Oracle® Containers for J2EE

Security Guide

10*g* Release 3 (10.1.3)

B14429-01

January 2006

ORACLE®

Oracle Containers for J2EE Security Guide, 10*g* Release 3 (10.1.3)

B14429-01

Primary Author: Brian Wright

Contributing Author: Elizabeth Hanes Perry

Contributor: Ganesh Kirti, Raymond Ng, Rachel Chan, Nithya Muralidharan, Kumar Valendhar, Moushmi Banerjee, Dheeraj Goswami, Sam Zhou, Srikant Tirumalai, Bill Bathurst, Debu Panda, Tom Snyder, Jeff Trent, Bob Nettleton, Vinay Shukla, Michael Hwa, Jayanthi Kulkarni, Kavita Tippani, Helen Zhao, Sandeep Bangera, Cania Lee Chung, Deepika Damojipurapu, Lakshmi Thiyagarajan, Serouj Ourishian, Phil Varner, Chaya Ramanujam, Jyotsna Laxminarayanan, Lelia Yin, Raghav Srinivisan, Dan Hynes, Alfred Franci

Contents

3 Overview of Security Administration and Configuration

4 Java VM Security Settings for OC4J

5 Tasks and Guidelines in Setting Security

6 Oracle Identity Management Security Provider

7 File-Based Security Provider

8 Login Modules

9 External LDAP Security Providers

10 COREid Access Security Provider

11 Integration with SSL and ORMIS

12 Oracle HTTPS for Client Connections

13 Web Application Security Configuration

14 EJB Security Configuration

15 Common Secure Interoperability Protocol

16 Security Support for Resource Adapters

A Tips and Troubleshooting for OC4J Security

B OracleAS JAAS Provider Samples

C OracleAS JAAS Provider Admintool Reference

List of Examples

List of Figures

List of Tables

Preface

This manual discusses Oracle 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. It discusses the Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider in detail, as well as discussing security implications of individual J2EE features, including Web applications, Enterprise JavaBeans (EJBs), the J2EE Connector Architecture, Secure Sockets Layer, and the Common Secure Interoperability Version 2 protocol (CSIv2).

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. Accessibility standards will continue to evolve over time, and Oracle 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 more information, visit the Oracle Accessibility Program Web site at

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

Accessibility of Code Examples in Documentation

Screen readers 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, some screen readers 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 does not own or control. Oracle neither evaluates nor makes any representations regarding the accessibility of these Web sites.

TTY Access to Oracle Support Services

Oracle provides dedicated Text Telephone (TTY) access to Oracle Support Services within the United States of America 24 hours a day, seven days a week. For TTY support, call 800.446.2398.

Related Documentation

For more information, see the following Oracle resources.

Additional OC4J documents:

- *Oracle Containers for J2EE Developer's Guide*

  This discusses items of general interest to developers writing an application to run on OC4J—issues that are not specific to a particular container such as the servlet, EJB, or JSP container. (An example is class loading.)

- *Oracle Containers for J2EE Deployment Guide*

  This covers information and procedures for deploying an application to an OC4J environment. This includes discussion of the deployment plan editor that comes with Oracle Enterprise Manager 10*g*.

- *Oracle Containers for J2EE Configuration and Administration Guide*

  This discusses how to configure and administer applications for OC4J, including use of the Oracle Enterprise Manager 10*g* Application Server Control Console, use of standards-compliant MBeans provided with OC4J, and, where appropriate, direct use of OC4J-specific XML configuration files.

- *Oracle Containers for J2EE Servlet Developer's Guide*

  This provides information for servlet developers regarding use of servlets and the servlet container in OC4J, including basic servlet development and use of JDBC and EJBs.

- *Oracle Containers for J2EE Support for JavaServer Pages Developer's Guide*

  This provides information about JavaServer Pages development and the JSP implementation and container in OC4J. This includes discussion of Oracle features such as the command-line translator and OC4J-specific configuration parameters.

- *Oracle Containers for J2EE JSP Tag Libraries and Utilities Reference*

  This provides conceptual information as well as detailed syntax and usage information for tag libraries and JavaBeans provided with OC4J.

- *Oracle Containers for J2EE Services Guide*

  This provides information about standards-based Java services supplied with OC4J, such as JTA, JNDI, JMS, JAAS, the Oracle Application Server Java Object Cache, and the XML Query Service.

- *JAAS Provider API Reference*

  This is a Javadoc set for the OracleAS JAAS Provider.

- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide*

This provides information about Enterprise JavaBeans development and the EJB implementation and container in OC4J.

- *Oracle Application Server Web Services Developer's Guide*

  This describes Web services development and configuration in OC4J and Oracle Application Server.

- *Oracle Application Server Advanced Web Services Developer's Guide*

  This book describes topics beyond basic Web service assembly. For example, it describes how to diagnose common interoperability problems, how to enable Web service management features (such as reliability, auditing, and logging), and how to use custom serialization of Java value types.

- *Oracle Application Server Web Services Security Guide*

  This describes Web services security and configuration in OC4J and Oracle Application Server.

From the Oracle Application Server core documentation group:

- *Oracle Application Server Administrator's Guide*
- *Oracle Application Server Enterprise Deployment Guide*
- *Oracle HTTP Server Administrator's Guide*
- *Oracle Process Manager and Notification Server Administrator's Guide*
- *Oracle Application Server Certificate Authority Administrator's Guide*

For Oracle Identity Management, Oracle Internet Directory, and OracleAS Single Sign-On:

- *Oracle Identity Management Administrator's Guide*
- *Oracle Identity Management Integration Guide*
- *Oracle Identity Management Guide to Delegated Administration*
- *Oracle Identity Management Application Developer's Guide*
- *Oracle Internet Directory Administrator's Guide*
- *Oracle Internet Directory API Reference*
- *Oracle Application Server Single Sign-On Administrator's Guide*

For Oracle COREid Access and Identity:

- *Oracle COREid Access and Identity Introduction*
- *Oracle COREid Access and Identity Installation Guide*
- *Oracle COREid Access and Identity Administration Guide*
- *Oracle COREid Access and Identity Developer Guide*
- *Oracle COREid Access and Identity Deployment Guide*

For additional information, see:

- The Sun Java and J2EE Web pages, especially the Java Authentication and Authorization Service (JAAS) Web site at :

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

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type within a paragraph indicates commands, URLs, Java class names and method names, file and directory names, text that appears on the screen, or text that you enter.

What's New

This section describes new features in this release:

Changes Since Release 10.1.2

The following security features and enhancements are added for the OC4J 10.1.3 implementation:

- Support for the COREid Access security provider
- Support for the LDAP-based provider in standalone OC4J
- Digest authentication support, and client certification authentication and authorization support
- Implementation of the Java Authorization Contract for Containers (JSR-115).
- JAAS integration with EJBs
- ORMI enhancements for SSL (ORMIS)
- Support for subject propagation (with ORMI or ORMIS)
- JMX and MBeans support (JSR-77) for security configuration
- New OC4J user and role accounts (see below)
- Enhanced Java 2 security support
- Web services security (described in another document)

In addition, note the following changes since the OC4J 10.1.2 implementation:

- There is a new consolidated "JAAS mode" for authorization, for both servlets and EJBs. This replaces previous `runas-mode` and `dosasprivileged-mode` functionality for servlets, and `USE_JAAS` functionality (introduced in preliminary 10.1.3 releases) for EJBs. The previous functionality is supported but deprecated in the OC4J 10.1.3 implementation.
- The instance-level `jazn-data.xml` configuration file used in previous releases to store user and role configuration (for the file-based provider), policy configuration (for the file-based, external LDAP, or custom security provider), and login module configuration (for all security providers) has been renamed `system-jazn-data.xml`. However, an application can optionally use an application-specific `jazn-data.xml` repository file to store user and role configuration for the file-based provider.

- The `XMLUserManager` class and its data store, `principals.xml`, are deprecated and will no longer be supported at a future release. We strongly encourage you to migrate your existing applications. For instructions, see "Migrating Principals from the principals.xml File" on page 7-15.
- The `com.evermind` package has been largely replaced by `oracle.j2ee`. Although the `com.evermind.*` classes continue to exist, they are deprecated; we encourage you to move your applications to `oracle.j2ee.*`.
- Custom `UserManager` classes are still supported at this release, but will be deprecated at a future release. We recommend that you use JAAS custom login modules instead of custom `UserManager` implementations.
- For the Oracle Identity Management security provider, the application realm and external realm are deprecated.
- The `external.synchronization` property is no longer supported.
- The default setting of the `jaas.username.simple` property is now "`true`"; in the 10.1.2 implementation the default setting was "`false`". This now means that by default, realm names are omitted from the names of authenticated principals returned by such methods as `getUserPrincipal()` and `getRemoteUser()` for servlets, and `getCallerPrincipal()` for EJBs.
- There have been some OC4J account name changes: the `admin` account is now `oc4jadmin`; the `administrators` role is now `oc4j-administrators`; the `jmx-users` role is now `oc4j-app-administrators`. For the file-based provider in standalone OC4J, `oc4jadmin` is initially deactivated. See "Predefined OC4J Accounts" on page 3-11.
- Required OC4J accounts are created automatically in Oracle Internet Directory when you associate an OC4J instance with an OID instance. See "Required OC4J Accounts Created in Oracle Internet Directory" on page 6-11.
- Setting `LD_LIBRARY_PATH` is no longer necessary in the 10.1.3 implementation.
- The `jazn.debug.log.enable` flag is no longer supported for logging. Use regular OC4J logging features. See "Logging" on page A-6.

1

Standard Security Concepts

This chapter provides an overview of the Java 2 security model, Java Authentication and Authorization Service (JAAS), and related security concepts. The following topics are covered:

Introducing the Java 2 Security Model and JAAS

The Java 2 Security Model is fundamental to the Oracle Application Server security implementation. 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 to set security policies for these domains.

The Java 2 Security Model by itself, however, has certain limitations. It is code-based only, as opposed to being declarative in deployment descriptors. It also has no policy management API, and uses a file-based implementation that does not scale well.

The Java Authentication and Authorization Service (JAAS) is a Java package that enables applications to authenticate and enforce access controls upon users. It 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. It also supports declarative security settings, in deployment descriptors, instead of being limited to code-based security settings.

See Also:

- "About Principals and Subjects" on page 1-4
- For a tutorial on Java 2 Security:

  `http://java.sun.com/docs/books/tutorial/security1.2/index.html`

- For full information on Java 2 Security:

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

- JAAS documentation at the following Web site for more specific discussions of key JAAS features:

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

Authentication and Authorization

This section covers the following topics regarding authentication and authorization.

Authentication and Authorization Concepts

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

- *Authentication* deals with the question "Who is trying to access my services?" In any system and application it is paramount to ensure that the identity of the entity or caller trying to access your application is identified in a secure manner. In a multitier application, the entity or caller can be a human user, a business application, a host, or one entity acting on behalf of (or impersonating) another entity.

  Authentication information, such as user names and passwords, is stored in a *user repository*, such as an XML file, database, or directory service. When a subject attempts to access a J2EE application, such as by logging in, it is the role of a *security provider* to look up the subject in the user repository and verify the subject's identity. A security provider is a module that provides an implementation of a specific security service such as authentication or authorization. The Oracle Internet Directory (OID) is an example of a user repository.

  See Also:

  - *Oracle Internet Directory Administrator's Guide* for information about Oracle Internet Directory.

  Although each J2EE application determines which user can use the application, it is the security provider that authenticates the user's identity through the user repository.

