permission-spec enabled="false">
        grant {permission java.lang.RuntimePermission "LoadLibrary", *'};
      </security-permission-spec>
    </security-permission>
  </connector>
</oc4j-connectors>
```

Specifying Container-Managed or Component-Managed Sign-On

Applications can use either application components or the Oracle Application Server Containers for J2EE application server to manage resource-adapter sign-on to the EIS system. Specify the manager using the `<res-auth>` deployment descriptor element for EJB or Web components. If `<res-auth>` is set to `Application`, then the application component signs on to the EIS programmatically. The application component is responsible for providing explicit security information for the sign-on. If `<res-auth>` is set to `Container`, then Oracle Application Server Containers for J2EE provides the resource principal and credentials that are required for signing on to the EIS.

Example:

```
Context initctx = new InitialContext();
// perform JNDI lookup to obtain a connection factory
javax.resource.cci.ConnectionFactory cxf =

(javax.resource.cci.ConnectionFactory)initctx.lookup("java:com/env/eis/MyEIS");
  // For container-managed sign-on, no security information is passed in the
getConnection call
  javax.resource.cci.Connection cx = cxf.getConnection();
  // If component-managed sign-on is specified, the code should instead provide
explicit security
  // information in the getConnection call
  // We need to get a new ConnectionSpec implementation instance for setting
login
  // attributes
```

```
com.myeis.ConnectionSpecImpl connSpec = ...
connSpec.setUserName("EISuser");
connSpec.setPassword("EISpassword");
javax.resource.cci.Connection cx = cxf.getConnection(connSpec);
```

In either case, the `createManagedConnection` method in the resource adapter's implementation of `javax.resource.spi.ManagedConnectionFactory` interface is called to create a physical connection to the EIS.

If you specify component-managed sign-on, then Oracle Application Server Containers for J2EE invokes the `createManagedConnection` method with a null `Subject` and the `ConnectionRequestInfo` object that is passed in from the application component code. If you specify container-managed sign-on, then Oracle Application Server Containers for J2EE provides a `javax.security.auth.Subject` object to the `createManagedConnection` method. The content of the `Subject` object depends on the value in the `<authentication-mechanism-type>` and `<credential-interface>` elements in the resource adapter deployment descriptor.

If `<authentication-mechanism-type>` is `BasicPassword` and `<credential-interface>` is `javax.resource.spi.security.PasswordCredential`, then the `Subject` object must contain `javax.resource.spi.security.PasswordCredential` objects in the private credential set.

On the other hand, if `<authentication-mechanism-type>` is Kerberos version 5 (Kerbv5) or any other non-password-based authentication mechanism, and `<credential-interface>` is `javax.resource.spi.security.GenericCredential`, then the `Subject` object must contain credentials represented by instances of implementers of the `javax.resource.spi.security.GenericCredential` interface. The `GenericCredential` interface is used for resource adapters that support non-password-based authentication mechanisms, such as Kerberos.

Authentication in Container-Managed Sign-On

When using container-managed sign-on, Oracle Application Server Containers for J2EE must provide a resource principal and its credentials to the EIS. The principal and credentials can be obtained in one of the following ways:

- Configured Identity: the resource principal is independent of the initiating or caller principal and can be configured at deployment time in a deployment descriptor.

- Principal Mapping: the resource principal is determined by a mapping from the identity and security attributes of the initiating or caller principal.
- Caller Impersonation: the resource principal acts on behalf of an initiating or caller principal by delegating the caller's identity and credentials to the EIS.
- Credentials Mapping: the resource principal is identical to the initiating or caller principal, but with its credential mapped from the authentication type that Oracle Application Server Containers for J2EE uses to the authentication type that the EIS uses. An example would be to map a public key certificate-based credential associated with a principal to a Kerberos credential.

Oracle Application Server Containers for J2EE supports all these methods with three authentication mechanisms:

- JAAS Pluggable Authentication
- User-Created Authentication Classes
- Modifying oc4j-ra.xml

The following sections discuss these mechanisms in detail.

JAAS Pluggable Authentication

Oracle Application Server Containers for J2EE furnishes a JAAS pluggable authentication framework that conforms to Appendix C in the Connector Architecture 1.0 specification. With this framework, an application server and its underlying authentication services remain independent from each other, and new authentication services can be plugged in without requiring modifications to the application server.

Authentication services can obtain resource principals and credentials using any of the following modules:

- Principal Mapping JAAS module
- Credential Mapping JAAS module
- Kerberos JAAS module (for Caller Impersonation)

The JAAS login modules can be furnished by the customer, the EIS vendors, or the resource adapter vendors. Login modules must implement the `javax.security.auth.spi.LoginModule` interface, as documented in the Sun JAAS specification.

Oracle Application Server Containers for J2EE provides initiating user subjects to login modules by passing an instance of `javax.security.auth.Subject` containing any public certificates and an instance of `oracle.j2ee.connector.InitiatingPrincipal` representing the Oracle Application Server Containers for J2EE user. Oracle Application Server Containers for J2EE can pass a null `Subject` if there is no authenticated user (that is, an anonymous user). The JAAS login module's login method must, based on the initiating user, find the corresponding resource principal and create new `PasswordCredential` or `GenericCredential` instances for the resource principal. The resource principal and credential objects are then added to the initiating `Subject` in the `commit` method. The resource credential is passed to the `createManagedConnection` method in the `javax.resource.spi.ManagedConnectionFactory` implementation that is provided by the resource adapter. If a null `Subject` is passed, the JAAS login module is responsible for creating a new `javax.security.auth.Subject` containing the resource principal and the appropriate credential.

The InitiatingPrincipal and InitiatingGroup Classes

The classes `oracle.j2ee.connector.InitiatingPrincipal` and `oracle.j2ee.connector.InitiatingGroup` represent Oracle Application Server Containers for J2EE users to the JAAS login modules. Oracle Application Server Containers for J2EE creates instances of `oracle.j2ee.connector.InitiatingPrincipal` and incorporates them into the Subject that is passed to the `initialize` method of the login modules. The `oracle.j2ee.connector.InitiatingPrincipal` class implements the `java.security.Principal` interface and adds the method `getGroups()`.

```
/**
 * Returns a Set of groups (or roles in JAZN terminology) that this
 * principal is a member of.
 *
 * @return A set of InitiatingGroup objects representing the groups
 *      that this pricipal belongs to.
 */
public Set getGroups()
```

The `getGroups` method returns a `java.util.Set` of `oracle.j2ee.connector.InitiatingGroup` objects, representing the Oracle Application Server Containers for J2EE groups or JAZN roles for this Oracle Application Server Containers for J2EE user. The group membership is defined in Oracle Application Server Containers for J2EE-specific descriptor files such as `principals.xml` or `jazn-data.xml`, depending on the user manager. The `oracle.j2ee.connector.InitiatingGroup` class implements but does not extend the `java.security.Principal` interface.

Login modules can use `getGroups()` to provide mappings between Oracle Application Server Containers for J2EE groups and EIS users. The `java.security.Principal` interface methods support mappings between Oracle Application Server Containers for J2EE users and EIS users. Login modules do not need to refer to the `oracle.j2ee.connector.InitiatingPrincipal` and `oracle.j2ee.connector.InitiatingGroup` classes if they do not provide mappings between Oracle Application Server Containers for J2EE groups and EIS users.

JAAS and the <connector-factory> Element

Each `<connector-factory>` element in `oc4j-ra.xml` can specify a different JAAS login module. Specify a name for the connector factory configuration in the `<jaas-module>` element. Here is an example of a `<connector-factory>` element in `oc4j-ra.xml` that uses JAAS login modules for container-managed sign-on:

```
  <connector-factory connector-name="myBlackbox" location="eis/myEIS1">
    <description>Connection to my EIS</description>
    <config-property name="connectionURL"
value="jdbc:oracle:thin:@localhost:5521:orcl" />
    <security-config>
      <jaas-module>
        <jaas-application-name>JAASModuleDemo</jaas-application-name>
      </jaas-module>
    </security-config>
  </connector-factory>
```

In JAAS, you must specify which `LoginModule` to use for a particular application, and in what order to invoke the `LoginModules`. JAAS uses the value that are specified in the `<jaas-application-name>` element to look up `LoginModules`. See the Oracle Application Server Containers for J2EE Security Guide for more information.

User-Created Authentication Classes

Oracle Application Server Containers for J2EE provides the `oracle.j2ee.connector.PrincipalMapping` interface for principal mapping.

```
package oracle.j2ee.connector;

public interface PrincipalMapping
{
/**
```

```
* Initializes the various settings for the PrincipalMapping implementation
class.
* Implementation class may use the properties for setting default user name and
* password, LDAP connect info, or default mapping.
*
* OC4J will pass the properties specified in the <principal-mapping-interface>
* element in oc4j-ra.xml to this method.
*
* @param prop A Properties object containing the set up information required
*             by the implementation class.
*/
  public void init(Properties prop);

/**
* The ManagedConnectionFactory instance that can be used in creating a
   * PasswordCredential.
   *
 * @param mcf The ManagedConnectionFactory instance that is needed when
  *creating a PasswordCredential instance
  */
  public void setManagedConnectionFactory(ManagedConnectionFactory mcf);

  /**
* Passes the authentication mechanism(s) supported by the resource
* adapter to the PrincipalMapping implementation class.
   * The key of the map passed is a String containing the supported mechanism
* type, such as "BasicPassword", or "Kerbv5". The value is a String
  * containig the corresponding credentials interface as declared in ra.xml,
* such as "javax.resource.spi.security.PasswordCredential".
  *
 * The map may contain multiple elements if the resource adatper supports
  * multiple authentication mechanisms.
   *
 * @param authMechanisms The authentication mechanisms and their corresponding
   *  credentials intereface supported by the resource adapter
  */
  public void setAuthenticationMechanisms(Map authMechanisms);

    /**
* This is the method that performs the principal mapping. An application user
   * subject is passed, and the implemetation of this method should return
   * a subject for use by the resource adapter to log in to the EIS resource
* per the Connector specifications.
   *
 * OC4J will only called this method for container-managed sign on.
```

```
*
 * @param initiatingSubject A Subject containing the application server logged
   *     in principals and public credentials.
     *
    * @return A Subject for use by resource adapter to log in to the remote EIS.
    *      It may return null if the proper resource principal cannot be
determined.
  */
  public Subject mapping(Subject initiatingSubject);
}
```

The `mapping` method must return a `Subject` containing the resource principal and credential. The `Subject` that is returned must adhere to either option A or option B in section 8.2.6 of the Connector Architecture 1.0 specification. Oracle Application Server Containers for J2EE invokes the `mapping` method with the initiating user as the `initiatingPrincipal`.

Oracle Application Server Containers for J2EE also provides the abstract class `oracle.j2ee.connector.AbstractPrincipalMapping`. This class furnishes a default implementation of the `setManagedConnectionFactory()` and `setAuthenticationMechanism()` methods, as well as utility methods to determine whether the resource adapter supports the `BasicPassword` or Kerberos version 5 (Kerbv5) authentication methods, and a method for extracting the `Principal` from the application server user `Subject`. By extending the `oracle.j2ee.connector.AbstractPrincipalMapping` class, developers need only implement the `init` and `mapping` methods.

Here are the utility methods provided by the `oracle.j2ee.connector.AbstractPrincipalMapping` class:

```
/**
   *  Utility method provided by this abstract class to return
  *  the ManagedConnectionFactory instance for use to create a
    *  PasswordCredentials object
   *
    *  @return The ManagedConnectionFactory instance that is needed when
    *    creating a PasswordCredential instance
    */
  public ManagedConnectionFactory getManagedConnectionFactory()

   /**
   * Utility method provided by this abstract class to return the Map
   * of all authentication mechanisms supported by this resource adapter.
  * The key of the map passed is a String containing the supported mechanism
```

```
 * type, such as "BasicPassword", or "Kerbv5". The value is a String
* containig the corresponding credentials interface as declared in ra.xml,
 * such as "javax.resource.spi.security.PasswordCredential".
*
  * @return The authentication mechanisms and their corresponding
    *      credentials intereface supported by the resource adpater
 */
  public Map getAuthenticationMechanisms()

   /**
 * Utility method provided by this abstract class to return whether
 * BasicPassword authention mechanism is supported by this resource
 * adapter.
 *
  * @return true if BasicPassword authentication mechanism is supported
 *      by the resource adapter, false otherwise.
*/
  public boolean isBasicPasswordSupported()

  /**
 * Utility method provided by this abstract class to return whether
 * Kerbv5 authention mechanism is supported by this resource
* adapter.
 *
  * @return true if Kerbv5 authentication mechanism is supported
*      by the resource adapter, false otherwise.
*/
  public boolean isKerbv5Supported()