  OC4J supports several different authentication methods, both standard and Oracle-specific. For details, see "Authentication in the OC4J Environment" on page 2-5.

- *Authorization* regards the question "Who can access what resources offered by which components?" In a J2EE application, resources are typically expressed in terms of URL patterns for Web applications, and method permissions for EJBs. Authorization is on a per-role basis, with appropriate permissions being assigned to each defined role in an application. This is further discussed in "Role-Based Access Control: Roles and Role Hierarchy" on page 1-9.

  Developers specify authorization for subjects in the application deployment descriptors.

Capability Model of Access Control

The *capability model* is a method for organizing authorization information. The Java 2 Security Model uses the capability model to control access to permissions. With this model, authorization is associated with an entity (referred to as a principal, defined shortly), such as a user named `frank` in the following example. Table 1–1 shows the permissions that user `frank` is authorized to use:

Table 1–1 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 1–1 are retrieved and granted to user `frank`. User `frank` is then free to execute the actions permitted by these permissions.

An *access control list* (ACL) is a table with information about which access rights each user has for a particular protected resource, such as a directory or individual file. Each resource has a security attribute that identifies its access control list. The list has an entry for each system user with access privileges.

JAAS Security Model Versus J2EE Security Model

J2EE defines a declarative authorization model for container-managed security that decouples applications from the underlying security infrastructure. Authorization policy is expressed statically in the application deployment descriptors, rather than in application code. Authorization is role-based and is granted at access-level, typically protecting resources such as a Web URL or an EJB method. Once access is granted, any functionality of the resource is available. This model is relatively *coarse-grained*, but suffices for many purposes.

By contrast, JAAS supports customized authentication and has an authorization model that is more dynamic and relatively *fine-grained*, where authorization is according to an adaptable security policy. The JAAS model is more customizable and extensible than the J2EE model, with features such as custom permission types.

For example, while J2EE security is sufficient for general protection of a Web URL or EJB method, JAAS security would be required to control who may access a file in the file system, or who may access security policy, create a user, or change a password.

As appropriate and necessary, you can use either model or both with an application. Both models are fully supported in OC4J. It is advisable to limit yourself to the J2EE authorization model whenever it meets your needs, given that the JAAS authorization model is more complicated to deploy and administer. Table 1–2 and Table 1–3 summarize the pros and cons.

Table 1–2 J2EE Security Pros and Cons

J2EE Security: Pros	J2EE Security: Cons
■ Easier to use. ■ Deploys within standard deployment descriptors. ■ Platform-independent. ■ Authentication managed by container.	■ Role-based with no permissions for defined roles. ■ Static: cannot be changed at runtime.

Table 1–3 JAAS Security Pros and Cons

JAAS Security: Pros	JAAS Security: Cons
■ Allows custom authentication modules. ■ You can authenticate to multiple user repositories. ■ Allows finer-grained access and authorization control. ■ You can define your own policy store.	■ More difficult to implement; more code-centric. ■ More difficult to administer and deploy.

About Principals and Subjects

A *principal* is a specific identity, such as a user named `frank` or a role named `hr`. A principal is represented by an instance of a class that implements the `java.security.Principal` interface. A principal class must define a namespace that contains a unique name for each instance of the class.

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 or other credentials. A subject is represented by an instance of the `javax.security.auth.Subject` class.

A subject can contain multiple identities, each represented by a principal. For example, a subject that represents a person, user `frank`, may have two principals:

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

After authentication of a user, a `Subject` instance represents the authenticated user, and then appropriate `Principal` instances are added to the `Subject` instance. The `Principal` instances are used in authorizing the authenticated user to perform specific privileged actions.

About Permissions, Policies, and Realms

This section provides an overview of permissions, policies, and related topics, covering the following:

- Security Permissions
- Security Policies
- Protection Domains
- Security Managers and Access Control

- Security Realms

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.

The `java.security.Permission` class is an abstract class that represents access to a given resource, and optionally a specified action on that resource. A key method of this class is `implies(Permission permission)`, which checks if the actions of the specified permission are implied by the actions of the permission instance upon which the method is called.

Here are common types of permissions and the classes that represent them (all extending `Permission`, either directly or indirectly):

- `java.security.AllPermission`
- `java.lang.RuntimePermission` (includes only a resource target)
- `java.io.FilePermission` (includes a resource and actions)

Table 1–4 identifies the elements that comprise a Java permission instance.

Table 1–4 Java Permission Instance Elements

Element	Description	Example
Class name	Permission class	`java.io.FilePermission`
Target	Target name (resource) to which this permission applies	Directory `/home/*`
Actions	Actions associated with this target	Read, write, and execute permissions on directory `/home/*`

Security Policies

A *policy* is an association between resources and users or roles. More specifically, a policy is a repository of JAAS authorization rules, containing information that answers the question: Given a grantee, what are the granted permissions of the grantee?

A policy is represented in Java by a `Policy` object (`java.security.Policy` or `javax.security.auth.Policy`); a policy object stores a set of permissions.

Policies are declared in `.policy` files, such as `java.policy` or `java2.policy`. A policy contains a collection of permission grants to principals, and may contain a reference to a keystore (described in "Key Encryption and Exchange" on page 1-10). The following are typical locations for policy files:

- *JAVA_HOME*`/lib/security/java.policy`
- *USER_HOME*`/java.policy`
- *ORACLE_HOME*`/j2ee/home/config/java2.policy`

Table 1–5 describes the Sun Microsystems implementation of policy file parameters. A codesource is represented by a `java.security.CodeSource` instance.

Table 1–5 Policy File Parameters

Parameter	Definition	Examples
Subject	One or more principal(s)	`duke`
Codesource	A URL location (codebase) and optionally an array of certificates (stored in a Java keystore `.jks` file)	`file:` (any file on the local file system) `http://*.oracle.com` (any file on any host at `oracle.com`) `file:${j2ee.home}/lib/oc4j-internal.jar`

Protection Domains

A protection domain groups permissions with a codesource, essentially representing the permissions granted to the codesource. (The policy currently in effect is what determines protection domains. In the default implementation of the `Policy` class, a protection domain is one grant entry in the file.)

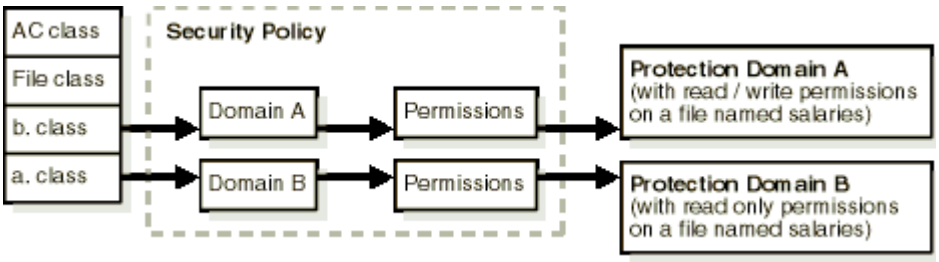
Each Java class is associated with a protection domain when it is loaded. Specifically, each class being loaded is associated with a `java.security.ProtectionDomain` instance. The permissions granted to this protection domain may be statically bound or dynamically determined when an access control check is performed. Each protection domain is assigned a set of permissions based on a configured security policy when the JVM is started.

A `ProtectionDomain` instance contains a codesource (described in the preceding section). It may also contain a `Principal` array describing who is executing the code, a classloader reference, and a permission collection (`java.security.PermissionCollection` instance) representing a collection of `Permission` objects.

The permission collection is effectively defined as the intersection of all permission sets assigned to protection domains at the moment of the security check.

Figure 1–1 shows how protection domains fit into the basic model for authorization checking at runtime.

Figure 1–1 Java 2 Security Model



Security Managers and Access Control

A *security manager* (`java.lang.SecurityManager` instance) allows an application to implement security policies. For any given operation that is attempted, the security manager allows the application to determine what the operation is and whether it should be allowed in the current security context. The `SecurityManager` class has a number of `checkXxx()` methods, each of which checks whether the operation *Xxx* is allowed, and throws an exception if it is not. This includes the instance method `checkPermission(Permission)`, which throws an exception if a requested access, specified by the given permission, is not permitted by the security policy currently in effect.

An *access controller* (`java.security.AccessController` instance) is also involved in access-control operations and decisions. The default implementation of the `SecurityManager` method `checkPermission(Permission)` actually calls the `AccessController` static method `checkPermission(Permission)`.

Basically, an access controller is used to do the following:

- Decide whether access to a system resource should be allowed or denied, based on the security policy currently in effect.
- Mark code as being privileged, thus affecting subsequent access determinations.
- Obtain a snapshot of the current calling context so access-control decisions from a different context can be made with respect to the saved context.

Any application that controls access to system resources should invoke `AccessController` methods if it is to use the specific security model and access control algorithm utilized by these methods. If, on the other hand, the application wishes to defer the security model to that of the `SecurityManager` installed at runtime, then it should instead invoke corresponding methods in the `SecurityManager` class.

In comparison, `SecurityManager` represents the concept of a central point of access control, while `AccessController` implements a particular access control algorithm, with special features such as the `doPrivileged()` method, which performs a specified privileged action with privileges enabled.

See Also:

- For more information about security management and comparison between security managers and access controllers:

  `http://java.sun.com/j2se/1.5.0/docs/guide/security/spec/security-spec.doc6.html`

Security Realms

The JAAS framework does not explicitly define user communities. However, J2EE has the concept of user communities called *realms*.

A realm is a collection of users and roles that are controlled by the same authentication policy. In other words, a realm is a protection domain, or security domain, that defines a set of permissions for authenticated users.

Each realm includes a set of configured users, roles, and policies. (In OC4J configuration, users, roles, and policies can all be configured within a realm definition.) At runtime, a realm defines an enterprise scope over which certain identity management policies (such as those corresponding to users and roles) are enforced.

See Also:

- "Tasks and Guidelines for Using Security Realms in OC4J" on page 5-3
- "Realm Management in LDAP-Based Environments" on page 6-2

Login Module Authentication

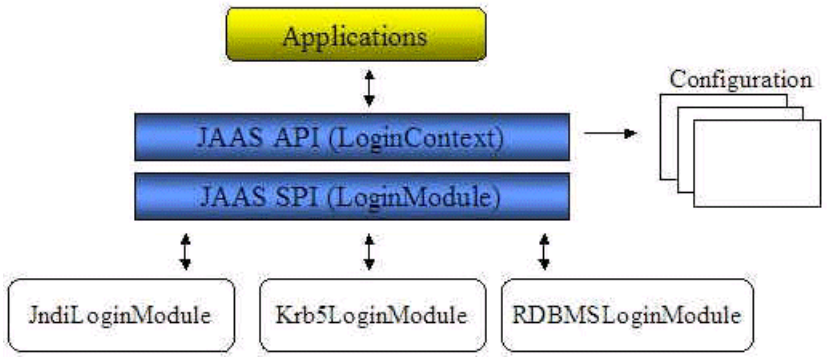
Within the JAAS pluggable authentication framework, an application server and any underlying authentication services remain independent from each other. Authentication services can be plugged in through JAAS *login modules* without requiring modifications to the application server or application code. A login module

is primarily responsible for authenticating a user based on supplied credentials (such as a password), and adding the proper principals (such as roles) to the subject. Possible types of JAAS login modules include a principal-mapping JAAS module, a credential-mapping JAAS module, or a Kerberos JAAS module.