 /**
 * Utility method provided by this abstract class to extract the
    * Principal object from the given application server user subject
  * passed from OC4J.
*
  * @param subject The application server user subject passed from
   *       OC4J.
*
  * @return The principal extracted from the given subject
   */
  public Principal getPrincipal(Subject subject)
```

After you create your implementation class, copy a JAR file containing the class into the directory containing the decompressed RAR file. This directory is typically *OC4J_HOME*/applications/application_name/*rar-name*. After copying the file, edit oc4j-ra.xml to contain a <principal-mapping-interface>

element for the new class; see "The <security-config> Element" on page 13-2 for details.

Extending AbstractPrincipalMapping

This simple example demonstrates how to extend the `oracle.j2ee.connector.AbstractPrincipalMapping` abstract class to provide a principal mapping that always maps the user to the default user and password. Specify the default user and password by using properties under the `<principal-mapping-interface>` element in `oc4j-ra.xml`.

The `PrincipalMapping` class is called `MyMapping`. It is defined as follows:

```
package com.acme.app;

import java.util.*;
import javax.resource.spi.*;
import javax.resource.spi.security.*;
import oracle.j2ee.connector.AbstractPrincipalMapping;
import javax.security.auth.*;
import java.security.*;

public class MyMapping extends AbstractPrincipalMapping
{
  String m_defaultUser;

   String m_defaultPassword;

    public void init(Properties prop)
    {
      if (prop != null)
      {
           // Retrieves the default user and password from the properties
         m_defaultUser = prop.getProperty("user");
           m_defaultPassword = prop.getProperty("password");
      }
    }

    public Subject mapping(Subject initiatingSubject)
    {
      // This implementation only supporst BasicPassword authentication
      // mechanism. Return if the resource adapter does not support it.
      if (!isBasicPasswordSupported())
            return null;
      // Use the utility method to retrieve the Principal from the
      // OC4J user. This code is included here only as an example.
      // The principal obtained is not being used in this method.
       Principal principal = getPrincipal(initiatingSubject);

    char[] resPasswordArray = null;
      if (m_defaultPassword != null)
         resPasswordArray = m_defaultPassword.toCharArray();

       // Create a PasswordCredential using the default user name and
      // password, and add it to the Subject per option A in section
    // 8.2.6 in the Connector 1.0 spec.
     PasswordCredential cred = new PasswordCredential(m_defaultUser,
resPasswordArray);
        cred.setManagedConnectionFactory(getManagedConnectionFactory());
       initiatingSubject.getPrivateCredentials().add(cred);
      return initiatingSubject;
    }
}
```

You add a `<principal-mapping-interface>` entry to `oc4j-ra.xml` that specifies `com.acme.app.MyMapping` for the principal mapping mechanism:

```
  <connector-factory name="..." location="...">
    ...
    <security-config>
   <principal-mapping-interface>
     <impl-class>com.acme.app.MyMapping</impl-class>
     <property name="user" value="scott" />
     <property name="password" value="tiger" />
   </principal-mapping-interface>
    </security-config>
    ...
  </connector-factory>
```

Modifying oc4j-ra.xml

If you prefer, you can create default principal mappings in the `oc4j-ra.xml` file. To employ the default principal mappings mechanism, use the `<principal-mapping-entries>` subelement under the `<security-config>` element. For syntax details, see "The <security-config> Element" on page 13-2.

Use the `<default-mapping>` element to specify the user name and password for the default resource principal. This principal is used to log on to the EIS if there is no `<principal-mapping-entry>` element whose initiating user corresponds to the current initiating principal. If no default mapping is specified, Oracle Application Server Containers for J2EE uses the values of the configuration properties `UserName` and `Password` from the deployment descriptor (either in `ra.xml` or `oc4j-ra.xml`), assuming that these defaults are acceptable to the resource adapter. If neither configuration properties nor a default mapping is specified, Oracle Application Server Containers for J2EE may not be able to log in to the EIS.

Each `<principal-mapping-entry>` element contains a mapping from initiating principal to resource principal and password.

For example, if the Oracle Application Server Containers for J2EE principal scott should be logged in to a certain EIS, myEIS1, as user name scott and password tiger, while all other Oracle Application Server Containers for J2EE users should be logged in to the EIS using user name guest with password guestpw, the <connector-factory> element in oc4j-ra.xml should look like this:

```
<connector-factory name="..." location="...">
    ...
  <security-config>
    <principal-mapping-entries>
      <default-mapping>
        <res-user>guest</res-user>
        <res-password>guestpw</res-password>
      </default-mapping>
      <principal-mapping-entry>
        <initiating-user>scott</initiating-user>
        <res-user>scott</res-user>
        <res-password>tiger</res-password>
      </principal-mapping-entry>
    </principal-mapping-entries>
  </security-config>
    ...
</connector-factory>
```

14 Configuring CSIv2

Oracle Application Server Containers for J2EE supports the Common Secure Interoperability Version 2 protocol (CSIv2). CSIv2 specifies different conformance levels; Oracle Application Server Containers for J2EE complies with the EJB specification, which requires conformance level 0.

This chapter covers the following topics:

Note: If your application uses JAAS, you must configure the JAAS Provider to use CSIv2; see Table 3–8, "RealmLoginModule Options" for details.

Introduction to CSIv2 Security Properties

Common Secure Interoperability version 2 (CSIv2) is an Object Management Group (OMG) standard for a secure interoperable wire protocol that supports authorization and identity delegation. You configure CSIv2 properties in three different locations:

- `internal_settings.xml`
- `orion-ejb-jar.xml`
- `ejb_sec.properties`

These configuration files are discussed in "CSIv2 Security Properties in internal-settings.xml" on page 14-4, "EJB Client Security Properties in ejb_sec.properties" on page 14-6, "CSIv2 Security Properties in orion-ejb-jar.xml" on page 14-6, and "EJB Client Security Properties in ejb_sec.properties" on page 14-8.

EJB Server Security Properties in internal-settings.xml

You specify server security properties in `internal-settings.xml`.

Note: You cannot edit `internal-settings.xml` with the Enterprise Manager.

This file specifies certain properties as values within `<sep-property>` entities. Table 14–1, "EJB Server Security Properties" contains a list of properties.

The table refers to *keystore* and *truststore* files, which use the Java Key Store (JKS), a JDK-specified format, to store keys and certificates. A keystore stores a map of private keys and certificates. A truststore stores trusted certificates for the certificate authorities (CAs; such as VeriSign and Thawte).

Table 14–1 EJB Server Security Properties

Property	Meaning
`port`	IIOP port number (defaults to `5555`)
`ssl`	`true` if IIOP/SSL is supported, `false` otherwise

Table 14–1 EJB Server Security Properties (Cont.)

Property	Meaning
`ssl-port`	IIOP/SSL port number (defaults to `5556`) This port is used for server-side authentication only. If your application uses client and server authentication, you also need to set `ssl-client-server-auth-port`.
`ssl-client-server-auth-port`	Port used for client and server authentication (defaults to `5557`). This is the port on which OC4J listens for SSL connections that require both client and server authentication. If not set, OC4J will listen on `ssl-port` + 1 for client-side authentication.
`keystore`	Name of keystore (used only if `ssl` is `true`)
`keystore-password`	the keystore password (used only if `ssl` is `true`)
`trusted-clients`	Comma-separated list of hosts whose identity assertions can be trusted. Each entry in the list can be an IP address, a host name, a host name pattern (for instance, `*.example.com`), or `*`; `*` alone means that all clients are trusted. The default is to trust no clients.
`truststore`	Name of truststore. If you do not specify a truststore for a server, OC4J uses the keystore as the truststore (used only if `ssl` is `true`).
`truststore-password`	Truststore password (can only be set if `ssl` is `true`)

Note: In Table 14–1, the properties `keystore-password` `andtruststore-password` support password indirection.

If Oracle Application Server Containers for J2EE is started by the Oracle Process Management Notification service (OPMN) in an OracleAS (as opposed to standalone) environment, then ports specified in `internal-settings.xml` are ignored. If OPMN is configured to disable IIOP for a particular Oracle Application Server Containers for J2EE instance, then, even though IIOP may be enable through `internal-settings.xml` (as pointed to by `server.xml`), IIOP is not enabled.

The following example shows a typical `internal-settings.xml` file:

```
<server-extension-provider name="IIOP"
        class="com.oracle.iiop.server.IIOPServerExtensionProvider">
    <sep-property name="port" value="5555" />
    <sep-property name="host" value="localhost" />
    <sep-property name="ssl" value="false" />
    <sep-property name="ssl-port" value="5556" />
    <sep-property name="ssl-client-server-auth-port" value="5557" />
    <sep-property name="keystore" value="keystore.jks" />
    <sep-property name="keystore-password" value="123456" />
    <sep-property name="truststore" value="truststore.jks" />
    <sep-property name="truststore-password" value="123456" />
    <sep-property name="trusted-clients" value="*" />
</server-extension-provider>
```

Note: Although the default value of `port` is one less than the default value for `ssl-port`, this relationship is not required.

Here is the DTD for `internal-settings.xml`:

```
<!-- A server extension provider that is to be plugged in to the server.
-->
<!ELEMENT server-extension-provider (sep-property*) (#PCDATA)>
<!ATTLIST server-extension-provider name class CDATA #IMPLIED>
<!ELEMENT sep-property (#PCDATA)>
<!ATTLIST sep-property name value CDATA #IMPLIED>
<!-- This file contains internal server configuration settings. -->
<!ELEMENT internal-settings (server-extension-provider*)>
```

CSIv2 Security Properties in internal-settings.xml

This section discusses the semantics of the values you set within the `<sep-property>` element in `internal_settings.xml`. For details of syntax, see "EJB Server Security Properties in internal-settings.xml" on page 14-2.

To use the CSIv2 protocol with Oracle Application Server Containers for J2EE, you must both set `ssl` to `true` and specify an IIOP/SSL port (`ssl-port`).

- If you do not set `ssl` to `true`, then CSIv2 is not enabled. Setting `ssl` to `true` permits clients and servers to use CSIv2, but does not require them to communicate using SSL.

- If you do not specify an `ssl-port`, then no CSIv2 component tag is inserted by the server into the IOR, even if you configure an `<ior-security-config>` entity in `orion-ejb-jar.xml`.

When IIOP/SSL is enabled on the server, Oracle Application Server Containers for J2EE listens on two different sockets—one for server authentication alone and one for server and client authentication. You specify the server authentication port within the `<sep-property>` element; the server and client authentication listener uses the port number immediately following.

For SSL clients using server authentication alone, you can specify:

- Truststore only
- Both keystore and truststore.
- Neither

If you specify neither keystore nor truststore, the handshake may fail if there are no default truststores established by the security provider.

SSL clients using client-side authentication must specify both a keystore and a truststore. The certificate from the keystore is used for client authentication.

CSIv2 Security Properties in ejb_sec.properties

If the client does not use client-side SSL authentication, you must set `client.sendpassword` in the `ejb_sec.properties` file in order for the client runtime to insert a security context and send the user name and password. You must also set `server.trustedhosts` to include your server.

Note: Server-side authentication takes precedence over a user name and password.

If the client does use client-side SSL authentication, the server extracts the `DistinguishedName` from the client's certificate and then looks it up in the corresponding user manager; it does not perform password authentication.

Trust Relationships

Two types of trust relationships exist:

- Clients trusting servers to transmit user names and passwords using non-SSL connections

- Servers trusting clients to send *identity assertions*, which delegate an originating client's identity

Clients list trusted servers in the EJB property `oc4j.iiop.trustedServers`. See Table 14–2, "EJB Client Security Properties" on page 14-9 for details. Servers list trusted clients in the `trusted-client` property of the `<sep-property>` element in `internal-settings.xml`. See "EJB Server Security Properties in internal-settings.xml" on page 14-2 for details.

Conformance level 0 of the EJB standard defines two ways of handling trust relationships:

- *presumed trust*, in which the server presumes that the logical client is trustworthy, even if the logical client has not authenticated itself to the server, and even if the connection is not secure
- *authenticated trust*, in which the target trusts the intermediate server based on authentication either at the transport level or in the `trusted-client` list or both

Note: You can also configure the server to both require SSL client-side authentication and also specify a list of trusted client (or intermediate) hosts that are allowed to insert identity assertions.

Oracle Application Server Containers for J2EE provides both kinds of trust; you configure trust using the bean's `<ior-security-config>` element in `orion-ejb-jar.xml`. See "CSIv2 Security Properties in orion-ejb-jar.xml" on page 14-6 for details.

CSIv2 Security Properties in orion-ejb-jar.xml

This section discusses the CSIv2 security properties for an EJB. You configure each individual bean's CSIv2 security policies in its `orion-ejb-jar.xml`. The CSIv2 security properties are specified within `<ior-security-config>` elements. Each element contains a `<transport-config>` element, an `<as-context>` element, and an `<sas-context>` element.

The <transport-config> element

This element specifies the transport security level. Each element within `<transport-config>` must be set to `supported`, `required`, or `none`. `None` means that the bean neither supports nor uses that feature; `supports` means that the bean permits the client to use the feature; `required` means that the bean insists that the client use the feature. The elements are:

- `<integrity>`—Is there a guarantee that all transmissions are received exactly as they were transmitted?
- `<confidentiality>`—Is there a guarantee that no third party was able to read transmissions?
- `<establish-trust-in-target>`—Does the server authenticate itself to the client?
- `<establish-trust-in-client>`—Does the client authenticate itself to the server?

Notes: If you set `<establish-trust-in-client>` to `required`, this overrides specifying `username_password` in `<as-context>`. If you do this, you must also set the `<required>` node value in the `<as-context>` section to `false`; otherwise access permission issues will arise.

Setting any of the `<transport-config>` properties to `required` means that the bean will use RMI/IIOP/SSL to communicate.

The <as-context> element

This element specifies the message-level authentication properties.

- `<auth-method>`—Must be set to either `username_password` or `none`. If set to `username_password`, beans use user names and passwords to authenticate the caller.
- `<realm>`—Must be set to `default` at this release.
- `<required>`—If set to `true`, the bean requires the caller to specify a user name and password.

The <sas-context> element

This element specifies the identity delegation properties. It has one element, `<caller-propagation>`, which can be set to `supported`, `required`, or `none`. If the `<caller-propagation>` element is set to `supported`, then this bean accepts delegated identities from intermediate servers. If it is set to `required`, then this bean requires all other beans to transmit delegated identities. If set to none, this bean does not support identity delegation.

An example:

```
<ior-security-config>
  <transport-config>
    <integrity>supported</integrity>
    <confidentiality>supported</confidentiality>
    <establish-trust-in-target>supported</establish-trust-in-target>
    <establish-trust-in-client>supported</establish-trust-in-client>
  </transport-config>
  <as-context>
    <auth-method>username_password</auth-method>
    <realm>default</realm>
    <required>true</required>
  </as-context>
  <sas-context>
    <caller-propagation>supported</caller-propagation>
  </sas-context>
</ior-security-config>
```

DTD

The DTD for the <ior-security-config> element is:

```
<!ELEMENT ior-security-config (transport-config?, as-context?
sas-context?) >
<!ELEMENT transport-config (integrity, confidentiality,
establish-trust-in-target, establish-trust-in-client) >
<!ELEMENT as-context (auth-method, realm, required) >
<!ELEMENT sas-context (caller-propagation) >
<!ELEMENT integrity (#PCDATA) >
<!ELEMENT confidentiality (#PCDATA)>
<!ELEMENT establish-trust-in-target (#PCDATA) >
<!ELEMENT establish-trust-in-client (#PCDATA) >
<!ELEMENT auth-method (#PCDATA) >
<!ELEMENT realm (#PCDATA) >
<!ELEMENT required (#PCDATA)> <!-- Must be true or false -->
<!ELEMENT caller-propagation (#PCDATA) >
```

EJB Client Security Properties in ejb_sec.properties

Any client, whether running inside a server or not, has EJB security properties. Table 14–2 lists the EJB client security properties controlled by the `ejb_sec.properties` file. By default, Oracle Application Server Containers for J2EE searches for this file in the current directory when running as a client or in `J2EE_HOME/config` when running in the server. You can specify this file's location explicitly with `-Dejb_sec_properties_location=`*`pathname`*.

Table 14–2 EJB Client Security Properties

Property	Meaning
`# oc4j.iiop.keyStoreLoc`	The path name for the keystore.
`# oc4j.iiop.keyStorePass`	The password for the keystore.
`# oc4j.iiop.trustStoreLoc`	The path name for the truststore.
`# oc4j.iiop.trustStorePass`	The password for the truststore.
`# oc4j.iiop.enable.clientauth`	Whether the client supports client-side authentication. If this property is set to `true`, you must specify a keystore location and password.

`oc4j.iiop.ciphersuites`	Which cipher suites are to be enabled. The valid cipher suites are:
	`TLS_RSA_WITH_RC4_128_MD5`
	`SSL_RSA_WITH_RC4_128_MD5`
	`TLS_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
	`SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA`
	`TLS_RSA_EXPORT_WITH_RC4_40_MD5`
	`SSL_RSA_EXPORT_WITH_RC4_40_MD5`
	`TLS_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
	`SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`

`nameservice.useSSL`	Whether to use SSL when making the initial connection to the server.
`client.sendpassword`	Whether to send user name and password in clear form (unencrypted) in the service context when not using SSL. If this property is set to `true`, the user name and password are sent only to servers listed in the `trustedServer` list.
`oc4j.iiop.trustedServers`	A list of servers that can be trusted to receive passwords sent in clear form. Has no effect if `client.sendpassword` is set to `false`. The list is comma-separated. Each entry in the list can be an IP address, a host name, a host name pattern (for instance, `*.example.com`), or `*`; `*` alone means that all servers are trusted.

Note: The properties marked with a # can be set either in `ejb_sec.properties` or as system properties. The settings in `ejb_sec.properties` always override settings specified as system properties.

---

15 Security Tips

These hints come from the Security Best Practices document, available from Oracle Technology Network (`http://otn.oracle.com`). Check the OTN Web site for updates.

HTTPS

Oracle HTTP Server (OHS) has several features that provide security to an application without requiring you to modify the application. You should evaluate and leverage these features before coding similar features yourself.   HTTP security features include:

- Authentication — OHS can authenticate users and pass the authenticated user-id to an application in a standard manner (`REMOTE_USER`). It also supports single sign-on, thus reusing existing login mechanisms.
- Authorization — OHS has directives that can allow access to your application only if the end user is authenticated and authorized. Again, no code change is required.
- Encryption — OHS can provide transparent SSL communication to end customers without any code change on the application.

Other suggestions for securing HTTPS:

- *Configure Oracle Application Server to fail attempts to use weak encryption.* You can configure Oracle Application Server to use only specified encryption ciphers for HTTPS connections. For instance, your application could reject connections from non-128-bit client-side SSL libraries. This ability is especially useful for banks and other financial institutions because it provides server-side control of the encryption strength for each connection.
- *Use HTTPS to HTTP appliances for accelerating HTTP over SSL.* Use HTTPS everywhere you need to. However, the huge performance overhead of HTTPS forces a trade-off in some situations.

  For a relatively low cost, HTTPS-to-HTTP appliances can change throughput on a 500MHz UNIX machine from 20-30 transactions per second to 6000 transactions per second, making this trade-off decision easier.

  Moreover, these appliances provide much better solutions than adding mathematics or cryptography cards to UNIX, Windows, or Linux boxes.

  Ensure that sequential HTTPS transfers are requested through the same Web server. Expect 40 to 50 milliseconds of CPU time to initiate SSL sessions on a 500 MHz machine. Most of this CPU time is spent in the key exchange logic, where the bulk encryption key is exchanged. Caching the bulk encryption will significantly reduces CPU overhead on subsequent accesses, provided that the accesses are routed to the same Web server.

- *Keep secure pages and pages not requiring security on separate servers.* Although it may be easier to place all pages for an application on one HTTPS server, this strategy has enormous performance costs. Reserve your HTTPS server for pages needing SSL, and put the pages not needing SSL on an HTTP server.

  If secure pages are composed of many GIF, JPEG, or other files to be displayed on the same screen, it is probably not worth the effort to segregate secure from nonsecure static content. The SSL key exchange (a major consumer of CPU cycles) is likely to be called exactly once in any case, and the overhead of bulk encryption is not that high.

Overall Security

- *When assigning privileges to modules, use the lowest levels that are adequate to perform the modules' function(s)*. Using low-level privileges provides "fault containment": if security is compromised, it is contained within a small area of the network and cannot invade the entire intranet.
- *Tune the SSLSessionCacheTimeout directive if you are using SSL.* The Apache server in OracleAS caches a client's SSL session information by default. With session caching, only the first connection to the server incurs high latency.

  In a simple test to connect and disconnect to an SSL-enabled server, the elapsed time for 5 connections was 11.4 seconds without SSL session caching; with session caching enabled, the elapsed time was 1.9 seconds.

  The default `SSLSessionCacheTimeout` is 300 seconds. Note that the duration of an SSL session is unrelated to the use of HTTP persistent connections. You can change the `SSLSessionCacheTimeout` directive in `httpd.conf` file to meet your application needs.

JAAS

- *Migrate your user management from principals.xml to the JAAS Provider.* In earlier releases of OracleAS, the J2EE application server component stored all user information in a file called `principals.xml` (including storing passwords in cleartext). The OracleAS JAAS Provider provides a similar simple security model as a default, without storing passwords in cleartext. The JAAS Provider also offers tight integration with OracleAS Infrastructure (including SSO and OID) out of the box.
- *Avoid writing custom user managers; instead, extend the JAAS Provider, SSO, and OID.* The Oracle Application Server Containers for J2EE (OC4J) container

continues to supply several methods and levels of extending security providers. Although you can extend the `UserManager` class to build a custom user manager, leveraging the rich functionality provided by the JAAS Provider, SSO, and OID gives you more time to focus on actual business logic instead of infrastructure code. Both SSO and OID provide APIs to integrate with external authentication servers and directories respectively.

- *Use SSO as the authentication mechanism with the JAAS Provider.* The JAAS Provider allows different authentication options. However, we strongly recommend leveraging the SSO server whenever possible because:
  - It is the default mechanism for most OracleAS components, such as Portal, Forms, Reports, Wireless, and so on.
  - It is easy to set up in a declarative fashion and does not require any custom programming.
  - It provides seamless PKI integration.
- *Use the JAAS Provider's declarative features to reduce programming.* Because most of the features in the OracleAS JAAS Provider are controlled declaratively, particularly in the area of authentication, developers can postpone setup until deployment time. This not only reduces the programming tasks for integrating a JAAS based application, it allows the deployer to use environment-specific security models for that application.
- *Use the fine-grained access control offered by the JAAS Provider and the Java permission model.* Unlike the "coarse-grained" J2EE authorization model as it exists today, the JAAS Provider integrated with OC4J allows any protected resource to be modeled using Java permissions. The Java permission model (and associated Permission class) is extensible and allows a flexible way to define fine-grained access control.
- *Use OID as the central repository for the JAAS Provider in production environments.* Although the JAAS Provider supports a flat-file XML-based repository useful for development and testing environments, it should be configured to use OID for production environments. OID provides LDAP standard features for modeling administrative metadata and is built on the Oracle database platform, inheriting all the database properties of scalability, reliability, manageability, and performance.
- *Take advantage of the authorization features of the JAAS Provider.* In addition to the authorization functionality defined in the JAAS 1.0 specification, the OracleAS JAAS Provider supports:
  - Hierarchical, role-based access control (RBAC)

- The ability to partition security policy by subscriber (that is, each user community)

These extensions provide a more scalable and manageable framework for security policies covering a large user population.

A JAAS Provider Standards and Samples

This appendix provides supplemental samples and standards.

This appendix contains these topics:

- Sample jazn-data.xml Code
- Supplemental Code Samples

Sample jazn-data.xml Code

This section presents a sample `jazn-data.xml` file which illustrates the specific standards that XML files must conform to. This `jazn-data.xml` file contains a realm, `jazn.com`, users (two with obfuscated passwords) and roles.

See Also:

- "Realm Management in XML-Based Environments" on page 4-5.
- "Realm and Policy Management" on page 4-2 for further information on managing the JAAS Provider in XML-based provider environment

Example A–1 Sample jazn-data.xml File

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<!DOCTYPE jazn-data PUBLIC "JAZN-XML Data"
"http://xmlns.oracle.com/ias/dtds/jazn-data.dtd">
<jazn-data>


<!-- JAZN Realm Data -->
<jazn-realm>
  <realm>
    <name>jazn.com</name>
    <users>
      <user>
        <name>SCOTT</name>
        <display-name>SCOTT</display-name>
        <credentials>{903}oZZYqmGc/iyCaDrD4qs2FHbXf3LAWtMN</credentials>
      </user>
      <user>
        <name>admin</name>
        <display-name>OC4J Administrator</display-name>
        <description>OC4J Administrator</description>
        <credentials>{903}FVb95KHGyzR9MkAS2Ru/72P/Ol6eOsQD</credentials>
      </user>
      <user>
        <name>anonymous</name>
        <description>The default guest/anonymous user</description>
      </user>
      <user>
        <name>pwForScott</name>
        <description>Password for database user Scott</description>
        <credentials>{903}pjbjHNP53w3haB3ygstBpsglEhQJ1dnN</credentials>
      </user>
      <user>
        <name>user</name>
        <description>The default user</description>
        <credentials>{903}Zg4KSjPqwZ6FGsCWbxiFSJpPFJNrq9Ww</credentials>
      </user>
      <user>
        <name>pwForSSL</name>
        <description>Password for ssl key and trust stores</description>
        <credentials>{903}uMg+4/e5znCrcQSH36NjbrkpHdgC6oMh</credentials>
      </user>
      <user>
        <name>pwForSystem</name>
        <description>Password for database system user </description>
        <credentials>{903}IUHuvYYGY5R9trDfQp7qY//livlqHjVV</credentials>
      </user>
    </users>
    <roles>
      <role>
        <name>administrators</name>
        <display-name>Realm Admin Role</display-name>
        <description>Administrative role for this realm.</description>
        <members>
          <member>
            <type>user</type>
            <name>admin</name>
          </member>
        </members>
      </role>
      <role>
        <name>jmxusers</name>
        <display-name>JMX users</display-name>
        <description>Allows access to application level user defined
MBeans</description>
        <members>
        </members>
      </role>
      <role>
        <name>users</name>
        <members>
          <member>
            <type>user</type>
            <name>user</name>
          </member>
          <member>
            <type>user</type>
            <name>SCOTT</name>
          </member>
          <member>
            <type>role</type>
            <name>administrators</name>
          </member>
        </members>
      </role>
      <role>
        <name>guests</name>
        <members>
          <member>
            <type>user</type>
            <name>anonymous</name>
          </member>
          <member>
            <type>role</type>
            <name>users</name>
          </member>
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>


<!-- JAZN Policy Data -->
<jazn-policy>
  <grant>
    <grantee>
      <principals>
        <principal>
          <realm-name>jazn.com</realm-name>
          <type>role</type>
          <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
          <name>jazn.com/jmxusers</name>
        </principal>
      </principals>
    </grantee>
    <permissions>
      <permission>
        <class>com.evermind.server.rmi.RMIPermission</class>
        <name>login</name>
```

```
          </permission>
        </permissions>
      </grant>
      <grant>
        <grantee>
          <principals>
            <principal>
              <realm-name>jazn.com</realm-name>
              <type>role</type>
              <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
              <name>jazn.com/users</name>
            </principal>
          </principals>
        </grantee>
        <permissions>
          <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
          </permission>
        </permissions>
      </grant>
      <grant>
        <grantee>
          <principals>
            <principal>
              <realm-name>jazn.com</realm-name>
              <type>role</type>
              <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
              <name>jazn.com/administrators</name>
            </principal>
          </principals>
        </grantee>
        <permissions>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>

    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$createrealm</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>dropuser</actions>
          </permission>
          <permission>
            <class>com.evermind.server.AdministrationPermission</class>
            <name>administration</name>
            <actions>administration</actions>
          </permission>
          <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>modifyrealmmetadata</actions>
          </permission>
          <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>

    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$createrole</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.RoleAdminPermission</class>
            <name>jazn.com/*</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>createrealm</actions>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>

    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$droprole</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>

    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$droprealm</name>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>

    <name>oracle.security.jazn.realm.RealmPermission$jazn.com$modifyrealmmetadata</n
    ame>
          </permission>
          <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>droprealm</actions>
          </permission>
          <permission>
            <class>oracle.security.jazn.policy.AdminPermission</class>
            <name>oracle.security.jazn.policy.RoleAdminPermission$jazn.com/*$</name>
          </permission>
        </permissions>
      </grant>
    </jazn-policy>


    <!-- Permission Class Data -->
    <jazn-permission-classes>
    </jazn-permission-classes>


    <!-- Principal Class Data -->
    <jazn-principal-classes>
    </jazn-principal-classes>


    <!-- Login Module Data -->
    <jazn-loginconfig>
      <application>
        <name>oracle.security.jazn.tools.Admintool</name>
        <login-modules>
          <login-module>
            <class>oracle.security.jazn.realm.RealmLoginModule</class>
            <control-flag>required</control-flag>
            <options>
              <option>
                <name>debug</name>
                <value>false</value>
              </option>
              <option>
                <name>addAllRoles</name>
                <value>true</value>
              </option>
            </options>
          </login-module>
        </login-modules>
      </application>
      <application>
        <name>oracle.security.jazn.oc4j.JAZNUserManager</name>
        <login-modules>
```

```
      <login-module>
        <class>oracle.security.jazn.realm.RealmLoginModule</class>
        <control-flag>required</control-flag>
        <options>
          <option>
            <name>addAllRoles</name>
            <value>true</value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>

</jazn-data>
```