A login module is an instance of a class that implements the `javax.security.auth.spi.LoginModule` interface, and is plugged in under an application to provide a particular type of authentication.

Within this framework, the `javax.security.auth.login.LoginContext` class provides the basic methods used to authenticate subjects such as users, roles, or computing services (when a user tries to log in to the application, for example). An application instantiates this class with a name and a callback handler (described shortly). When the `login()` method of a `LoginContext` instance is invoked by an application that a subject is trying to access, the `LoginContext` instance consults configuration settings, using a mechanism that employs the name that was passed in, to determine the appropriate login module to invoke for the application. Figure 1–2 summarizes this.

Figure 1–2 Login Modules



A *callback handler* is a `javax.security.auth.callback.CallbackHandler` instance that allows a login module to interact with a user to obtain login information. The only method specified by `CallbackHandler` is the `handle(Callback[])` method, which takes an array of callbacks, which are instances of a class that implements the `java.security.auth.callback.Callback` interface. Callbacks do not retrieve or display requested information from the underlying security service, but simply provide the functionality to pass the requests to an application and, as applicable, to return the requested information back to the security service. Callback implementations in the `javax.security.auth.callback` package include: a name callback handler (`NameCallback`) to handle a user name, a password callback handler (`PasswordCallback`) to handle a password, and a text input callback handler (`TextInputCallback`) to handle any field in a login form other than a user name or password field.

If authentication succeeds, then the authenticated subject can be retrieved by invoking the `getSubject()` method of the `LoginContext` instance.

Different login modules can be configured with different applications, and a single application can use multiple login modules. The JAAS framework defines a two-phase authentication process to coordinate the login modules configured for an application.

Custom or external (third-party) login modules may be used with any given application. Oracle provides the login modules `RealmLoginModule` (for the

file-based and LDAP-based providers), `LDAPLoginModule` (for external LDAP providers), and `CoreIDLoginModule` (for the COREid Access provider).

Note: An application that uses declarative J2EE authentication with OC4J and the OracleAS JAAS Provider does not have to create a `LoginContext` instance; it is created by OC4J implicitly.

See Also:

- Chapter 8, "Login Modules"

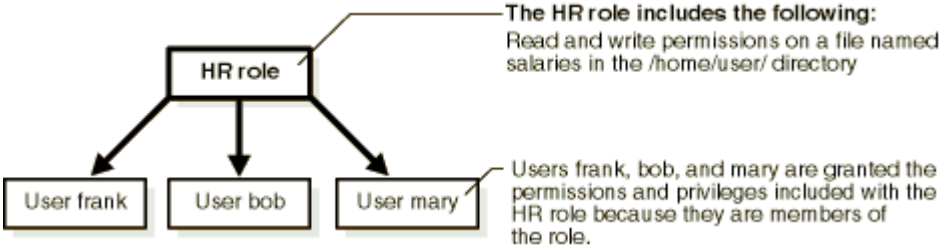
Role-Based Access Control: Roles and Role Hierarchy

A *role* is equivalent to a logical group of users. Wherever a role is discussed in this document, you can think of it as a group of users who will be performing the same tasks and therefore be given the same access capabilities. Roles are the identities that each application uses to indicate access rights to its different objects and functions. A user assumes a role to gain access to an appropriate set of these resources.

Role-based access control is a JAAS feature that 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. Figure 1–3 provides an example of role-based access control.

Figure 1–3 Role-Based Access Control



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 by updating all 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.

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.

See Also:

- "Overview of Security Role Mapping" on page 2-13

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 user name / password combinations.
- Identity propagation: This allows one client to act as the agent of another client, using the identity of the original client.

Secure Sockets Layer and HTTPS

The Secure Sockets Layer (SSL) is the industry-standard point-to-point protocol which provides confidentiality through 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 Apache JServ Protocol link between the Oracle HTTP Server and OC4J processes.

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.

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, and country)
- 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.

Key Encryption and Exchange

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 public key of the server, and sends the encrypted bulk key to the server.
4. The server decrypts the bulk encryption key using the private key of the server.

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.

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 if the signer of the signer is on the approved list, 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 approved list are considered to be 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. (Note that certificates of the CAs themselves are always self-signed.)

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 can authenticate other parties as well as authenticate itself to other parties. The keystore password is obfuscated. 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.

In Oracle Application Server, an Oracle *wallet* is equivalent to a keystore.

Identity Propagation

Identify propagation, or *subject propagation*, refers to 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.

In OC4J, subject propagation is used with IIOP, in accordance with the CSIv2 specification. It is also used with ORMI if it is enabled on the client and server.

See Also:

- "Enabling and Configuring Subject Propagation for ORMI" on page 14-11

Developing Secure J2EE Applications

J2EE software development is based on a develop-deploy-manage cycle. The Oracle Application Server security implementation plays an important role in the deploy-manage part of the cycle. 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 developer creates Web components, enterprise beans, applets, servlets, and application clients.

   The Oracle Application Server security implementation offers programmatic interfaces, but the developer can create components without making use of those interfaces.

2. The developer defines J2EE logical roles and assigns them privileges through security constraints.

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

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

4. The assembler defines application-level security constraints and resolves potential conflicts between module-level configurations.

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

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

6. 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.

2

Overview of OC4J Security

This chapter introduces the Oracle Containers for J2EE (OC4J) security implementation. This implementation allows developers to integrate authentication, authorization, and delegation services with their applications.

The key component of this implementation is the Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider, which supports the JAAS specification.

This chapter introduces OC4J security, including the OracleAS JAAS Provider, and related key topics:

- Introducing the OracleAS JAAS Provider and Security Providers
- Authentication in the OC4J Environment
- Authorization in the OC4J Environment
- Overview of Security Role Mapping

See Also:

- For Oracle Application Server general security information and infrastructure, the 10.1.2 version of the *Oracle Application Server Security Guide*, a document that is not part of the 10.1.3 documentation set but is available at the following location:

  `http://www.oracle.com/technology/documentation/appserver1012.html`

Introducing the OracleAS JAAS Provider and Security Providers

The JAAS framework and the Java 2 Security model form the foundation of JAAS, which Oracle supports through the Oracle Application Server Java Authentication and Authorization Service (JAAS) Provider. The OracleAS JAAS Provider is easily integrated with J2SE and J2EE applications that use the Java 2 Security model, and 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.

In addition to the OracleAS JAAS Provider, the other key aspect of the security framework for OC4J applications is support for several particular security providers: file-based, LDAP directory-based, external LDAP directory, and custom.

The rest of this section covers the following topics:

- Overview of the OracleAS JAAS Provider

Overview of the OracleAS JAAS Provider

The OracleAS JAAS Provider implements the JAAS Login Configuration Provider interface and the JAAS Policy Provider interface:

- The Login Configuration Provider implementation is involved in retrieving login module configuration information and ensuring that the appropriate login module is invoked for authentication. An XML file is used to store JAAS login module configurations.
- The Policy Provider implementation supports either of two repositories to store policies for authorization: an XML file or directory service. (This is as opposed to the Sun Microsystems Policy Provider implementation, for example, which uses the file `java2.security` as a policy repository.) Policies contain the rules, referred to as the permissions or privileges, that authorize a user to access and use resources, such as reading from or writing to a file.

Using OracleAS JAAS Provider, applications can enforce fine-grained access control upon resource users. The three key steps when a security-aware application is running in OC4J are the following:

1. Set up and invoke the login module, which involves the OracleAS JAAS Provider.
2. Authenticate the user attempting to log in, which is the role of the security provider.
3. Authorize the user by checking permissions for whatever the user is attempting to accomplish, which involves the OracleAS JAAS Provider.

By default, OracleAS JAAS Provider is configured as part of the OC4J product.

Note: In earlier releases, the term "JAZN" was used to refer to the OracleAS JAAS Provider. This term is no longer used in general, but still appears in code (such as class and package names) and the Admintool shell prompt.

Summary of JAAS Framework Features

Table 2–1 summarizes JAAS framework features implemented by the OracleAS JAAS Provider.

Table 2–1 JAAS Framework Features

Feature	Description	See Also
Authentication	■ Integrates with OracleAS Single Sign-On for 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 login module.	"Authentication in the OC4J Environment" on page 2-5
Declarative model	■ Integrates J2EE deployment descriptors, such as. `web.xml`, with JAAS security.	
Authorization	■ Supports the J2EE authorization model. ■ Supports the JAAS authorization model. ■ Supports the Java Authorization Contract for Containers	"Authorization in the OC4J Environment" on page 2-7
Realm management	■ The package `oracle.security.jazn.realm` is provided to support user and role management.	
Policy management	■ The package `oracle.security.jazn.policy` is provided for administration of authorization policy.	
Administration	■ Supports administration and configuration using Oracle Enterprise Manager 10*g* or a command-line tool (the OracleAS JAAS Provider Admintool).	"Tools for Oracle Application Server and OracleAS JAAS Provider" on page 3-2
`JAZNUserManager`	■ Supplies a security provider implementation that integrates with the file-based provider, Oracle Identity Management, and COREid Access. This class is in the `oracle.security.jazn.oc4j` package.	

Supported Security Providers

Oracle Application Server supports the following security providers. Each security provider is associated with an appropriate login module (`RealmLoginModule` for the file-based and LDAP-based providers), which is effectively part of the security provider. In addition, each security provider uses a repository for secure and centralized storage, retrieval, and administration of data that consists of realm information (users and roles) and JAAS policy information (permissions).

- File-based (XML-based) provider

  The file-based provider, discussed in Chapter 7, "File-Based Security Provider", is a fast, lightweight JAAS login module implementation that uses an XML repository. User, role, and policy information is typically stored in the OC4J instance-level file `system-jazn-data.xml`.

  This is the default security provider.

- LDAP-based provider: Oracle Identity Management

  This is the security provider if you want to use the Oracle Internet Directory (OID) as your user repository, with or without Oracle Application Server Single Sign-On, as described in Chapter 6, "Oracle Identity Management Security Provider". The Oracle Identity Management provider stores user, role, realm, and policy information in Oracle Internet Directory, which is based on the Lightweight Directory Access Protocol (LDAP) for centralized storage of information.

  This security provider, intended for production environments, is scalable, secure, enterprise-ready, and integrated with OracleAS Single Sign-On.

OC4J must be associated with an Oracle Internet Directory instance in order to use Oracle Identity Management.

- External LDAP providers

  Oracle Application Server supports external (third-party) LDAP providers such as Sun Java System Directory Server or Microsoft Active Directory, as described in Chapter 9, "External LDAP Security Providers". The external LDAP provider implements a custom login module, `LDAPLoginModule`.

- Custom security providers

  Oracle Application Server allows you or a third party to implement a custom security provider using custom login modules to implement special authentication functionality for an application, as described in Chapter 8, "Login Modules". A custom login module implements the standard JAAS login module interface. You can configure custom login modules when you deploy an application through Oracle Enterprise Manager 10*g*. The configuration is stored in the OC4J `system-jazn-data.xml` file.

  Support for custom login modules is implemented through an extension of the file-based provider.