Supplemental Code Samples

The following code samples are intended as supplemental information. This section presents the following:

- Supplementary Code Sample: Creating an Application Realm
- Supplementary Code Sample: Modifying User Permissions

Supplementary Code Sample: Creating an Application Realm

The following code sample creates an Application Realm with the objects shown in Table A–1. The objects to be modified are presented in bold.

Table A–1 Objects In Sample Application Realm Creation Code

Objects	Names
sample organization	`dev.com`
`adminUser` (optional)	`John.Singh`
`adminRole`	`administrator`
sample realm name	`devRealm`

Example A–2 Application Realm Creation Code

```
import oracle.security.jazn.spi.ldap.*;
import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;

import java.util.*;

/**
 * Creates an application realm.
 */

public class CreateRealm extends Object
{
    public CreateRealm() {};

    public static void main (String[] args) {
      CreateRealm test = new CreateRealm();
      test.createAppRealm();
    }

    void createAppRealm() {
    Realm realm=null;


 try {
     Hashtable prop = new Hashtable();
     prop.put(Realm.LDAPProperty.USERS_SEARCHBASE,"cn=users,o=dev.com");
```

```
          // specifying the following LDAP directory object class
            // is optional.  When specified, it will
          // be used as a filter to search for users
          prop.put(Realm.LDAPProperty.USERS_OBJ_CLASS,"orclUser");

          // adminUser is optional
         String adminUser = "John.Singh";

          String adminRole = "administrator";

          RealmManager realmMgr = JAZNContext.getRealmManager();

          InitRealmInfo realmInfo = new
          InitRealmInfo(InitRealmInfo.RealmType.APPLICATION_REALM, adminUser,
          adminRole, prop);
          realm = realmMgr.createRealm("devRealm", realmInfo);
          }

     catch (Exception e) {
          e.printStackTrace();
         }
       }
     }
```

Supplementary Code Sample: Modifying User Permissions

Example A–3 demonstrates granting `java.io.FilePermission` to a user named `Jane.Smith`. The objects to be modified are presented in bold.

Table A–2 lists the objects in Example A–3.

Table A–2 Objects In Sample Modifying User Permissions Code

Objects	Names	Comments
`RealmUser user`	`Jane.Smith`	
`codesource cs`	`file:/home/task.jar`	
`File path`	`report.data`	Path is the pathname of the file.
sample organization	`abc.com`	`abc.com` does not appear in this code directly.
sample External Realm	`abcRealm`	

Example A–3 Modifying User Permissions Code

Code Sample

```
import oracle.security.jazn.*;
import oracle.security.jazn.policy.*;
import oracle.security.jazn.realm.*;
import java.lang.*;
import java.security.*;
import java.util.*;
import java.net.*;
import java.io.*;

public class Init {

    public static void main(String[] args) {

    try {
      RealmManager realmMgr = JAZNContext.getRealmManager();
            Realm realm = realmMgr.getRealm("abcRealm");
            UserManager userMgr = realm.getUserManager();
            RoleManager roleMgr = realm.getRoleManager();
            final JAZNPolicy policy = JAZNContext.getPolicy();

            final RealmUser user = userMgr.getUser("Jane.Smith");

            AccessController.doPrivileged (new PrivilegedAction() {
                    public Object run() {

                try {
                  CodeSource cs = new CodeSource(new URL("
                            file:/home/task.jar"), null);
                    HashSet prop = new HashSet();
                    prop.add((Principal) user);

                    // assign permission to principals
                    policy.grant(new Grantee(prop, cs), new
                             FilePermission("report.data", "read"));

                    return null;
                        } catch (JAZNException e1) {
                            e1.printStackTrace();
                        } catch (java.net.MalformedURLException e2) {
                            e2.printStackTrace();
                        }
                    return null;
                    }
                }
             );

        } catch (JAZNException e) {
            e.printStackTrace();
        }
    }
}
```

Discussion Of Sample Code

The sample code shown in Example A–3 grants a user, `Jane.Smith`, permission to use the sample application, `AccessTest1` as follows:

The name `cs` is assigned to the `file:/home/task.jar`, which includes the sample application `AccessTest1`:

```
CodeSource cs = new CodeSource(new URL("
                            file:/home/task.jar"), null);
```

`Jane.Smith` is the user added to the hashset prop:

```
HashSet prop = new HashSet();
                    prop.add((Principal) user);
```

`Jane.Smith` is granted permission, on the `Codesource cs`, to read the file `report.data.`

```
policy.grant(new Grantee(prop, cs), new
                             FilePermission("report.data", "read"));
```

B JAAS Provider Schemas

This appendix gives the XSD (XML Schema) specifications for the `jazn-data.xml` file used by the JAAS XML-based Provider and the `jazn.xml` file used to configure both JAAS providers.

Schema for jazn-data.xml

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" >
  <xsd:element name="jazn-data" type="JAZN-DATAjazn-dataType"/>
  <xsd:complexType name="JAZN-DATAjazn-dataType">
    <xsd:sequence >
      <xsd:element name="jazn-realm" type="JAZN-DATAjazn-realmType"
minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-policy" type="JAZN-DATAjazn-policyType"
minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-permission-classes"
type="JAZN-DATAjazn-permission-classesType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-principal-classes"
type="JAZN-DATAjazn-principal-classesType" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="jazn-loginconfig" type="JAZN-DATAjazn-loginconfigType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-realmType">
    <xsd:sequence>
      <xsd:element name="realm" type="JAZN-DATArealmType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATArealmType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="users" type="JAZN-DATAusersType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="roles" type="JAZN-DATArolesType" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="jazn-policy" type="JAZN-DATAjazn-policyType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAusersType">
    <xsd:sequence>
      <xsd:element name="user" type="JAZN-DATAuserType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAuserType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="credentials" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATArolesType">
    <xsd:sequence>
      <xsd:element name="role" type="JAZN-DATAroleType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAroleType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="members" type="JAZN-DATAmembersType" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAmembersType">
```

```
    <xsd:sequence>
      <xsd:element name="member" type="JAZN-DATAmemberType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAmemberType">
    <xsd:sequence >
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-policyType">
    <xsd:sequence>
      <xsd:element name="grant" type="JAZN-DATAgrantType" minOccurs="0"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>


<!-- no way to define key attribute for grant type -->
  <xsd:complexType name="JAZN-DATAgrantType">
    <xsd:sequence >
      <xsd:element name="grantee" type="JAZN-DATAgranteeType" />
      <xsd:element name="permissions" type="JAZN-DATApermissionsType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="grantee-names" type="xsd:string" use="default"
value=""/>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed"
value="grantee-names"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAgranteeType">
    <xsd:sequence >
      <xsd:element name="display-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="principals" type="JAZN-DATAprincipalsType"
minOccurs="0" maxOccurs="1"/>
      <xsd:element name="codesource" type="JAZN-DATAcodesourceType"
minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipalsType">
    <xsd:sequence>
      <xsd:element name="principal" type="JAZN-DATAprincipalType" minOccurs="0"
```

```
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipalType">
    <xsd:sequence >
      <xsd:element name="realm-name" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" minOccurs="0" maxOccurs="1"/>
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAcodesourceType">
    <xsd:sequence>
      <xsd:element name="url" type="xsd:string" />
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermissionsType">
    <xsd:sequence>
      <xsd:element name="permission" type="JAZN-DATApermissionType"
minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermissionType">
    <xsd:sequence >
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="actions" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-permission-classesType">
    <xsd:sequence>
      <xsd:element name="permission-class" type="JAZN-DATApermission-classType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATApermission-classType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="target-descriptors"
type="JAZN-DATAtarget-descriptorsType" />
      <xsd:element name="action-descriptors"
type="JAZN-DATAaction-descriptorsType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAtarget-descriptorsType">
    <xsd:sequence>
      <xsd:element name="target-descriptor"
type="JAZN-DATAtarget-descriptorType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAtarget-descriptorType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAaction-descriptorsType">
    <xsd:sequence>
      <xsd:element name="action-descriptor"
type="JAZN-DATAaction-descriptorType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAaction-descriptorType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-principal-classesType">
    <xsd:sequence>
      <xsd:element name="principal-class" type="JAZN-DATAprincipal-classType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAprincipal-classType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
      <xsd:element name="type" type="xsd:string" />
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="name-description-map"
type="JAZN-DATAname-description-mapType" minOccurs="0" maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAname-description-mapType">
    <xsd:sequence>
      <xsd:element name="name-description-pair"
type="JAZN-DATAname-description-pairType" minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAname-description-pairType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="description" type="xsd:string" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAjazn-loginconfigType">
    <xsd:sequence>
      <xsd:element name="application" type="JAZN-DATAapplicationType"
minOccurs="0" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAapplicationType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="login-modules" type="JAZN-DATAlogin-modulesType" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAlogin-modulesType">
    <xsd:sequence>
      <xsd:element name="login-module" type="JAZN-DATAlogin-moduleType"
minOccurs="1" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAlogin-moduleType">
    <xsd:sequence >
      <xsd:element name="class" type="xsd:string" />
      <xsd:element name="control-flag" type="xsd:string" />
      <xsd:element name="options" type="JAZN-DATAoptionsType" minOccurs="0"
maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="class
control-flag"/>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAoptionsType">
    <xsd:sequence>
      <xsd:element name="option" type="JAZN-DATAoptionType" minOccurs="1"
maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
  <xsd:complexType name="JAZN-DATAoptionType">
    <xsd:sequence >
      <xsd:element name="name" type="xsd:string" />
      <xsd:element name="value" type="xsd:string" />
    </xsd:sequence>
    <xsd:attribute name="Key" type="JAZN-DATAkeyset" use="fixed" value="name
value"/>
  </xsd:complexType>



  <xsd:simpleType name="JAZN-DATAkeyset">
    <xsd:list itemType="xsd:string"/>
  </xsd:simpleType>




</xsd:schema>
```

Schema for jazn.xml

```
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" >
<xsd:element name="jazn" type="jaznType"/>
<xsd:complexType name="jaznType">
<xsd:sequence >
<xsd:element name="property" type="jaznPropertyType" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element name="jazn-web-app" type="jazn-web-appType" minOccurs="0"
maxOccurs="1"/>
```

```
</xsd:sequence>
<xsd:attribute name="provider" type="xsd:string" use="optional"/>
<xsd:attribute name="location" type="xsd:string" use="optional"/>
<xsd:attribute name="default-realm" type="xsd:string" use="optional"/>
<xsd:attribute name="persistence" type="xsd:string" use="optional"/>
<xsd:attribute name="config" type="xsd:string" use="optional"/>
</xsd:complexType>
<xsd:complexType name="jazn-web-appType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="auth-method" type="xsd:string" use="optional"/>
<xsd:attribute name="runas-mode" type="runas-modeType" use="default"
value="false"/>
<xsd:attribute name="doasprivileged-mode" type="doasprivileged-modeType"
use="default" value="true"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>
<xsd:simpleType name="runas-modeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="true"/>
<xsd:enumeration value="false"/>
</xsd:restriction>
</xsd:simpleType>
<xsd:simpleType name="doasprivileged-modeType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="true"/>
<xsd:enumeration value="false"/>
</xsd:restriction>
</xsd:simpleType>

<xsd:complexType name="jaznPropertyType">
<xsd:simpleContent>
<xsd:extension base="xsd:string">
<xsd:attribute name="Key" type="jaznKeyset" use="fixed" value="name"/>
<xsd:attribute name="name" type="xsd:string" use="optional"/>
<xsd:attribute name="value" type="xsd:string" use="optional"/>
</xsd:extension>
</xsd:simpleContent>
</xsd:complexType>

<xsd:simpleType name="jaznKeyset">
<xsd:list itemType = "xsd:string"/>
</xsd:simpleType>
</xsd:schema>
```

```
<!ENTITY % CHARSET "CDATA">
<!ENTITY % WEBPATH "CDATA">
<!ENTITY % NUMBER "CDATA">
<!ENTITY % HOST "CDATA">
<!ENTITY % PATH "CDATA">
<!ENTITY % CLASSNAME "CDATA">
<!-- A group that this security-role-mapping implies. Ie all the members of the
specified group are included in this role. -->
<!ELEMENT group (#PCDATA)>
<!ATTLIST group name CDATA #IMPLIED
>
<!-- An attribute sent to the context. The only mandatory attribute in JNDI is
the 'java.naming.factory.initial' which is the classname of the context factory
implementation. -->
<!ELEMENT context-attribute (#PCDATA)>
<!ATTLIST context-attribute name CDATA #IMPLIED
value CDATA #IMPLIED
>
<!-- Defines the relative/absolute path to a file containing mime-mappings to
use. -->
<!ELEMENT mime-mappings (#PCDATA)>
<!ATTLIST mime-mappings path CDATA #IMPLIED
>
<!-- Specifies a codebase where classes used by this application
(servlets/beans, etc) can be found. -->
<!ELEMENT classpath (#PCDATA)>
<!ATTLIST classpath path CDATA #REQUIRED
>
<!-- The specification of an optional javax.naming.Context implementation used
for retrieving the resource. This is useful when hooking up with 3rd party
modules, such as a 3rd party JMS server for instance. Either use the context
implementation supplied by the resource vendor or if none exists write an
implementation which in turn negotiates with the vendor software. -->
<!ELEMENT lookup-context (context-attribute+)>
<!ATTLIST lookup-context location CDATA #IMPLIED
>
<!-- Specifies a servlet to use as request-tracker, request-trackers are invoked
for every request and are useful for logging purposes etc. -->
<!ELEMENT request-tracker (#PCDATA)>
<!ATTLIST request-tracker servlet-name CDATA #IMPLIED
>
<!-- The resource-ref element is used for the declaration of a reference to an
external resource such as a datasource, JMS queue, mail session or similar.
The resource-ref-mapping ties this to a JNDI-location when deploying. -->
<!ELEMENT resource-ref-mapping (lookup-context?)>
```

```
<!ATTLIST resource-ref-mapping location CDATA #IMPLIED
name CDATA #REQUIRED
>
<!-- Tag that is defined if the application is to be clustered. Clustered
applications have their ServletContext and session data
shared between the apps in the cluster, the values have to be either
Serializable or be remote RMI-objects (implement java.rmi.Remote). -->
<!ELEMENT cluster-config (#PCDATA)>
<!ATTLIST cluster-config host %HOST; "230.0.0.1"
id CDATA "based on local IP"
port %NUMBER; "9127"
>
<!-- Specifies an optional access-mask for this application, hostnames and
ip/subnets can be used to filter out allowed clients of this application. -->
<!ELEMENT access-mask (host-access*, ip-access*)>
<!ATTLIST access-mask default (allow|deny) "allow"
>
<!-- Overrides the value of an env-entry in the assembly descriptor. It is used
to keep the .ear (assembly) clean from deployment-specific values. The body is
the value. -->
<!ELEMENT env-entry-mapping (#PCDATA)>
<!ATTLIST env-entry-mapping name CDATA #IMPLIED
>

<!-- Specifies the Expires setting for a given set of resources, useful for
caching policies (for instance for browsers not to reload images as frequently
as documents etc). -->
<!ELEMENT expiration-setting (#PCDATA)>
<!ATTLIST expiration-setting expires CDATA #IMPLIED
url-pattern CDATA #IMPLIED
>
<!-- Overrides the value of a context-param in the assembly descriptor. It is
used to keep the .ear (assembly) clean from deployment-specific values. The body
is the value. -->
<!ELEMENT context-param-mapping (#PCDATA)>
<!ATTLIST context-param-mapping name CDATA #IMPLIED
>
<!-- Session-tracking settings for this application. -->
<!ELEMENT session-tracking (session-tracker*)>
<!ATTLIST session-tracking autoencode-absolute-urls (true|false) "false"
autoencode-urls (true|false) "true"
autojoin-session (true|false) "false"
cookie-domain CDATA #IMPLIED
cookie-max-age %NUMBER; "in memory only"
cookies (enabled|disabled) "enabled"
```

```
>
<!-- A user that this security-role-mapping implies. -->
<!ELEMENT user (#PCDATA)>
<!ATTLIST user name CDATA #IMPLIED
>
<!-- Adds a virtual directory mapping, used to include files that doesnt
physically reside below the document-root among the web-exposed files. -->
<!ELEMENT virtual-directory (#PCDATA)>
<!ATTLIST virtual-directory real-path %PATH; #IMPLIED
virtual-path %PATH; #IMPLIED
>
<!-- Specifies an ip/netmask who is allowed access. -->
<!ELEMENT ip-access (#PCDATA)>
<!ATTLIST ip-access ip CDATA #REQUIRED
mode (allow|deny) #REQUIRED
netmask CDATA #IMPLIED
>
<!-- Specifies a servlet to use as chainer for a specified mime-type. Useful to
filter/transform certain kinds of output. -->
<!ELEMENT servlet-chaining (#PCDATA)>
<!ATTLIST servlet-chaining mime-type CDATA #IMPLIED
servlet-name CDATA #IMPLIED
>
<!-- Specifies a domain or netmask who is allowed access. -->
<!ELEMENT host-access (#PCDATA)>
<!ATTLIST host-access domain CDATA #REQUIRED
mode (allow|deny) #REQUIRED
>
<!-- The ejb-ref element is used for the declaration of a reference to
another enterprise bean's home. The ejb-ref-mapping ties this to a JNDI-location
when deploying. -->
<!ELEMENT ejb-ref-mapping (#PCDATA)>
<!ATTLIST ejb-ref-mapping location CDATA #IMPLIED
name CDATA #REQUIRED
>
<!-- The runtime mapping (to groups and users) of a role. Maps to a
security-role of the same name in the assembly descriptor. -->
<!ELEMENT security-role-mapping (group*, user*)>
<!ATTLIST security-role-mapping impliesAll CDATA #IMPLIED
name CDATA #IMPLIED
>
<!-- Specifies a servlet to use as session-tracker, session-trackers are invoked
as soon as a session is created and are useful for logging purposes etc. -->
<!ELEMENT session-tracker (#PCDATA)>
<!ATTLIST session-tracker servlet-name CDATA #IMPLIED
```

```
>
<!-- JAZN configuration -->
<!ELEMENT jazn-web-app (#PCDATA)>
<!ATTLIST jazn-web-app auth-method CDATA #IMPLIED
runas-mode (true | false) "false"
doasprivileged-mode (true | false) "true"
>
<!-- Web-app class loader configuration -->
<!ELEMENT web-app-class-loader EMPTY>
<!ATTLIST web-app-class-loader
search-local-classes-first (true | false) "false"
include-war-manifest-class-path (true | false) "true"
>
<!--
Copyright (c) 2000 Sun Microsystems, Inc.,
901 San Antonio Road,
Palo Alto, California 94303, U.S.A.
All rights reserved.
Sun Microsystems, Inc. has intellectual property rights relating to
technology embodied in the product that is described in this document.
In particular, and without limitation, these intellectual property
rights may include one or more of the U.S. patents listed at
http://www.sun.com/patents and one or more additional patents or
pending patent applications in the U.S. and in other countries.
This document and the product to which it pertains are distributed
under licenses restricting their use, copying, distribution, and
decompilation. This Product or document may be reproduced but may not
be changed without prior written authorization of Sun and its
licensors, if any.
Third-party software, including font technology, is copyrighted and
licensed from Sun suppliers.
Sun, Sun Microsystems, the Sun logo, Java, JavaServer Pages, Java
Naming and Directory Interface, JDBC, JDK, JavaMail and and
Enterprise JavaBeans are trademarks or registered trademarks of Sun
Microsystems, Inc. in the U.S. and other countries.
Federal Acquisitions: Commercial Software - Government Users Subject to
Standard License Terms and Conditions.
DOCUMENTATION IS PROVIDED "AS IS" AND ALL EXPRESS OR IMPLIED
CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED
WARRANTY OF MERCHANTABILITY, FITNESS FOR FOR A PARTICULAR PURPOSE OR
NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT SUCH
DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.
```