- COREid Access

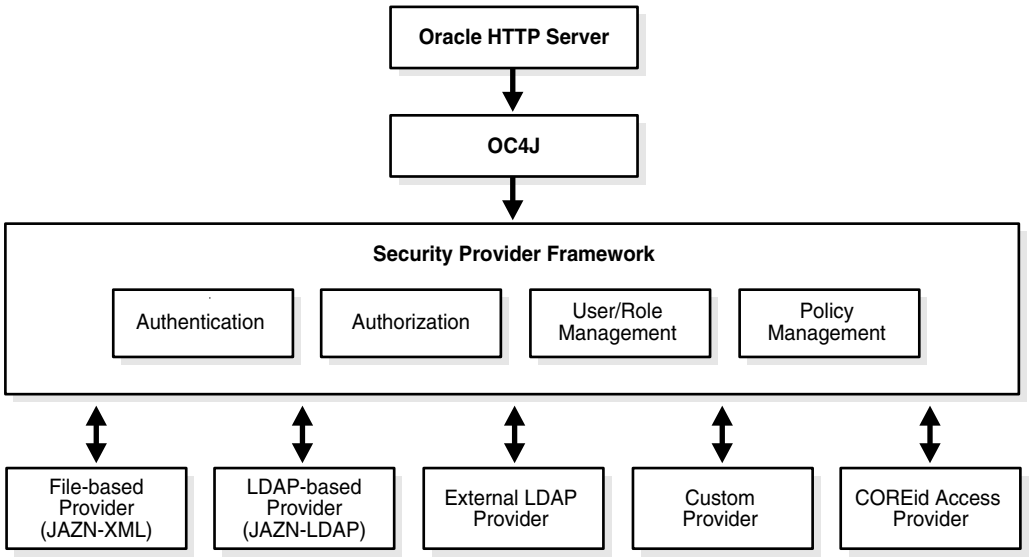
  Beginning with the OC4J 10.1.3 implementation, an additional choice for security provider is COREid Access, part of Oracle COREid Access and Identity, as described in Chapter 10, "COREid Access Security Provider". This is an enterprise-class authentication, authorization, and auditing solution that provides centralized security administration. This includes functionality for access control, single sign-on (separate from OracleAS Single Sign-On), personalization, and user profile management in heterogeneous application environments across a variety of application servers, legacy applications, and databases. Use `CoreIDLoginModule` with this security provider.

Note: Note the following terminology in this document:

- The terms "file-based provider" and "XML-based provider" (sometimes referred to as "JAZN-XML") are equivalent.
- In the context of Oracle Application Server, the term "LDAP-based provider" (also sometimes "JAZN-LDAP") refers to Oracle Identity Management and its repository, the Oracle Internet Directory.
- In the OC4J 10.1.3 implementation, the term "custom security provider" is essentially synonymous with "custom login module".

Figure 2–1 shows how the supported security providers interact with the overall security provider framework.

Figure 2–1 OC4J Security Architecture



Support for DataSourceUserManager

As an alternative to using security providers introduced in the preceding section, the OC4J 10.1.3 implementation continues to support the `DataSourceUserManager` class, for retrieving and managing user data from a database. The database must be specified as a data source, for which you provide a JNDI location in your configuration. The configuration also specifies the relevant database tables and fields.

Important: The `DataSourceUserManager` class is deprecated in the OC4J 10.1.3 implementation, but is still supported for backward compatibility. In future releases, it will be replaced by a JAAS-based login module.

See Also:

- "Tasks for DataSourceUserManager" on page 5-17 for information about `DataSourceUserManager` methods, initialization parameters, and configuration

Authentication in the OC4J 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 OC4J environment is performed by one of the following:

- OracleAS Single Sign-On
- COREid Access (optionally including single sign-on)
- OracleAS JAAS Provider `RealmLoginModule` login module or desired custom login module (for non-SSO environments)

Before HTTP requests can be dispatched to the target servlet, the OracleAS JAAS Provider `JAZNUserManager` (or, alternatively, a developer-supplied `UserManager` implementation), which coordinates authentication, gets the authenticated user

information (set by `mod_osso` for SSO, for example) from the HTTP request object, and sets the JAAS subject in OC4J.

Important: Developer-supplied `UserManager` classes are deprecated in the OC4J 10.1.3 implementation and will be desupported in a future release. Use custom login modules instead.

Several different methods of authentication can be used for a J2EE Web application. The following are standard:

- **Basic**

  With basic authentication, the user is prompted directly for a user name and password, without going through OracleAS Single Sign-On. A login module (such as `RealmLoginModule`, for example) is used to generate a login dialog.

- **Digest**

  With the digest authentication mechanism, the password that a client presents to authenticate itself is encrypted through the use of an MD5 digest. This is transmitted in the request message. From a user perspective, digest authentication behaves in the same way as basic authentication. (The digest method is not supported for an external LDAP provider or custom provider.)

- **Form**

  When the user attempts to access a protected resource through form-based authentication, OC4J displays an application-specific login screen, prompting for user name and password. (The form method is not supported for a custom provider.)

- **Client-cert**

  This method authenticates the client through HTTPS. The user must possess a public key certificate.

The following are Oracle-specific:

- **SSO**

  For this authentication method, OracleAS Single Sign-On is used to authenticate users.

- **COREIDSSO**

  For this authentication method, COREid single sign-on (distinct from OracleAS Single Sign-On) is used to authenticate users.

Note: For either the file-based provider or Oracle Identity Management, we recommend digest authentication as a more secure solution than basic authentication.

See Also:

- "Specifying the Authentication Method (auth-method)" on page 13-1

Authorization in the OC4J Environment

There are three main aspects of authorization in OC4J:

- J2EE authorization
- OracleAS JAAS Provider authorization and "JAAS mode"
- Java Authorization Contract for Containers (an enhancement of the J2EE authorization model)

See Also:

- "JAAS Security Model Versus J2EE Security Model" on page 1-3 for an overview of fine-grained versus coarse-grained authorization

J2EE Authorization

There are standard J2EE implementations that allow servlets and EJBs to retrieve information about a user or caller. You can use these methods in determining if a user or caller should be allowed to access a resource.

A Web application can use the following methods in a `javax.servlet.http.HttpServletRequest` instance:

- `User getUserPrincipal()`

  Returns a principal object containing the name of the authenticated user making the request. The OC4J implementation returns an instance of `oracle.j2ee.security.User`, which extends the standard `java.security.Principal`.

- `String getRemoteUser()`

  Returns the login name of the authenticated user making the request (or `null` if the user is not authenticated).

- `boolean isUserInRole(String rolename)`

  Determines whether the authenticated user making the request is a member of the specified role.

An EJB application can use the following methods in a `javax.ejb.EJBContext` instance:

- `Principal getCallerPrincipal()`

  Returns a principal object that identifies the caller. The OC4J implementation returns an instance of `oracle.j2ee.security.User`, which extends the standard `java.security.Principal`.

- `boolean isCallerInRole(String rolename)`

  Determines whether the caller is a member of the specified role.

JAAS Authorization and JAAS Mode

OracleAS JAAS Provider allows any protected resource to be modeled using Java permissions. The Java permission model (and associated `java.security.Permission` class) is extensible and allows a flexible way to define fine-grained access control.

OracleAS JAAS Provider supports the following features related to fine-grained authorization:

- JAAS mode, which is related to standard `Subject.doAs()` and `Subject.doAsPrivileged()` functionality for either servlets or EJBs
- OracleAS JAAS Provider realm and policy API features
- Features for granting permissions
- Features for checking permissions

See Also:

- "Tasks for JAAS Mode and Authorization" on page 5-6 for related task-oriented steps and examples
- For standard JAAS programming reference information:

  `http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASRefGuide.html`

- For detailed JAAS API information, `java.security.*` packages and `javax.security.auth.*` packages:

  `http://java.sun.com/j2se/1.4.2/docs/api/`

- *JAAS Provider API Reference* for Javadoc for the OracleAS JAAS Provider

Introduction to JAAS Mode

The OC4J 10.1.3 implementation provides a fine-grained authorization feature called *JAAS mode*, which is related to standard functionality of the static methods `doAs()` and `doAsPrivileged()` in the `Subject` class. These methods work as follows:

- `Object doAs(Subject, PrivilegedAction)`

  Performs the specified privileged action (a computation to be performed with privileges enabled) as the specified subject. This method associates the subject with the access control context (`AccessControlContext` instance) of the current thread, appending the subject's permissions to the permission of that access control context. The returned object is what is returned by the `run()` method of the privileged action. There is also a variation that takes `java.security.PrivilegedExceptionAction` instead of `java.security.PrivilegedAction`, for computations that can throw checked exceptions.

- `Object doAsPrivileged(Subject, PrivilegedAction, AccessControlContext)`

  This method has the same functionality as the `doAs()` method, but within the specified access control context instead of using the access control context of the thread, appending the subject's permissions to the permissions of the specified context.

These methods are used in your application according to your JAAS mode setting. Set JAAS mode in the `jaas-mode` attribute of the `<jazn>` element in the `orion-application.xml` file of your application. JAAS mode determines `doAs()` or `doAsPrivileged()` usage as follows:

- With the setting `jaas-mode="doAs"`, application modules are executed under `Subject.doAs()`.

- With the setting `jaas-mode="doAsPrivileged"`, modules are executed under `Subject.doAsPrivileged()`, therefore not being limited by the access-control restrictions of the server.
- With the setting `jaas-mode="null"` (default), modules are executed under neither method.

Because `jaas-mode` is set in the application-level `orion-application.xml` file, it will affect any Web module or EJB in the application.

Note: JAAS mode replaces `runas-mode` and `doasprivileged-mode` settings in the `<jazn-web-app>` element of `orion-application.xml` or `orion-web.xml`.

These settings are deprecated in the OC4J 10.1.3 implementation, but still supported for backward compatibility.

The setting `jaas-mode="null"` is equivalent to `runas-mode="false"`; `jaas-mode="doas"` is equivalent to `runas-mode="true"` with `doasprivileged-mode="false"`; `jaas-mode="doAsPrivileged"` is equivalent to `runas-mode="true"` with `doasprivileged-mode="true"`.

See Also:

- "Security Managers and Access Control" on page 1-6 for an introduction to access controllers and access control contexts
- "Tasks for JAAS Mode and Authorization" on page 5-6

OracleAS JAAS Provider Realm and Policy Features

This section discusses OracleAS JAAS Provider classes and methods related to JAAS authorization.

An instance of the `oracle.security.jazn.JAZNConfig` class represents a configuration of the `<jazn>` element. This class includes the following methods:

- `JAZNConfig getJAZNConfig()`

  This static method of the `JAZNConfig` class returns a `JAZNConfig` instance.

- `RealmManager getRealmManager()`

  This instance method of the `JAZNConfig` class returns a `RealmManager` instance, which is used to manage realms.

The `oracle.security.jazn.realm.RealmManager` class includes the following instance method:

- `Realm getRealm(String)`

  This method returns a realm object for the specified realm name.

An instance of the `oracle.security.jazn.realm.Realm` class provides access to the store of users and roles for the particular realm. In the `realm` package, user management is defined by the `UserManager` interface and role management is defined by the `RoleManager` interface. The `Realm` class includes the following instance methods:

- `UserManager getUserManager()`

This method returns a `UserManager` instance, which you can use to manage users in this realm.

- `RoleManager getRoleManager()`

  This method returns a `RoleManager` instance, which you can use to manage roles in this realm.

Use an `oracle.security.jazn.realm.UserManager` instance to manage (add, retrieve, or remove) users in the realm. This interface includes the following method:

- `RealmUser getUser(String)`

  This method returns a user object, for the specified name of a user in the realm.

Features for Granting Permissions

The `JAZNConfig` class, mentioned in the preceding section, also has the following method:

- `JAZNPolicy getPolicy()`

  This method returns an `oracle.security.jazn.policy.JAZNPolicy` instance, which represents the repository of authorization policies.