```
Copyright (c) 2000 Sun Microsystems, Inc.,
901 San Antonio Road,
Palo Alto, California 94303, E'tats-Unis.
Tous droits re'serve's.
Sun Microsystems, Inc. a les droits de proprie'te' intellectuels
relatants a` la technologie incorpore'e dans le produit qui est de'crit
dans ce document. En particulier, et sans la limitation, ces droits de
proprie'te' intellectuels peuvent inclure un ou plus des brevets
ame'ricains e'nume're's a` http://www.sun.com/patents et un ou les
brevets plus supple'mentaires ou les applications de brevet en attente
dans les E'tats-Unis et dans les autres pays.
Ce produit ou document est prote'ge' par un copyright et distribue'
avec des licences qui en restreignent l'utilisation, la copie, la
distribution, et la de'compilation. Ce produit sa documention associe
n peut e^tre reproduite, par quelque moyen que ce soit, sans
l'autorisation pre'alable et e'crite de Sun et de ses bailleurs de
licence, le cas e'che'ant.
Le logiciel de'tenu par des tiers, et qui comprend la technologie
relative aux polices de caracte`res, est prote'ge' par un copyright et
licencie' par des fournisseurs de Sun.
Sun, Sun Microsystems, le logo Sun, Java, JavaServer Pages, Java
Naming and Directory Interface, JDBC, JDK, JavaMail et and
Enterprise JavaBeans sont des marques de fabrique ou des marques
de'pose'es de Sun Microsystems, Inc. aux E'tats-Unis et dans d'autres
pays.
LA DOCUMENTATION EST FOURNIE "EN L'E'TAT" ET TOUTES AUTRES CONDITIONS,
DECLARATIONS ET GARANTIES EXPRESSES OU TACITES SONT FORMELLEMENT
EXCLUES, DANS LA MESURE AUTORISEE PAR LA LOI APPLICABLE, Y COMPRIS
NOTAMMENT TOUTE GARANTIE IMPLICITE RELATIVE A LA QUALITE MARCHANDE, A
L'APTITUDE A UNE UTILISATION PARTICULIERE OU A L'ABSENCE DE
CONTREFAC,ON.
-->
<!--
This is the XML DTD for the Servlet 2.3 deployment descriptor.
All Servlet 2.3 deployment descriptors must include a DOCTYPE
of the following form:
<!DOCTYPE web-app PUBLIC
"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
"http://java.sun.com/dtd/web-app_2_3.dtd">
-->
<!--
The following conventions apply to all J2EE deployment descriptor
elements unless indicated otherwise.
- In elements that contain PCDATA, leading and trailing whitespace
in the data may be ignored.
- In elements whose value is an "enumerated type", the value is
```

```
case sensitive.
- In elements that specify a pathname to a file within the same
JAR file, relative filenames (i.e., those not starting with "/")
are considered relative to the root of the JAR file's namespace.
Absolute filenames (i.e., those starting with "/") also specify
names in the root of the JAR file's namespace. In general, relative
names are preferred. The exception is .war files where absolute
names are preferred for consistency with the servlet API.
-->

<!--
The web-app element is the root of the deployment descriptor for
a web application.
-->
<!ELEMENT web-app (icon?, display-name?, description?, distributable?,
context-param*, filter*, filter-mapping*, listener*, servlet*,
servlet-mapping*, session-config?, mime-mapping*, welcome-file-list?,
error-page*, taglib*, resource-env-ref*, resource-ref*, security-constraint*,
login-config?, security-role*, env-entry*, ejb-ref*, ejb-local-ref*)>
<!--
The auth-constraint element indicates the user roles that should
be permitted access to this resource collection. The role-name
used here must either correspond to the role-name of one of the
security-role elements defined for this web application, or be
the specially reserved role-name "*" that is a compact syntax for
indicating all roles in the web application. If both "*" and
rolenames appear, the container interprets this as all roles.
If no roles are defined, no user is allowed access to the portion of
the web application described by the containing security-constraint.
The container matches role names case sensitively when determining
access.

Used in: security-constraint
-->
<!ELEMENT auth-constraint (description?, role-name*)>
<!--
The auth-method element is used to configure the authentication
mechanism for the web application. As a prerequisite to gaining access to any
web resources which are protected by an authorization
constraint, a user must have authenticated using the configured
mechanism. Legal values for this element are "BASIC", "DIGEST",
"FORM", or "CLIENT-CERT".
Used in: login-config
-->
<!ELEMENT auth-method (#PCDATA)>
```

```
<!--
The context-param element contains the declaration of a web
application's servlet context initialization parameters.
Used in: web-app
-->
<!ELEMENT context-param (param-name, param-value, description?)>
<!--
The description element is used to provide text describing the parent
element. The description element should include any information that
the web application war file producer wants to provide to the consumer of
the web application war file (i.e., to the Deployer). Typically, the tools
used by the web application war file consumer will display the description
when processing the parent element that contains the description.
Used in: auth-constraint, context-param, ejb-local-ref, ejb-ref,
env-entry, filter, init-param, resource-env-ref, resource-ref, run-as,
security-role, security-role-ref, servlet, user-data-constraint,
web-app, web-resource-collection
-->
<!ELEMENT description (#PCDATA)>
<!--
The display-name element contains a short name that is intended to be
displayed by tools. The display name need not be unique.
Used in: filter, security-constraint, servlet, web-app
Example:
<display-name>Employee Self Service</display-name>
-->
<!ELEMENT display-name (#PCDATA)>
<!--
The distributable element, by its presence in a web application
deployment descriptor, indicates that this web application is
programmed appropriately to be deployed into a distributed servlet
container
Used in: web-app
-->
<!ELEMENT distributable EMPTY>
<!--
The ejb-link element is used in the ejb-ref or ejb-local-ref
elements to specify that an EJB reference is linked to an
enterprise bean.
The name in the ejb-link element is composed of a
path name specifying the ejb-jar containing the referenced enterprise
bean with the ejb-name of the target bean appended and separated from
the path name by "#". The path name is relative to the war file
containing the web application that is referencing the enterprise bean.
This allows multiple enterprise beans with the same ejb-name to be
```

```
uniquely identified.
Used in: ejb-local-ref, ejb-ref
Examples:
<ejb-link>EmployeeRecord</ejb-link>
<ejb-link>../products/product.jar#ProductEJB</ejb-link>
-->
<!ELEMENT ejb-link (#PCDATA)>
<!--
The ejb-local-ref element is used for the declaration of a reference to
an enterprise bean's local home. The declaration consists of:
- an optional description
- the EJB reference name used in the code of the web application
that's referencing the enterprise bean
- the expected type of the referenced enterprise bean
- the expected local home and local interfaces of the referenced
enterprise bean
- optional ejb-link information, used to specify the referenced
enterprise bean
Used in: web-app
-->
<!ELEMENT ejb-local-ref (description?, ejb-ref-name, ejb-ref-type,
local-home, local, ejb-link?)>
<!--
The ejb-ref element is used for the declaration of a reference to
an enterprise bean's home. The declaration consists of:
- an optional description
- the EJB reference name used in the code of
the web application that's referencing the enterprise bean
- the expected type of the referenced enterprise bean
- the expected home and remote interfaces of the referenced
enterprise bean
- optional ejb-link information, used to specify the referenced
enterprise bean
Used in: web-app
-->
<!ELEMENT ejb-ref (description?, ejb-ref-name, ejb-ref-type,
home, remote, ejb-link?)>
<!--
The ejb-ref-name element contains the name of an EJB reference. The
EJB reference is an entry in the web application's environment and is
relative to the java:comp/env context. The name must be unique
within the web application.
It is recommended that name is prefixed with "ejb/".
Used in: ejb-local-ref, ejb-ref
Example:
```

```
<ejb-ref-name>ejb/Payroll</ejb-ref-name>
-->
<!ELEMENT ejb-ref-name (#PCDATA)>
<!--
The ejb-ref-type element contains the expected type of the
referenced enterprise bean.
The ejb-ref-type element must be one of the following:
<ejb-ref-type>Entity</ejb-ref-type>
<ejb-ref-type>Session</ejb-ref-type>
Used in: ejb-local-ref, ejb-ref
-->
<!ELEMENT ejb-ref-type (#PCDATA)>
<!--
The env-entry element contains the declaration of a web application's
environment entry. The declaration consists of an optional
description, the name of the environment entry, and an optional
value. If a value is not specified, one must be supplied
during deployment.
-->
<!ELEMENT env-entry (description?, env-entry-name, env-entry-value?,
env-entry-type)>
<!--
The env-entry-name element contains the name of a web applications's
environment entry. The name is a JNDI name relative to the
java:comp/env context. The name must be unique within a web application.
Example:
<env-entry-name>minAmount</env-entry-name>
Used in: env-entry
-->
<!ELEMENT env-entry-name (#PCDATA)>
<!--
The env-entry-type element contains the fully-qualified Java type of
the environment entry value that is expected by the web application's
code.
The following are the legal values of env-entry-type:
java.lang.Boolean
java.lang.Byte
java.lang.Character
java.lang.String
java.lang.Short
java.lang.Integer
java.lang.Long
java.lang.Float
java.lang.Double
Used in: env-entry
```

```
-->
<!ELEMENT env-entry-type (#PCDATA)>
<!--
The env-entry-value element contains the value of a web application's
environment entry. The value must be a String that is valid for the
constructor of the specified type that takes a single String
parameter, or for java.lang.Character, a single character.
Example:
<env-entry-value>100.00</env-entry-value>
Used in: env-entry
-->
<!ELEMENT env-entry-value (#PCDATA)>
<!--
The error-code contains an HTTP error code, ex: 404
Used in: error-page
-->
<!ELEMENT error-code (#PCDATA)>
<!--
The error-page element contains a mapping between an error code
or exception type to the path of a resource in the web application
Used in: web-app
-->
<!ELEMENT error-page ((error-code | exception-type), location)>
<!--
The exception type contains a fully qualified class name of a
Java exception type.
Used in: error-page
-->
<!ELEMENT exception-type (#PCDATA)>
<!--
The extension element contains a string describing an
extension. example: "txt"
Used in: mime-mapping
-->
<!ELEMENT extension (#PCDATA)>
<!--
Declares a filter in the web application. The filter is mapped to
either a servlet or a URL pattern in the filter-mapping element, using
the filter-name value to reference. Filters can access the
initialization parameters declared in the deployment descriptor at
runtime via the FilterConfig interface.
Used in: web-app
-->
<!ELEMENT filter (icon?, filter-name, display-name?, description?,
filter-class, init-param*)>
```

```
<!--
The fully qualified classname of the filter.
Used in: filter
-->
<!ELEMENT filter-class (#PCDATA)>
<!--
Declaration of the filter mappings in this web application. The
container uses the filter-mapping declarations to decide which filters
to apply to a request, and in what order. The container matches the
request URI to a Servlet in the normal way. To determine which filters
to apply it matches filter-mapping declarations either on servlet-name,
or on url-pattern for each filter-mapping element, depending on which
style is used. The order in which filters are invoked is the order in
which filter-mapping declarations that match a request URI for a
servlet appear in the list of filter-mapping elements.The filter-name
value must be the value of the <filter-name> sub-elements of one of the
<filter> declarations in the deployment descriptor.
Used in: web-app
-->
<!ELEMENT filter-mapping (filter-name, (url-pattern | servlet-name))>
<!--
The logical name of the filter. This name is used to map the filter.
Each filter name is unique within the web application.
Used in: filter, filter-mapping
-->
<!ELEMENT filter-name (#PCDATA)>
<!--
The form-error-page element defines the location in the web app
where the error page that is displayed when login is not successful
can be found. The path begins with a leading / and is interpreted
relative to the root of the WAR.
Used in: form-login-config
-->
<!ELEMENT form-error-page (#PCDATA)>
<!--
The form-login-config element specifies the login and error pages
that should be used in form based login. If form based authentication
is not used, these elements are ignored.
Used in: login-config
-->
<!ELEMENT form-login-config (form-login-page, form-error-page)>
<!--
The form-login-page element defines the location in the web app
where the page that can be used for login can be found. The path
begins with a leading / and is interpreted relative to the root of the WAR.
```

```
Used in: form-login-config
-->
<!ELEMENT form-login-page (#PCDATA)>
<!--
The home element contains the fully-qualified name of the enterprise
bean's home interface.
Used in: ejb-ref
Example:
<home>com.aardvark.payroll.PayrollHome</home>
-->
<!ELEMENT home (#PCDATA)>
<!--
The http-method contains an HTTP method (GET | POST |...).
Used in: web-resource-collection
-->
<!ELEMENT http-method (#PCDATA)>
<!--
The icon element contains small-icon and large-icon elements that
specify the file names for small and a large GIF or JPEG icon images
used to represent the parent element in a GUI tool.
Used in: filter, servlet, web-app
-->
<!ELEMENT icon (small-icon?, large-icon?)>
<!--
The init-param element contains a name/value pair as an
initialization param of the servlet
Used in: filter, servlet
-->
<!ELEMENT init-param (param-name, param-value, description?)>
<!--
The jsp-file element contains the full path to a JSP file within
the web application beginning with a `/'.
Used in: servlet
-->
<!ELEMENT jsp-file (#PCDATA)>
<!--
The large-icon element contains the name of a file
containing a large (32 x 32) icon image. The file
name is a relative path within the web application's
war file.
The image may be either in the JPEG or GIF format.
The icon can be used by tools.
Used in: icon
Example:
<large-icon>employee-service-icon32x32.jpg</large-icon>
```

```
-->
<!ELEMENT large-icon (#PCDATA)>
<!--
The listener element indicates the deployment properties for a web
application listener bean.
Used in: web-app
-->
<!ELEMENT listener (listener-class)>
<!--
The listener-class element declares a class in the application must be
registered as a web application listener bean. The value is the fully qualified
classname of the listener class.

Used in: listener
-->
<!ELEMENT listener-class (#PCDATA)>
<!--
The load-on-startup element indicates that this servlet should be
loaded (instantiated and have its init() called) on the startup
of the web application. The optional contents of
these element must be an integer indicating the order in which
the servlet should be loaded. If the value is a negative integer,
or the element is not present, the container is free to load the
servlet whenever it chooses. If the value is a positive integer
or 0, the container must load and initialize the servlet as the
application is deployed. The container must guarantee that
servlets marked with lower integers are loaded before servlets
marked with higher integers. The container may choose the order
of loading of servlets with the same load-on-start-up value.
Used in: servlet
-->
<!ELEMENT load-on-startup (#PCDATA)>
<!--
The local element contains the fully-qualified name of the
enterprise bean's local interface.
Used in: ejb-local-ref
-->
<!ELEMENT local (#PCDATA)>
<!--
The local-home element contains the fully-qualified name of the
enterprise bean's local home interface.
Used in: ejb-local-ref
-->
<!ELEMENT local-home (#PCDATA)>
<!--
```

```
The location element contains the location of the resource in the web
application relative to the root of the web application. The value of
the location must have a leading `/'.
Used in: error-page
-->
<!ELEMENT location (#PCDATA)>
<!--
The login-config element is used to configure the authentication
method that should be used, the realm name that should be used for
this application, and the attributes that are needed by the form login
mechanism.
Used in: web-app
-->
<!ELEMENT login-config (auth-method?, realm-name?, form-login-config?)>
<!--
The mime-mapping element defines a mapping between an extension
and a mime type.
Used in: web-app
-->
<!ELEMENT mime-mapping (extension, mime-type)>
<!--
The mime-type element contains a defined mime type. example:
"text/plain"
Used in: mime-mapping
-->
<!ELEMENT mime-type (#PCDATA)>
<!--
The param-name element contains the name of a parameter. Each parameter
name must be unique in the web application.

Used in: context-param, init-param
-->
<!ELEMENT param-name (#PCDATA)>
<!--
The param-value element contains the value of a parameter.
Used in: context-param, init-param
-->
<!ELEMENT param-value (#PCDATA)>
<!--
The realm name element specifies the realm name to use in HTTP
Basic authorization.
Used in: login-config
-->
<!ELEMENT realm-name (#PCDATA)>
<!--
```

```
The remote element contains the fully-qualified name of the enterprise
bean's remote interface.
Used in: ejb-ref
Example:
<remote>com.wombat.empl.EmployeeService</remote>
-->
<!ELEMENT remote (#PCDATA)>
<!--
The res-auth element specifies whether the web application code signs
on programmatically to the resource manager, or whether the Container
will sign on to the resource manager on behalf of the web application. In the
latter case, the Container uses information that is supplied by the
Deployer.
The value of this element must be one of the two following:
<res-auth>Application</res-auth>
<res-auth>Container</res-auth>
Used in: resource-ref
-->
<!ELEMENT res-auth (#PCDATA)>
<!--
The res-ref-name element specifies the name of a resource manager
connection factory reference. The name is a JNDI name relative to the
java:comp/env context. The name must be unique within a web application.
Used in: resource-ref
-->
<!ELEMENT res-ref-name (#PCDATA)>
<!--
The res-sharing-scope element specifies whether connections obtained
through the given resource manager connection factory reference can be
shared. The value of this element, if specified, must be one of the
two following:
<res-sharing-scope>Shareable</res-sharing-scope>
<res-sharing-scope>Unshareable</res-sharing-scope>
The default value is Shareable.
Used in: resource-ref
-->
<!ELEMENT res-sharing-scope (#PCDATA)>
<!--
The res-type element specifies the type of the data source. The type
is specified by the fully qualified Java language class or interface
expected to be implemented by the data source.
Used in: resource-ref
-->
<!ELEMENT res-type (#PCDATA)>
<!--
```

```
The resource-env-ref element contains a declaration of a web application's
reference to an administered object associated with a resource
in the web application's environment. It consists of an optional
description, the resource environment reference name, and an
indication of the resource environment reference type expected by
the web application code.
Used in: web-app
Example:
<resource-env-ref>
<resource-env-ref-name>jms/StockQueue</resource-env-ref-name>
<resource-env-ref-type>javax.jms.Queue</resource-env-ref-type>
</resource-env-ref>
-->
<!ELEMENT resource-env-ref (description?, resource-env-ref-name,
resource-env-ref-type)>
<!--
The resource-env-ref-name element specifies the name of a resource
environment reference; its value is the environment entry name used in
the web application code. The name is a JNDI name relative to the
java:comp/env context and must be unique within a web application.
Used in: resource-env-ref
-->
<!ELEMENT resource-env-ref-name (#PCDATA)>
<!--
The resource-env-ref-type element specifies the type of a resource
environment reference. It is the fully qualified name of a Java
language class or interface.
Used in: resource-env-ref
-->
<!ELEMENT resource-env-ref-type (#PCDATA)>
<!--
The resource-ref element contains a declaration of a web application's
reference to an external resource. It consists of an optional
description, the resource manager connection factory reference name,
the indication of the resource manager connection factory type
expected by the web application code, the type of authentication
(Application or Container), and an optional specification of the
shareability of connections obtained from the resource (Shareable or
Unshareable).
Used in: web-app
Example:
<resource-ref>
<res-ref-name>jdbc/EmployeeAppDB</res-ref-name>
<res-type>javax.sql.DataSource</res-type>
<res-auth>Container</res-auth>
```

```
<res-sharing-scope>Shareable</res-sharing-scope>
</resource-ref>
-->
<!ELEMENT resource-ref (description?, res-ref-name, res-type, res-auth,
res-sharing-scope?)>
<!--
The role-link element is a reference to a defined security role. The
role-link element must contain the name of one of the security roles
defined in the security-role elements.
Used in: security-role-ref
-->
<!ELEMENT role-link (#PCDATA)>
<!--
The role-name element contains the name of a security role.
The name must conform to the lexical rules for an NMTOKEN.
Used in: auth-constraint, run-as, security-role, security-role-ref
-->
<!ELEMENT role-name (#PCDATA)>
<!--
The run-as element specifies the run-as identity to be used for the
execution of the web application. It contains an optional description, and
the name of a security role.
Used in: servlet
-->
<!ELEMENT run-as (description?, role-name)>
<!--
The security-constraint element is used to associate security
constraints with one or more web resource collections
Used in: web-app
-->
<!ELEMENT security-constraint (display-name?, web-resource-collection+,
auth-constraint?, user-data-constraint?)>
<!--
The security-role element contains the definition of a security
role. The definition consists of an optional description of the
security role, and the security role name.
Used in: web-app
Example:
<security-role>
<description>
This role includes all employees who are authorized
to access the employee service application.
</description>
<role-name>employee</role-name>
</security-role>
```

```
-->
<!ELEMENT security-role (description?, role-name)>
<!--
The security-role-ref element contains the declaration of a security
role reference in the web application's code. The declaration consists
of an optional description, the security role name used in the code,
and an optional link to a security role. If the security role is not
specified, the Deployer must choose an appropriate security role.
The value of the role-name element must be the String used as the
parameter to the EJBContext.isCallerInRole(String roleName) method
or the HttpServletRequest.isUserInRole(String role) method.
Used in: servlet
-->
<!ELEMENT security-role-ref (description?, role-name, role-link?)>
<!--
The servlet element contains the declarative data of a
servlet. If a jsp-file is specified and the load-on-startup element is
present, then the JSP should be precompiled and loaded.
Used in: web-app
-->
<!ELEMENT servlet (icon?, servlet-name, display-name?, description?,
(servlet-class|jsp-file), init-param*, load-on-startup?, run-as?,
security-role-ref*)>
<!--
The servlet-class element contains the fully qualified class name
of the servlet.
Used in: servlet
-->
<!ELEMENT servlet-class (#PCDATA)>
<!--
The servlet-mapping element defines a mapping between a servlet
and a url pattern
Used in: web-app
-->
<!ELEMENT servlet-mapping (servlet-name, url-pattern)>
<!--
The servlet-name element contains the canonical name of the
servlet. Each servlet name is unique within the web application.
Used in: filter-mapping, servlet, servlet-mapping
-->
<!ELEMENT servlet-name (#PCDATA)>
<!--
The session-config element defines the session parameters for
this web application.
Used in: web-app
```

```
-->
<!ELEMENT session-config (session-timeout?)>
<!--
The session-timeout element defines the default session timeout
interval for all sessions created in this web application. The
specified timeout must be expressed in a whole number of minutes.
If the timeout is 0 or less, the container ensures the default
behaviour of sessions is never to time out.
Used in: session-config
-->
<!ELEMENT session-timeout (#PCDATA)>
<!--
The small-icon element contains the name of a file
containing a small (16 x 16) icon image. The file
name is a relative path within the web application's
war file.
The image may be either in the JPEG or GIF format.
The icon can be used by tools.
Used in: icon
Example:
<small-icon>employee-service-icon16x16.jpg</small-icon>
-->
<!ELEMENT small-icon (#PCDATA)>
<!--
The taglib element is used to describe a JSP tag library.
Used in: web-app
-->
<!ELEMENT taglib (taglib-uri, taglib-location)>
<!--
the taglib-location element contains the location (as a resource
relative to the root of the web application) where to find the Tag
Libary Description file for the tag library.
Used in: taglib
-->
<!ELEMENT taglib-location (#PCDATA)>
<!--
The taglib-uri element describes a URI, relative to the location
of the web.xml document, identifying a Tag Library used in the Web
Application.
Used in: taglib
-->
<!ELEMENT taglib-uri (#PCDATA)>
<!--
The transport-guarantee element specifies that the communication
between client and server should be NONE, INTEGRAL, or
```

```
CONFIDENTIAL. NONE means that the application does not require any
transport guarantees. A value of INTEGRAL means that the application
requires that the data sent between the client and server be sent in
such a way that it can't be changed in transit. CONFIDENTIAL means
that the application requires that the data be transmitted in a
fashion that prevents other entities from observing the contents of
the transmission. In most cases, the presence of the INTEGRAL or
CONFIDENTIAL flag will indicate that the use of SSL is required.
Used in: user-data-constraint
-->
<!ELEMENT transport-guarantee (#PCDATA)>
<!--
The url-pattern element contains the url pattern of the mapping. Must
follow the rules specified in Section 11.2 of the Servlet API
Specification.
Used in: filter-mapping, servlet-mapping, web-resource-collection
-->
<!ELEMENT url-pattern (#PCDATA)>
<!--
The user-data-constraint element is used to indicate how data
communicated between the client and container should be protected.
Used in: security-constraint
-->
<!ELEMENT user-data-constraint (description?, transport-guarantee)>
<!--
The web-resource-collection element is used to identify a subset
of the resources and HTTP methods on those resources within a web
application to which a security constraint applies. If no HTTP methods
are specified, then the security constraint applies to all HTTP
methods.
Used in: security-constraint
-->
<!ELEMENT web-resource-collection (web-resource-name, description?,
url-pattern*, http-method*)>
<!--
The web-resource-name contains the name of this web resource
collection.
Used in: web-resource-collection
-->
<!ELEMENT web-resource-name (#PCDATA)>
<!--
The welcome-file element contains file name to use as a default
welcome file, such as index.html
Used in: welcome-file-list
-->
```

```
<!ELEMENT welcome-file (#PCDATA)>
<!--
The welcome-file-list contains an ordered list of welcome files
elements.
Used in: web-app
-->
<!ELEMENT welcome-file-list (welcome-file+)>
<!--
The ID mechanism is to allow tools that produce additional deployment
information (i.e., information beyond the standard deployment
descriptor information) to store the non-standard information in a
separate file, and easily refer from these tool-specific files to the
information in the standard deployment descriptor.
Tools are not allowed to add the non-standard information into the
standard deployment descriptor.
-->
<!ATTLIST auth-constraint id ID #IMPLIED>
<!ATTLIST auth-method id ID #IMPLIED>
<!ATTLIST context-param id ID #IMPLIED>
<!ATTLIST description id ID #IMPLIED>
<!ATTLIST display-name id ID #IMPLIED>
<!ATTLIST distributable id ID #IMPLIED>
<!ATTLIST ejb-link id ID #IMPLIED>
<!ATTLIST ejb-local-ref id ID #IMPLIED>
<!ATTLIST ejb-ref id ID #IMPLIED>
<!ATTLIST ejb-ref-name id ID #IMPLIED>
<!ATTLIST ejb-ref-type id ID #IMPLIED>
<!ATTLIST env-entry id ID #IMPLIED>
<!ATTLIST env-entry-name id ID #IMPLIED>
<!ATTLIST env-entry-type id ID #IMPLIED>
<!ATTLIST env-entry-value id ID #IMPLIED>
<!ATTLIST error-code id ID #IMPLIED>
<!ATTLIST error-page id ID #IMPLIED>
<!ATTLIST exception-type id ID #IMPLIED>
<!ATTLIST extension id ID #IMPLIED>
<!ATTLIST filter id ID #IMPLIED>
<!ATTLIST filter-class id ID #IMPLIED>
<!ATTLIST filter-mapping id ID #IMPLIED>
<!ATTLIST filter-name id ID #IMPLIED>
<!ATTLIST form-error-page id ID #IMPLIED>
<!ATTLIST form-login-config id ID #IMPLIED>
<!ATTLIST form-login-page id ID #IMPLIED>
<!ATTLIST home id ID #IMPLIED>
<!ATTLIST http-method id ID #IMPLIED>
<!ATTLIST icon id ID #IMPLIED>
```