The `JAZNPolicy` interface includes the following methods:

- `void grant(Grantee, Permission)`

  This method grants the specified permission to the specified grantee, taking as input an `oracle.security.jazn.policy.Grantee` instance and a `java.security.Permission` instance.

- `void revoke(Grantee, Permission)`

  This method revokes the specified permission for the specified grantee.

- `boolean hasPermission(Grantee, Permission)`

  This method determines whether the specified grantee has the specified permission.

The `Grantee` constructor takes a `Principal` instance as input:

- `new Grantee(Principal)`

There are various types of permissions that can be constructed, including the following. (These are all standard JDK features.)

- `new Permission(String permname)`

  The `java.security.Permission` constructor takes a string to specify the name of the permission.

- `new BasicPermission(String permname)`

  The `java.security.Permission` constructor takes a string to specify the name of the permission.

- `new FilePermission(String path, String actions)`

  The `java.security.FilePermission` constructor takes one string to specify the path of the file in question, and another string that is a comma-delimited list of permissible actions. Supported actions are "read", "write", "execute", and "delete".

- `new AllPermission()`

An instance of the `java.security.AllPermission` class is a permission that represents all other permissions. Its constructor takes no parameters.

Notes:

- While there are standard mechanisms for checking permissions (as described in the next section), JAAS does not provide standard mechanisms for managing users and granting permissions. This is why the classes and methods described in this section are Oracle-specific.
- You can also grant, list, and revoke permissions using the OracleAS JAAS Provider Admintool, as discussed in Appendix C, "OracleAS JAAS Provider Admintool Reference".

Features for Checking Permissions

Access control, such as checking permissions, was introduced in "Security Managers and Access Control" on page 1-6. You can retrieve and check permissions using a policy object.

The abstract class `javax.security.auth.Policy` includes the following methods:

- `Policy getPolicy()`

  This static method returns a `Policy` instance.

- `PermissionCollection getPermissions(Subject, CodeSource)`

  This method, given a `javax.security.auth.Subject` instance and a `java.security.CodeSource` instance, returns a `java.security.PermissionCollection` instance that indicates the set of permissions allowed, given the characteristics of the protection domain. The codesource field can be `null`.

The `PermissionCollection` class includes the following method:

- `boolean implies(Permission)`

  This abstract method indicates whether the specified permission is implied by the set of permissions in the permission collection.

Note: The `javax.security.auth.Policy` class is deprecated in JDK 1.4 but fully supported in the OC4J 10.1.3 implementation and still supported by the Sun Microsystems JDK and J2SE. As of this release, the replacement `java.security.Policy` class is not yet supported by OC4J.

See Also:

- "Security Policies" on page 1-5 for an introduction to codesources

OracleAS JAAS Provider Permission Classes

Table 2–2 summarizes permission classes supplied by the OracleAS JAAS Provider.

See Also:

- For information about the classes discussed, the OracleAS JAAS Provider Javadoc: *JAAS Provider API Reference*.

Table 2–2 OracleAS 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` and `dropRealm`). `RealmPermission` extends the class `java.security.Permission`, and is used like any regular Java permission. A `RealmPermission` instance associates a realm name (target name) with a list of actions.

Implementation of Java Authorization Contract for Containers

The OC4J 10.1.3 implementation supports the Java Authorization Contract for Containers (Java ACC), as specified in JSR-115. This is a contract between containers and authorization service providers, allowing authorization to be decoupled from the container. OC4J authorization functionality is delegated to a standard Java ACC provider. The contract defined in JSR-115 interacts with an application server container, deployment tools, and policy provider, and is divided into the following subcontracts:

- Provider configuration subcontract
- Policy configuration subcontract
- Policy decision and enforcement subcontract

Java ACC provides new `java.security.Permission` class implementations that adhere to the J2EE authorization model. Under Java ACC, access decisions by the container are made according to operations on instances of these `Permission` classes. Java ACC defines the ways that policy providers make use of the new permission classes to address requirements of the J2EE authorization model, as follows:

- Roles defined as collections of permissions
- Granting the permissions of a role to a principal
- Determining whether a principal has been granted the permissions of a role

Note that as a result of the Java ACC contract, J2EE security constraints are translated into Java 2 permissions, so that the J2EE security model now fully leverages the J2SE security model. Yet Java ACC still fully preserves the existing J2EE declarative security model as well as the J2EE security API.

See Also:

- "Using the Java Authorization Contract for Containers" on page 5-9
- For general information about Java ACC:

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

Overview of Security Role Mapping

Two distinct role types are available to application developers creating secure applications in J2EE environments: J2EE roles and JAAS roles. OC4J allows you to map a J2EE role to a JAAS role, so that a user who is a member of a given J2EE role has access to resources that are accessible from the associated JAAS role.

The basic steps in security roles and mapping are as follows:

1. Specify logical J2EE security roles, through standard J2EE functionality, in deployment descriptors such as `web.xml` and `ejb-jar.xml`. There is nothing OC4J-specific in this step. A role is declared in a `<security-role>` element.
2. As applicable, specify security role references to link roles in your application code to roles declared through `<security-role>` elements. This is accomplished in standard deployment descriptors through `<security-role-ref>` elements. Through this mechanism, you can adjust your definitions of logical security roles without having to change your application code, then simply map logical roles to application roles as desired. There is nothing OC4J-specific in this step.
3. Configure JAAS roles, or use default roles. For the file-based provider, for example, JAAS roles are reflected in the OC4J `system-jazn-data.xml` file, or optionally in an application-specific `jazn-data.xml` file.
4. Map J2EE roles to JAAS roles. You can accomplish this through Application Server Control, and mappings are reflected in `<security-role-mapping>` elements in `orion-application.xml`, `orion-ejb-jar.xml`, or `orion-web.xml`.

See Also:

- "Mapping J2EE Security Roles to JAAS Roles" on page 5-13
- "Web Application Security Role Configuration" on page 13-6
- "Authenticating and Authorizing EJB Applications" on page 14-2

3

Overview of Security Administration and Configuration

This section provides an overview of features and tools for security administration and configuration in OC4J and Oracle Application Server, covering the following topics:

- General OC4J Deployment and Configuration Features
- Tools for Oracle Application Server and OracleAS JAAS Provider
- JMX and MBeans Administration
- Overview of Configuration Files and Key Elements
- OC4J System Application
- Summary of OC4J Accounts
- Summary of Configuration Repositories and Security Management Tools

General OC4J Deployment and Configuration Features

OC4J supports the following standards for deploying and managing applications in a J2EE environment:

- *Java Management Extensions (JMX) 1.2* specification allows standard interfaces to be created for managing resources, such as services and applications, in a J2EE environment. The OC4J implementation of JMX provides a user interface that you can use to completely manage an OC4J server and applications running within it.
- *Java 2 Platform, Enterprise Edition Management Specification* (JSR-77) allows objects known as *MBeans* (managed beans) to be created for runtime management of applications in a J2EE environment. In OC4J, you can directly access MBeans through the System MBean Browser in Oracle Enterprise Manager 10*g*, but many of their properties are exposed in a more user-friendly way through other features of Enterprise Manager.
- *Java 2 Enterprise Edition Deployment API Specification* (JSR-88) defines a standard API for configuring and deploying J2EE applications and modules into a J2EE-compatible environment. The OC4J implementation includes the ability to create or edit a *deployment plan* containing the OC4J-specific configuration data needed to deploy a component into OC4J.

See Also:

- *Oracle Containers for J2EE Deployment Guide* and *Oracle Containers for J2EE Configuration and Administration Guide* for general information about OC4J deployment, configuration, and administration

Tools for Oracle Application Server and OracleAS JAAS Provider

Managing security in the J2SE and J2EE environments involves creating and managing realms, users, roles, permissions, and policy. The following Oracle tools are involved in managing security configuration:

- Oracle Enterprise Manager 10*g* Application Server Control is used for overall security administration and configuration during and after deployment, and to manage the file-based provider.
- OracleAS JAAS Provider Admintool is used to manage the file-based provider, and also to manage policies and login modules for any security provider.
- Oracle Identity Management and Oracle Internet Directory tools: Oracle Delegated Administration Services (DAS) and Oracle Directory Manager (`oidadmin`) are used to manage users and roles in Oracle Internet Directory for Oracle Identity Management.

These tools will be summarized more thoroughly in the subsections that follow.

Note: Wherever possible, Oracle Enterprise Manager 10*g* Application Server Control should be your first-choice tool to administer OC4J, including OC4J security. For features that the Application Server Control does not support, you can, as applicable, use the OracleAS JAAS Provider Admintool. Occasionally, you will have to directly manipulate a configuration file, particularly the instance-level `jazn.xml` file (discussed in "The jazn.xml File" on page 3-9).

See Also:

- "Summary of Configuration Repositories and Security Management Tools" on page 3-12

Overview of Enterprise Manager

Typically, you should use Oracle Enterprise Manager 10*g* Application Server Control to deploy and administer your applications. The user interface for this is the Application Server Control Console. Application Server Control includes features for the following:

- Deploying an application to OC4J. This includes a deployment plan editor. For security, this also includes features to specify the security provider and security role mapping during deployment.
- Using the System MBean Browser for MBean configuration and operations (further discussed in "JMX and MBeans Administration" on page 3-5). Also be aware, however, that many parameters corresponding to MBeans properties are exposed through other pages of the Application Server Control Console. Avoid direct manipulation of OC4J MBeans whenever possible.

- Changing to a different security provider after deployment, or updating security provider settings.
- Performing OC4J runtime administration and configuration.

OC4J-specific XML configuration files are updated automatically by OC4J when you use the Application Server Control Console.

Notes:

- In standalone OC4J you also have the option of using the command-line OC4J `admin_client.jar` tool, which operates through the OC4J `system` application, to deploy and bind your J2EE applications. Alternatively, if you use the Oracle JDeveloper tool to develop your application, you can use it to deploy the application and any resource adapters as well.
- Whenever a configuration change is made using Application Server Control or the OC4J security provider MBean, the application must be restarted. Until the application is restarted, all other operations of the security provider MBean are invalidated and will return an error message.

See Also:

- *Oracle Application Server Administrator's Guide* for more information about Application Server Control
- *Oracle Containers for J2EE Configuration and Administration Guide* for information about the admin_client.jar utility
- "OC4J System Application" on page 3-10

Overview of the OracleAS JAAS Provider Admintool

The OracleAS JAAS Provider Admintool, for use during development, is a lightweight Java application with the following management features:

- For the file-based provider: administration for users, roles, policies, and login modules
- For Oracle Identity Management: administration for policies and login modules, plus read-only access to users and roles
- For external LDAP providers: administration for policies and login modules
- For custom security providers: administration for policies and login modules

Admintool functions can be called directly from a command line or through an interactive shell. The Admintool is located in *ORACLE_HOME*`/j2ee/home/jazn.jar`.

The general command-line syntax is as follows:

```
% java -jar jazn.jar [-user username -password pwd] [option1 option2 ... ]
```

When you use the Admintool for the file-based provider, by default it updates the `system-jazn-data.xml` file in the *ORACLE_HOME*`/j2ee/home/config` directory.

Note: In general, changes made by the Admintool are not effective until you restart OC4J.

See Also:

- Appendix C, "OracleAS JAAS Provider Admintool Reference"

Overview of Oracle Identity Management and Oracle Internet Directory Tools

This section provides an overview of tools to manage Oracle Internet Directory, when using Oracle Identity Management as your security provider.