```
<!ATTLIST init-param id ID #IMPLIED>
<!ATTLIST jsp-file id ID #IMPLIED>
<!ATTLIST large-icon id ID #IMPLIED>
<!ATTLIST listener id ID #IMPLIED>
<!ATTLIST listener-class id ID #IMPLIED>
<!ATTLIST load-on-startup id ID #IMPLIED>
<!ATTLIST local id ID #IMPLIED>
<!ATTLIST local-home id ID #IMPLIED>
<!ATTLIST location id ID #IMPLIED>
<!ATTLIST login-config id ID #IMPLIED>
<!ATTLIST mime-mapping id ID #IMPLIED>
<!ATTLIST mime-type id ID #IMPLIED>
<!ATTLIST param-name id ID #IMPLIED>
<!ATTLIST param-value id ID #IMPLIED>
<!ATTLIST realm-name id ID #IMPLIED>
<!ATTLIST remote id ID #IMPLIED>
<!ATTLIST res-auth id ID #IMPLIED>
<!ATTLIST res-ref-name id ID #IMPLIED>
<!ATTLIST res-sharing-scope id ID #IMPLIED>
<!ATTLIST res-type id ID #IMPLIED>
<!ATTLIST resource-env-ref id ID #IMPLIED>
<!ATTLIST resource-env-ref-name id ID #IMPLIED>
<!ATTLIST resource-env-ref-type id ID #IMPLIED>
<!ATTLIST resource-ref id ID #IMPLIED>
<!ATTLIST role-link id ID #IMPLIED>
<!ATTLIST role-name id ID #IMPLIED>
<!ATTLIST run-as id ID #IMPLIED>
<!ATTLIST security-constraint id ID #IMPLIED>
<!ATTLIST security-role id ID #IMPLIED>
<!ATTLIST security-role-ref id ID #IMPLIED>
<!ATTLIST servlet id ID #IMPLIED>
<!ATTLIST servlet-class id ID #IMPLIED>
<!ATTLIST servlet-mapping id ID #IMPLIED>
<!ATTLIST servlet-name id ID #IMPLIED>
<!ATTLIST session-config id ID #IMPLIED>
<!ATTLIST session-timeout id ID #IMPLIED>
<!ATTLIST small-icon id ID #IMPLIED>
<!ATTLIST taglib id ID #IMPLIED>
<!ATTLIST taglib-location id ID #IMPLIED>
<!ATTLIST taglib-uri id ID #IMPLIED>
<!ATTLIST transport-guarantee id ID #IMPLIED>
<!ATTLIST url-pattern id ID #IMPLIED>
<!ATTLIST user-data-constraint id ID #IMPLIED>
<!ATTLIST web-app id ID #IMPLIED>
<!ATTLIST web-resource-collection id ID #IMPLIED>
```

```
<!ATTLIST web-resource-name id ID #IMPLIED>
<!ATTLIST welcome-file id ID #IMPLIED>
<!ATTLIST welcome-file-list id ID #IMPLIED>
<!-- This file contains the orion-specific configuration for a web-application.
The path to the file is located at
ORION_HOME/application-deployments/deploymentName/warname(.war)/orion-web.xml or
(web-app-root/)WEB-INF/orion-web.xml if no deployment-directory is specified in
server.xml. -->
<!ELEMENT orion-web-app ( classpath*, context-param-mapping*, mime-mappings*,
virtual-directory*, access-mask?, cluster-config?, servlet-chaining*,
request-tracker*, session-tracking?, resource-ref-mapping*,
security-role-mapping*, env-entry-mapping*, ejb-ref-mapping*,
expiration-setting*, web-app?, jazn-web-app?, web-app-class-loader?)>
<!ATTLIST orion-web-app autoreload-jsp-beans (true|false) "true"
autoreload-jsp-pages (true|false) "true"
default-buffer-size CDATA "2048"
default-charset %CHARSET; "iso-8859-1"
default-mime-type CDATA #IMPLIED
deployment-version CDATA #IMPLIED
development (true|false) "false"
directory-browsing (allow|deny) "deny"
file-modification-check-interval %NUMBER; "1000"
internationalize-resources (true|false) "false"
jsp-cache-directory CDATA #IMPLIED
jsp-cache-tlds (true|fase) "true"
jsp-taglib-locations CDATA #IMPLIED
jsp-print-null (true|false) "true"
jsp-timeout %NUMBER; "0 (never)"
simple-jsp-mapping (true|false) "false"
enable-jsp-dispatcher-shortcut (true|false) "true"
persistence-path CDATA #IMPLIED
servlet-webdir %PATH; "/servlet/"
source-directory CDATA #IMPLIED
temporary-directory CDATA #IMPLIED
>




<!ENTITY % CLASSNAME "CDATA">
<!-- A group that this security-role-mapping implies. Ie all the members of the
specified group are included in this role. -->
<!ELEMENT group (#PCDATA)>
<!ATTLIST group name CDATA #IMPLIED
>
```

```
<!-- A relative (to the application root) or absolute path to a directory where
application state should be stored across restarts. -->
<!ELEMENT persistence (#PCDATA)>
<!ATTLIST persistence path CDATA #IMPLIED
>
<!-- An argument used when invoking the client. -->
<!ELEMENT argument (#PCDATA)>
<!ATTLIST argument value CDATA #IMPLIED
>
<!-- A short description of this component. -->
<!ELEMENT description (#PCDATA)>
<!-- A relative/absolute path to log events to. -->
<!ELEMENT file (#PCDATA)>
<!ATTLIST file path CDATA #IMPLIED
>
<!-- An ODL formated log file. The max-file-size is the maximum number of
kilobytes a single log file is allowed to grow to. The max-directory-size is the
maximum number of kilobytes that the directory is allowed to contain. -->
<!ELEMENT odl (#PCDATA)>
<!ATTLIST odl path CDATA #REQUIRED max-file-size CDATA #IMPLIED
max-directory-size CDATA #IMPLIED>
<!-- A ejb-jar module of the application. -->
<!ELEMENT ejb-module (#PCDATA)>
<!ATTLIST ejb-module path CDATA #IMPLIED
remote (true|false) "false"
>
<!-- A relative/absolute path/URL to a directory or a .jar/.zip to add as a
library-path for this server. Directories are scanned for jars/zips to include
at startup. -->
<!ELEMENT library (#PCDATA)>
<!ATTLIST library path CDATA #IMPLIED
>
<!-- The read-access policy. -->
<!ELEMENT read-access (namespace-resource)>
<!-- A e-mail address to log events to. A valid mail-session also needs to be
specified if this option is used. -->
<!ELEMENT mail (#PCDATA)>
<!ATTLIST mail address CDATA #IMPLIED
>
<!-- A list of arguments to used when invoking the app-client if starting it
in-process (auto-start="true"). -->
<!ELEMENT arguments (argument*)>
<!-- Namespace (naming context) security policy for RMI clients. -->
<!ELEMENT namespace-access (read-access, write-access)>
<!-- Contains a name/value pair initialization param. -->
```

```
<!ELEMENT property (#PCDATA)>
<!ATTLIST property name CDATA #IMPLIED
value CDATA #IMPLIED
>
<!-- -->
<!ELEMENT data-sources (#PCDATA)>
<!ATTLIST data-sources path CDATA #IMPLIED
>
<!-- The write-access policy. -->
<!ELEMENT write-access (namespace-resource)>
<!-- An application-client module of the application. An app-client is a GUI or
console-based standalone client that interracts with the server. -->
<!ELEMENT client-module (arguments?)>
<!ATTLIST client-module auto-start (true|false) "false"
deployment-time CDATA #IMPLIED
path CDATA #IMPLIED
user CDATA #IMPLIED
>
<!-- A user that this security-role-mapping implies. -->
<!ELEMENT user (#PCDATA)>
<!ATTLIST user name CDATA #IMPLIED
>
<!-- Specifies an optional user-manager to use, example user-managers are
com.evermind.sql.DataSourceUserManager,
com.evermind.ejb.EJBUserManager, etc... Used to integrate existing systems and
provide custom user-managers for
web-applications. -->
<!ELEMENT user-manager (description?, property*)>
<!ATTLIST user-manager class %CLASSNAME; #IMPLIED
display-name CDATA #IMPLIED
>
<!-- An orion-ejb-jar.xml file contains the deploy-time info for an application.
It is located in
ORION_HOME/application-deployments/deploymentName/orion-application.xml after
deployment and META-INF/orion-application.xml below the application root if
bundled with the application or if no deployment-directory is specified in
server.xml. If using deployment-directory (which is the default) the bundled
version will be copied to the deployment location if and only if no file exists
at that location. It is used to specify initial (first time) deployment
properties.
After each deployment the deployment file is reformatted/augmented/altered by
the server to add any new/missing info to it. -->
<!ELEMENT orion-application
(ejb-module*,web-module*,client-module*,commit-coordinator?,security-role-mappin
g*, persistence?, library*, principals?, mail-session*, user-manager?, log?,
jazn?, data-sources?, connectors?, resource-provider*, namespace-access?,
jndi-clustering?)>
<!ATTLIST orion-application autocreate-tables (true|false) "true"
autodelete-tables (true|false) "false"
default-data-source CDATA #IMPLIED
deployment-version CDATA #IMPLIED
treat-zero-as-null (true|false) "false"
>
<!-- A web-application module of the application. Each web-application can be
installed on any site and in any context on those sites (for instance
http://www.myserver.com/myapp/). -->
<!ELEMENT web-module (#PCDATA)>
<!ATTLIST web-module id CDATA #IMPLIED
path CDATA #IMPLIED
>
<!-- -->
<!ELEMENT principals (#PCDATA)>
<!ATTLIST principals path CDATA #IMPLIED
>
<!-- Logging settings. -->
<!ELEMENT log (file*, mail*, odl*)>
<!-- The runtime mapping (to groups and users) of a role. Maps to a
security-role of the same name in the assembly descriptor. -->
<!ELEMENT security-role-mapping (group*, user*)>
<!ATTLIST security-role-mapping impliesAll CDATA #IMPLIED
name CDATA #IMPLIED
>
<!-- The session SMTP-server host (if using SMTP). -->
<!ELEMENT mail-session (description?, property*)>
<!ATTLIST mail-session location CDATA #IMPLIED
smtp-host CDATA #IMPLIED
username CDATA #IMPLIED
password CDATA #IMPLIED
>
<!-- A resource with a specific security setting. -->
<!ELEMENT namespace-resource (security-role-mapping)>
<!ATTLIST namespace-resource root CDATA #REQUIRED
>
<!-- -->
<!ELEMENT connectors (#PCDATA)>
<!ATTLIST connectors path CDATA #IMPLIED
>
<!-- JAZN configuration -->
<!ELEMENT jazn-web-app (#PCDATA)>
<!ATTLIST jazn-web-app auth-method CDATA ""
```

```
runas-mode (true | false) "false"
doasprivileged-mode (true | false) "true"
>
<!-- -->
<!ELEMENT jazn (property*, jazn-web-app?)>
<!ATTLIST jazn provider (XML | LDAP) "XML"
location CDATA #IMPLIED
default-realm CDATA #IMPLIED
persistence (NONE | ALL | VM_EXIT) "VM_EXIT"
config CDATA ""
>
<!ELEMENT jndi-clustering (#PCDATA)>
<!ATTLIST jndi-clustering enabled (true | false) "true" >
<!-- -->
<!ELEMENT commit-coordinator (commit-class, property*)>
<!-- -->
<!ELEMENT commit-class (#PCDATA)>
<!ATTLIST commit-class class %CLASSNAME; #IMPLIED
>
<!-- Specifies a resource-provider to plug-in, ie.
com.evermind.server.deployment.ContextScanningResourceProvider -->
<!ELEMENT resource-provider (description?, property+)>
<!ATTLIST resource-provider class %CLASSNAME; #IMPLIED
name CDATA #IMPLIED
>
<!-- Specifies a user manager for use access to security sensitive strings.
If no element is present, then the UserManager for the application
itself will become used.
-->
<!ELEMENT password-manager (principals?, jazn?, user-manager?)>
```

Index