Overview of Delegated Administration Services

Delegated administration is an important feature of the Oracle Identity Management infrastructure. It enables you to store all data for users, groups, and services in a central directory, while distributing the administration of that data to various administrators and end users. It does this in a way that respects the various security requirements in your environment.

Suppose, for example, that your enterprise stores all user, group, and services data in a central directory, and requires one administrator for user data, and another for the e-mail service. Delegated administration as provided by the Oracle Identity Management infrastructure enables different administrators with different security requirements to administer centralized data in a way that is both secure and scalable. Privileges can be delegated with Oracle Delegated Administration Services to (among other things) create, edit, and delete users and groups; assign privileges to users and groups; and manage services and accounts.

Oracle Delegated Administration Services is a set of pre-defined, Web-based units for performing directory operations on behalf of a user. It frees directory administrators from the more routine directory management tasks by enabling them to delegate specific functions to other administrators and to end users. It provides most of the functionality that directory-enabled applications require, such as creating a user entry, creating a group entry, searching for entries, and changing user passwords.

You can use Oracle Delegated Administration Services to develop your own tools for administering application data in the directory. Alternatively, you can use the Oracle Internet Directory Self-Service Console, a tool based on Delegated Administration Services. This tool comes ready to use with Oracle Internet Directory.

See Also:

- *Oracle Identity Management Guide to Delegated Administration*

Overview of Oracle Directory Manager

Oracle Directory Manager is an online administration tool with a Java-based graphical user interface that you can use to administer Oracle Internet Directory. The executable file is located in the *ORACLE_HOME*`/bin` directory, and you can run it from the command line as follows:

```
% oidadmin
```

In general, any directory-specific configuration or maintenance task not available through Application Server Control can be accomplished through Oracle Directory Manager (as well as various command-line interfaces supplied with Oracle Internet Directory).

You can use Oracle Directory Manager for tasks such as the following:

- Configuring realms
- Specifying password policies

- Configuring the Oracle Directory Synchronization Service and Oracle Internet Directory connectors and agents

You can also manage features such as attribute uniqueness, plug-ins, garbage collection, change logs, replication, query optimization, debug logging, and access control lists.

See Also:

- *Oracle Internet Directory Administrator's Guide* for general information about Oracle Directory Manager

JMX and MBeans Administration

OC4J support for the JMX specification allows standard interfaces to be created for managing resources dynamically in a J2EE environment. The OC4J implementation of JMX provides a JMX client, the System MBean Browser, that you can use to manage an OC4J instance through MBeans that are provided with OC4J.

An MBean is a Java object that represents a JMX manageable resource. Each manageable resource within OC4J is managed through an instance of the appropriate MBean. Each MBean provided with OC4J exposes a management interface that is accessible through the System MBean Browser in the Application Server Control Console. You can set MBean attributes, execute operations to call methods on an MBean, subscribe to notifications of errors or specific events, and display execution statistics.

To access the browser from the OC4J Home page, select the **Administration** tab and then, under the list of tasks, go to the JMX task "System MBean Browser". From the browser, you can do the following:

- Select the MBean of interest in the left-hand frame.
- Use the **Attributes** tab in the right-hand frame to view or change attributes. A settable attribute has a field where you can type in a new value. Then apply the change.
- Use the **Operations** tab in the right-hand frame to invoke methods on the MBean. Select the operation of interest. In the Operation window, you can invoke it with specified parameter settings.
- Use the **Notifications** tab (where applicable) in the right-hand frame to subscribe to notifications. You can select each item for which you want notification, and then apply the changes.
- Use the **Statistics** tab (where applicable) in the right-hand frame to display execution statistics.

Be aware that MBeans and their attributes vary regarding when changes take effect. In the *runtime model*, changes take effect immediately. In the *configuration model*, some changes take effect when the resource is restarted, others when the application is restarted, and still others when OC4J is restarted. There is also variation in whether changes are persisted.

Overview of Configuration Files and Key Elements

This section provides an overview of the following key XML files and elements for security configuration:

- The orion-application.xml File (<jazn> and <jazn-web-app> Elements)

- The system-application.xml File
- The system-jazn-data.xml File
- Application-Specific jazn-data.xml File (Optional)
- The jazn.xml File

Note: In general, you should use the Application Server Control Console or OracleAS JAAS Provider Admintool (both discussed earlier in this chapter) for configuration and administration, instead of manipulating configuration files directly. Using these tools results in the appropriate entries automatically being made in the configuration files.

See Also:

- "Summary of Configuration Repositories and Security Management Tools" on page 3-12

The orion-application.xml File (<jazn> and <jazn-web-app> Elements)

The OC4J `orion-application.xml` file is for general (not just security-related) application-level configuration. Settings in this file apply across a single J2EE application (EAR file).

For security settings in `orion-application.xml`, there is the `<jazn>` element. In particular, this element can specify the security provider, the user and role repository location, and the default realm for the application, as in the following example to use the file-based provider:

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

(The `system-jazn-data.xml` file, discussed in "The system-jazn-data.xml File" on page 3-7, would actually be the repository by default, but is specified here for illustrative purposes.)

A subelement of `<jazn>` in `orion-application.xml` is the `<jazn-web-app>` element, which is where you specify OC4J-specific authentication methods (using the `auth-method` attribute) for Web applications.

Note: If there is no `<jazn>` element in `orion-application.xml`, the security provider settings defer to those of the instance-level `jazn.xml` file (where the file-based provider with the `system-jazn-data.xml` repository and `jazn.com` default realm are the default settings).

The system-application.xml File

The OC4J configuration file is associated with the OC4J `system` application, which is described in "OC4J System Application" on page 3-10. For the `system` application, `system-application.xml` is equivalent to the `orion-application.xml` file for a deployed application.

The `system-application.xml` file, through its `<jazn>` element, specifies the file-based security provider for OC4J instance-level user and role settings (including

some used for special OC4J functionality). The `system-application.xml` file points to the `system-jazn-data.xml` file (described in the next section), which is also instance-level, as the repository for these settings, which are located under the `<jazn-realm>` element.

By default, OC4J expects the `system-application.xml` to be in the *`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/config` directory.

The system-jazn-data.xml File

The `system-jazn-data.xml` file is a new file in the OC4J 10.1.3 implementation. This file (as well as `system-application.xml`) is associated with the OC4J `system` application, which is described in "OC4J System Application" on page 3-10.

The `system-application.xml` file points to the `system-jazn-data.xml` file as the repository for OC4J instance-level user and role settings (located under the `<jazn-realm>` element) for the file-based provider, which uses `system-jazn-data.xml` for authentication and authorization. (Note that the file-based provider is the default security provider.)

The `system-jazn-data.xml` file also stores JAAS login module configuration (under the `<jazn-loginconfig>` element). In addition, by default, it stores instance-level policy and permission configuration (under the `<jazn-policy>` and `<jazn-permission-classes>` elements).

By default, OC4J expects the `system-jazn-data.xml` file to be in the *`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/config` directory.

There is a persistence mode that governs how often changes are written to the `system-jazn-data.xml` file and, if applicable (for the file-based provider), to an application-level `jazn-data.xml` file There are three possible values for persistence, according to the `<jazn>` element `persistence` attribute in either the instance-level `jazn.xml` file or application-level `orion-application.xml` file:

- "`NONE`": Do not write changes to `system-jazn-data.xml`.
- "`ALL`": Write changes after every modification.
- "`VM_EXIT`" (default): Write changes when the Java Virtual Machine exits.

Here is an example:

```
<jazn provider="XML" persistence="ALL" ... >
   ...
</jazn>
```

Notes:

- In previous releases, `system-jazn-data.xml` was named `jazn-data.xml`. For the file-based provider, you can still use a file named `jazn-data.xml` to store user and role information, but this file would be application-specific. See the next section, "Application-Specific jazn-data.xml File (Optional)".
- Settings in the `system-jazn-data.xml` file can be manipulated using Application Server Control or the OracleAS JAAS Provider Admintool.
- Changes made to the `system-jazn-data.xml` file are visible to all applications that use it.
- The `system-jazn-data.xml` file contains accounts for predefined OC4J users and roles. See "Predefined OC4J Accounts in system-jazn-data.xml" on page 7-12.
- White space in element settings is significant, such as the differences between the following:

```
<name>scott</name>
<name>scott </name>
<name> scott</name>
<name> scott </name>
```

Application-Specific jazn-data.xml File (Optional)

When you use the file-based provider, you can optionally still use a `jazn-data.xml` file as the user and role repository, but this file is application-specific. You can specify its location in the `<jazn>` element of the `orion-application.xml` file:

```
<jazn provider="XML" location="path/jazn-data.xml">
   ...
</jazn>
```

Here is the default location:

ORACLE_HOME/j2ee/*instance_name*/application-deployments/*app_name*

Note that if `orion-application.xml` is configured exactly as follows, but the `jazn-data.xml` file is not packaged with the application, then one will be created during deployment:

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

Persistence mode for changes to the repository, described in the preceding section for `system-jazn-data.xml`, also affects `jazn-data.xml`.

Notes:

- Think of the application-specific `jazn-data.xml` file as a repository, not as a configuration file.
- White space in element settings is significant, such as the differences between the following:

```
<name>scott</name>
<name>scott </name>
<name> scott</name>
<name> scott </name>
```

The jazn.xml File

The `jazn.xml` file, located in the *ORACLE_HOME*`/j2ee/`*instance_name*`/config` directory, is an OC4J instance-level configuration file for the OracleAS JAAS Provider. It specifies the instance-level security provider and repository for policy and permission settings. The main element of the `jazn.xml` file is the `<jazn>` element, with largely the same functionality as discussed earlier for the `orion-application.xml` file for application-level settings.

By default, `jazn.xml` specifies the file-based provider, with `system-jazn-data.xml` as the repository and `jazn.com` as the default realm:

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

The `jazn.xml` file for the OC4J `home` instance, referred to as the *bootstrap* `jazn.xml` file, is typically located in the *ORACLE_HOME*`/j2ee/home/config` directory. It is read at OC4J startup and used by the OracleAS JAAS Provider runtime. Without a valid `jazn.xml` file, the OracleAS JAAS Provider cannot begin running.

If you use Application Server Control to associate OC4J with an Oracle Internet Directory instance in order to use the Oracle Identity Management security provider, then the `<jazn>` element of the bootstrap `jazn.xml` file is updated appropriately for the Oracle Internet Directory instance. For example:

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   ...
</jazn>
```

Note: If changes are made to `jazn.xml` after OC4J starts up, they have no effect on the OracleAS JAAS Provider.

You can optionally use a system property to specify an alternative location for the bootstrap `jazn.xml` file. When the OracleAS JAAS Provider starts, it searches for `jazn.xml` in the following order, stopping the search as soon as it finds one:

1. Location specified by the system property `oracle.security.jazn.config`
2. Location specified by the system property `java.security.auth.policy`
3. *J2EE_HOME*`/config`, where *J2EE_HOME* is specified by the system property `oracle.j2ee.home`

4. *ORACLE_HOME*/j2ee/home/config, where *ORACLE_HOME* is specified by the system property oracle.home (this is generally the same location as *J2EE_HOME*/config)
5. ./config

Sample jazn.xml Files

Here are sample jazn.xml files, first with the default configuration for the file-based provider:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jazn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/jazn-10_0.xsd"
 schema-major-version="10"
 schema-minor-version="0"
 provider="XML"
 location="./system-jazn-data.xml"
 default-realm="jazn.com"
/>
```

And for the LDAP-based provider:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jazn xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/jazn-10_0.xsd"
 schema-major-version="10"
 schema-minor-version="0"
 provider="LDAP"
 location="ldap://myoid.us.oracle.com:389"
/>
```

OC4J System Application

Be aware that the OC4J system application is a new internal component in the OC4J 10.1.3 implementation. It is auto-deployed to the OC4J instance the first time OC4J is started. This application was added primarily to address issues related to deploying or redeploying applications to OC4J.

The system application is at the root of the application hierarchy, and provides classes and configuration required at OC4J startup, including shared libraries imported by default by all other deployed applications. It is an OC4J internal component only. Applications cannot be deployed to it, nor can it be declared the parent of another application. The OC4J default application continues to serve as the default parent of all deployed applications.

The system application is configured to use the file-based provider for user and role settings, using system-jazn-data.xml for the repository. These settings should not be altered.

The OC4J-specific application descriptor for the system application is system-application.xml, with the same functionality as orion-application.xml for a deployed application. (For the default application, the OC4J-specific application descriptor is application.xml, not to be confused with the J2EE standard application.xml file for deployed applications.) These files are located in the *ORACLE_HOME*/j2ee/*instance_name*/config directory.

Note: By default, the OC4J `default` application also uses `system-jazn-data.xml`.

See Also:

- *Oracle Containers for J2EE Configuration and Administration Guide* for information about the OC4J `system` and `default` applications

Summary of OC4J Accounts

This section provides a summary of key OC4J accounts, covering the following topics:

- Predefined OC4J Accounts
- Activation of the oc4jadmin Account
- Configuring a New Administration Account
- Configuring an Anonymous User

Predefined OC4J Accounts

The OC4J 10.1.3 implementation includes predefined "bootstrap" users and roles for Oracle Internet Directory (when you use Oracle Identity Management) or the file-based provider.

For the file-based provider, the accounts are predefined in the `system-jazn-data.xml` file. For Oracle Internet Directory, they are created automatically as default accounts as part of the OC4J-OID association process.

The following predefined accounts are common to both providers:

- `oc4jadmin` user (formerly `admin`)
- `oc4j-administrators` role (formerly `administrators`), with member `oc4jadmin`, RMI permission "login" granted, and administration permission "administration" granted
- `oc4j-app-administrators` role (formerly `jmx-users`), with RMI permission "login" granted, to allow access to JMX application-level connectors

The following additional accounts are predefined for the file-based provider:

- `anonymous` user, initially deactivated

  Activate `anonymous` directly in the `system-jazn-data.xml` file, by changing the `deactivated` attribute of the `<user>` element from "`true`" to "`false`". Unlike for `oc4jadmin`, there is no support in the OracleAS JAAS Provider Admintool for activating `anonymous`.

- `users` role, for RMI/EJB access
- `jtaadmin` user, to allow transaction propagation over ORMI

Do not remove any of these accounts, or the administrative functions of the OracleAS JAAS Provider will not work.

See Also:

- The next section, "Activation of the oc4jadmin Account"

Activation of the oc4jadmin Account

The `oc4jadmin` account (formerly the `admin` account) is activated during Oracle Application Server installation, but is initially deactivated for the file-based provider in standalone OC4J. It is activated under the following circumstances:

- When standalone OC4J is first started (and you are prompted for a password)
- When you run the OracleAS JAAS Provider Admintool with the `-activateadmin` option

  You also specify the password as part of this command:

  ```
  % java -jar jazn.jar -activateadmin password
  ```

Configuring a New Administration Account

By default, `oc4jadmin` is the administration account for OC4J. When using either the file-based provider or Oracle Identity Management, you can specify a different administration account by setting the `admin.user` property in the instance-level `jazn.xml` file, as follows:

```
<jazn ... >
   ...
   <property name="admin.user" value="desired_admin_user_name" />
   ...
</jazn>
```

Then configure the account in the user repository with correct group membership and privileges, as appropriate. In Oracle Internet Directory, you can use DAS to create users and roles grant roles to users. To assign permissions, you can use the OracleAS JAAS Provider Admintool (or the OracleAS JAAS Provider MBean).

Configuring an Anonymous User

When using either the file-based provider or Oracle Identity Management, you can map an anonymous user to an existing user by setting the `anonymous.user` property in the instance-level `jazn.xml` file. For example, assuming there is a user `PUBLIC` in Oracle Internet Directory:

```
<jazn ... >
   ...
   <property name="anonymous.user" value="PUBLIC" />
   ...
</jazn>
```

Summary of Configuration Repositories and Security Management Tools

Management tools and configuration repositories have been discussed previously, but Table 3–1 summarizes the configuration repositories and the preferred management tools to use for the various types of configuration for each security provider.

Where applicable, Application Server Control is the preferred tool.

Table 3–1 Configuration Repositories and Preferred Management Tools

Security Provider	Repository and Management Tool for Users and Roles	Repository and Management Tool for Policies	Repository and Management Tool for JAAS Login Modules
File-based	system-jazn-data.xml (or application-specific jazn-data.xml) Use Application Server Control Console.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	n/a
Oracle Identity Management	Oracle Internet Directory Use DAS (or OracleAS JAAS Provider Admintool for read-only).	Oracle Internet Directory Use OracleAS JAAS Provider Admintool.	n/a
External LDAP	External (third-party) LDAP server Use tool supplied by provider.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	system-jazn-data.xml Use Application Server Control Console or OracleAS JAAS Provider Admintool.
Custom security provider	Custom security repository Use tool supplied by provider.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	system-jazn-data.xml Use Application Server Control Console or OracleAS JAAS Provider Admintool.
COREid Access	Oracle COREid Access and Identity Use COREid Access Manager.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	n/a

4

Java VM Security Settings for OC4J

This chapter discusses tasks related to configuring the security system at the OC4J instance level. It contains the following topics:

- Specifying an Alternate JAAS Policy Provider
- Specifying a Java 2 Security Manager and Policy File
- Enabling Subject Propagation for ORMI

> **Note:** Set system properties by using the `-D` command-line option when starting OC4J, as described in the *Oracle Containers for J2EE Configuration and Administration Guide*.

Specifying an Alternate JAAS Policy Provider

If you use the Java virtual machine shipped with Oracle Application Server, the OracleAS 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, because by default, the JVM uses the Sun Microsystems JAAS provider.

> **Note:** When you use OC4J, the JAAS configuration properties are set by default during OC4J startup, so in most circumstances there is no need to worry about setting these properties. You set them only when you are running a J2SE application outside OC4J.

You can specify Oracle-specific JAAS properties in a separate file that you supply to the JVM when you run OC4J.

Oracle supplies a default file, *ORACLE_HOME*`/j2ee/home/config/jazn.security.props`, that specifies the OracleAS JAAS Provider.

- To replace all security properties with the Oracle properties (note the two equals signs, "=="):

  ```
  java -Djava.security.properties==propfile
  ```

- To append the Oracle-specific properties to existing security properties:

  ```
  java -Djava.security.properties=propfile
  ```

Specifying a Java 2 Security Manager and Policy File

The OracleAS JAAS Provider checks permissions only when a security manager (`java.lang.SecurityManager` instance) has been installed. Specify a security manager in one of two ways:

- Calling `System.setSecurityManager()`
- Setting the system property `java.security.manager` when starting OC4J (or using this property with no setting to use the standard default security manager)

You can use either mechanism to install the default security manager or a custom security manager.

The permissions granted to particular classes by the default security manager are determined by reading a policy file. The default policy file is supplied as part of J2SE. You can specify a policy file explicitly using the system property `java.security.policy`, as in:

```
-Djava.security.policy=policyfilepath
```

Within an Oracle Application Server installation, OC4J instances run by default with no security manager. If you choose to install a security manager, you must specify one that does not interfere with normal OC4J functions.

The following example starts OC4J with the default security manager:

```
% java -Doracle.home=ORACLE_HOME -Djava.security.manager \
       -Djava.security.policy=ORACLE_HOME/j2ee/home/config/java2.policy \
       -jar oc4j.jar
```

See the following subsections for related information:

- Creating a Java 2 Policy File
- Using PrintingSecurityManager to Debug Java 2 Policy

See Also:

- "Security Managers and Access Control" on page 1-6 for an overview of security managers

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`.

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 JAAS 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 JAAS permissions to the demo to run JAAS 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;
}
```

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=ORACLE_HOME
```

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 security manager 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

To assist you in identifying all the required permissions for an application running on OC4J, Oracle provides a custom security manager, `PrintingSecurityManager`, that does not throw security exceptions. Instead, it prints a message specifying what exceptions the default security manager would have thrown. `PrintingSecurityManager` also generates the policy grants that would avoid the security exceptions.

Run `PrintingSecurityManager` as follows, assuming you run OC4J from *ORACLE_HOME*`/j2ee/home`:

```
% java -Xbootclasspath/p:lib/oc4j-psm.jar -Doracle.home=ORACLE_HOME \
   -Djava.security.manager=oracle.oc4j.security.PrintingSecurityManager \
   -Djava.security.policy=ORACLE_HOME/j2ee/home/config/java2.policy -jar oc4j.jar
```

(`-Xbootclasspath` puts `PrintingSecurityManager` into the boot classpath, where it runs with all the permissions.)

`PrintingSecurityManager` generates output that lists the following:

- Which code source requires which permissions
- A policy grant that you can copy and paste into the policy file

By default, these outputs go to `System.out`, but you can specify output files through the following system properties, the first for messages about missing permissions, and the second for policy grants:

```
-Doracle.oc4j.security.manager.printing.file=filenamepath
-Doracle.oc4j.security.manager.printing.generated.grants.file=filenamepath
```

Note: `PrintingSecurityManager` is not tied to OC4J, so you can use it outside of OC4J.

Enabling Subject Propagation for ORMI

Subject propagation is always used in OC4J with IIOP, in accordance with the CSIv2 specification. It is also used with ORMI if you specifically enable it on the client and server. You can accomplish this with the following system property setting at each end:

```
-Dsubject.propagation=true
```

In the current release, this setting controls subject propagation at a global OC4J level.

See Also:

- "Enabling and Configuring Subject Propagation for ORMI" on page 14-11 for complete information about subject propagation in OC4J

5

Tasks and Guidelines in Setting Security

This chapter discusses key tasks and related guidelines to consider when setting up security for your applications:

- Guidelines for Password Management
- Tasks and Guidelines for Using Security Realms in OC4J
- Tasks for JAAS Mode and Authorization
- Using the Java Authorization Contract for Containers
- Packaging Considerations for OC4J Configuration Files
- Deployment Tasks and Guidelines for Security
- Post-Deployment Considerations
- Tasks for DataSourceUserManager

See Also:

- The following Web site for OC4J "how-to" examples:

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

Guidelines for Password Management

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 indirection*, which replaces cleartext passwords with information necessary to look up the password in another location (`system-jazn-data.xml` in OC4J).
- *Password obfuscation*, which replaces passwords stored in cleartext files with an encrypted version of the password.

The rest of this discussion covers these topics:

- Creating an Indirect Password
- Specifying a Password Manager in system-application.xml
- Password Obfuscation in OC4J Configuration Files

Creating an Indirect Password

The following configuration files support password indirection in one or more elements:

- `data-sources.xml`: `password` attribute of `<data-source>` element
- `ra.xml`: `<res-password>` element
- `rmi.xml`: `keystore-password` attribute of `<ssl-config>` element
- `application.xml`: `password` attributes of `<resource-provider>` and `<commit-coordinator>` elements
- `jms.xml`: `<password>` element
- `*-web-site.xml`: `keystore-password` attribute of `<ssl-config>` element

To make any of these passwords indirect, replace the literal password string with a string containing "`->`" followed by either the user name or by the realm and user name separated by a slash ("`/`").

Here are some examples of indirect versus direct passwords.

- `<data-source password="->Scott">`

  Look up `Scott` in the default realm, and use the password stored in the password manager.

- `<res-password="->customers/Scott">`

  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 Password Manager in system-application.xml

The `<password-manager>` element in the OC4J-specific `system-application.xml` file (associated with the OC4J `system` application) specifies the security provider that is used to look up indirect passwords (discussed in the preceding section, "Creating an Indirect Password"). If this element is omitted, the security provider of the global application is used for authentication and authorization of indirect passwords. The `<jazn>` element within a `<password-manager>` element in `system-application.xml` can be different from the `<jazn>` element at the top level.

Note that for security reasons, credentials stored in Oracle Internet Directory cannot usually be retrieved in decrypted (cleartext) format, which means that Oracle Internet Directory cannot be used as a password manager for your application. To resolve this, you can specify the file-based provider as your application password manager, even when your application uses Oracle Identity Management as the security provider.

To do this, add an entry such as the following to the OC4J-specific `system-application.xml` file:

```
<password-manager>
  <jazn provider="XML"
    location=ORACLE_HOME/j2ee/instance_name/config/system-jazn-data.xml>
  </jazn>
</password-manager>
```

Note: By default, `system-jazn-data.xml` is used as the password manager.

Password Obfuscation in OC4J Configuration Files

The JAAS configuration files `jazn.xml` and `system-jazn-data.xml` (or optionally an application-specific `jazn-data.xml` file) contain user names and passwords for JAAS authorization. To protect these files, OC4J uses password obfuscation.

Generally, whenever you update `jazn.xml` or `system-jazn-data.xml`, OC4J reads the file, then rewrites it with obfuscated (encrypted) versions of all passwords.

In addition (relevant for Oracle Identity Management), a setting for the `ldap.password` property within a `<jazn>` element, such as in `orion-application.xml`, is obfuscated. For example:

```
<jazn ... >
   <property name="ldap.password" value="welcome123"/>
   ...
</jazn>
```

In other OC4J configuration, you can avoid exposing password cleartext by using password indirection, as explained in "Creating an Indirect Password" on page 5-2.

Note: In `system-jazn-data.xml` or an application-specific `jazn-data.xml` file, you can use clear (human-readable) passwords by setting the `clear` attribute of the `<credentials>` element to "`true`", though this is discouraged:

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

To explicitly specify password obfuscation, precede the password with "!" (in which case "!" is not considered part of the password):

```
<credentials>!welcome</credentials>
```

Tasks and Guidelines for Using Security Realms in OC4J

In OC4J, both the file-based provider and LDAP-based Oracle Identity Management use the concept of security realms, introduced in "Security Realms" on page 1-7. You can configure a single realm or multiple realms, and the default realm is specified through your OC4J configuration. Note that the concept of realms is not supported when you use external LDAP providers such as Active Directory or Sun Java System Directory Server.

This section discusses key details for using security realms to control authentication and authorization in OC4J, covering the following topics:

- Default Realm with the File-Based Provider or Oracle Identity Management
- Using the Default Realm
- Using a Nondefault Realm
- Using Multiple Realms
- Omitting the Realm Name When Retrieving an Authenticated Principal

Default Realm with the File-Based Provider or Oracle Identity Management

A default realm is specified in the `default-realm` attribute of the `<jazn>` element. For the file-based provider, this is either at application level in your `orion-application.xml` file, or at the OC4J level in the instance-level `jazn.xml` file. For Oracle Identity Management, this is in the bootstrap `jazn.xml` file.

For the file-based provider, `jazn.com` is configured as the default realm by default, at the instance level:

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

For Oracle Identity Management, the default realm is according to the Oracle Internet Directory, where it is determined during Oracle Internet Directory installation. After you associate OC4J with an Oracle Internet Directory instance, the default realm is reflected at the OC4J instance level, such as in the following example:

```
<jazn provider="LDAP" location="ldap://www.example.com:636" default-realm="us"/>
```

"Using the Default Realm" on page 5-5, discusses guidelines to be aware of when you use the default realm.

Important:

- A default realm should always be specified, even if you use only one realm. For the file-based provider, this means you should specify a default realm when you configure your security provider during application deployment.
- Do not remove configuration of the `jazn.com` realm from `system-jazn-data.xml`; it is there by default and must remain there for use by the OC4J `system` application.

Evaluation of the Default Realm for File-Based Provider or Oracle Identity Management

As noted in the preceding section, a default realm should always be configured. However, for reference purposes only, this section discusses the progression that is followed to determine the default realm if one is not specified, when using the file-based provider or Oracle Identity Management.

If your application uses the file-based provider:

1. OracleAS JAAS Provider looks for default realm configuration at the application level, in the `orion-appliation.xml` file. If a default realm is found there, it is used as the default realm for your application.
2. If there is no default realm setting at the application level, OracleAS JAAS Provider looks for default realm configuration at the OC4J instance level, in the `jazn.xml` file:
   - If `jazn.xml` sets `provider="XML"`, OracleAS JAAS Provider uses the default realm specified in `jazn.xml`, if one is specified, as the default realm for your application. If none is specified, an error is thrown to indicate that you are missing the default realm attribute.
   - If `jazn.xml` sets `provider="LDAP"`, OracleAS JAAS Provider uses `jazn.com` as the default realm for your application.

If your application uses Oracle Identity Management:

1. If configuration specifies the LDAP-based provider both for your application and at the OC4J instance level (in `jazn.xml`), then OracleAS JAAS Provider looks for default realm configuration in `jazn.xml`. If a default realm is found there, it is used as the default realm for your application.
2. If configuration does *not* specify the LDAP-based provider at the OC4J instance level, or if there is no default realm setting at the instance level, the Oracle Internet Directory default subscriber is used as the default realm. (This is configured in the Oracle Internet Directory server.)

Using the Default Realm

For authentication, when you use the default realm, there is no need to prefix the realm name with a user name. For example, if a user `scott` is in the default realm `jazn.com`, for authentication the user name need only be specified as "`scott`".

This is also true for applicable OC4J components and services such as JNDI, JMS, and J2CA.

Similarly, for password indirection, the OC4J deployment descriptor need not prefix the realm name in the user name specified for indirection: "`scott`".

Using a Nondefault Realm

If you are using a nondefault realm—such as `acme.com`, for this discussion—you must always prefix user names with the realm name. To authenticate the user `scott` in `acme.com`, for example, you would have to specify "`acme.com/scott`", not just "`scott`".

This is also the case for applicable OC4J components and services such as JNDI, JMS, and J2CA.

Similarly, for password indirection, the OC4J deployment descriptor must prefix the realm name in the user name specified for indirection, if the user is in a nondefault realm: "`acme.com/scott`".

Using Multiple Realms

When multiple realms are configured, you must prefix user names with the realm name for any nondefault realm that you use. For this discussion, assume the realms `jazn.com`, `acme.com`, and `example.org` are configured, with `jazn.com` being the default realm. Further assume user `scott` is in `jazn.com`, while user `ralph` is in `example.org`.

To specify `scott` for authentication, you need only specify the user as "`scott`", because he is in the default realm `jazn.com`.

To specify `ralph` for authentication, you must specify "`example.org/ralph`".

This is also the case for applicable OC4J components and services such as JNDI, JMS, and J2CA. The realm name must be specified for a user in any nondefault realm.

Similarly, for password indirection, the OC4J deployment descriptor must prefix the realm name in the user name specified for indirection if the user is in any nondefault realm: "`example.org/ralph`". But you need not specify the realm name for any user in the default realm, such as "`scott`".

Important: Always set `jaas.username.simple` to "`false`" when multiple realms are configured. (See the next section, "Omitting the Realm Name When Retrieving an Authenticated Principal".)

Omitting the Realm Name When Retrieving an Authenticated Principal

Unless you configure custom realms, it is typically desirable to omit the realm name from the authenticated principal that is returned by key methods for servlets, EJBs, and Web services. In OC4J, use the `jaas.username.simple` property to control this behavior. This property affects the following methods:

- `getUserPrincipal()` method of any `HTTPServletRequest` instance (servlets)
- `getRemoteUser()` method of any `HTTPServletRequest` instance (servlets)
- `getCallerPrincipal()` method of any `EJBContext` instance (EJBs)
- `getUserPrincipal()` method of any `ServletEndpointContext` instance (Web services)

With a "`true`" property setting, which is the default, principal names returned by these methods do not include the realm name: for example, "`scott`".

If you set the property to "`false`", then principal names returned by these methods are prefixed with the realm name: for example, "`jazn.com/scott`".

To specify a "`false`" setting, use a `<property>` subelement of the `<jazn>` element (in `orion-application.xml` for application level, or in the instance-level `jazn.xml` file for OC4J instance level) as follows:

```
<jazn ... >
   ...
   <property name="jaas.username.simple" value="false" />
   ...
</jazn>
```

Important: Always set `jaas.username.simple` to "`false`" when multiple realms are configured.

Tasks for JAAS Mode and Authorization

This section provides sample code for the following steps, to use J2EE and JAAS authorization in an application:

1. Use J2EE Authorization
2. Use OracleAS JAAS Provider Policy Management
3. Use OracleAS JAAS Provider JAAS Mode

The samples here use a servlet method, but the basic functionality is similar for EJBs.

See Also:

- "JAAS Authorization and JAAS Mode" on page 2-7 for an overview of features shown here
- Appendix B, "OracleAS JAAS Provider Samples" for the complete example

Use J2EE Authorization

This sample servlet `doGet()` method uses standard J2EE authorization methods to retrieve a user and principal, and determine whether a user is in the specified role.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
    ServletOutputStream out = response.getOutputStream();

    response.setContentType("text/html");
    out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
    out.println("Time stamp: " + new Date().toString());
    out.println("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
    out.println("request.isUserInRole('ar_developer') = " +
                request.isUserInRole("ar_developer") + "<br>");
    out.println("request.getUserPrincipal = " +
                request.getUserPrincipal() + "<br>");
    out.println("</BODY>");
    out.println("</HTML>");
}
```

Use OracleAS JAAS Provider Policy Management

In this example, the `doGet()` method shown in the preceding section, "Use J2EE Authorization", is expanded to use the OracleAS JAAS Provider policy management API to grant file permissions to a user.

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
       throws ServletException, IOException {
   ServletOutputStream out = response.getOutputStream();

   response.setContentType("text/html");
   out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
   out.println("Time stamp: " + new Date().toString());
   out.println("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
   out.println("request.isUserInRole('ar_developer') = " +
               request.isUserInRole("ar_developer") + "<br>");
   out.println("request.getUserPrincipal = " +
               request.getUserPrincipal() + "<br>");

//Grant Permissions to a user developer

//get JAZNConfiguration related info
   JAZNConfig jc = JAZNConfig.getJAZNConfig();

   //create a Grantee for "developer"
   RealmManager realmmgr = jc.getRealmManager();
   Realm realm = realmMgr.getRealm("jazn.com");
   UserManager userMgr = realm.getUserManager();
   final RealmUser user = userMgr.getUser("developer");

   //grant scott file permission
   JAZNPolicy policy = jc.getPolicy();
```

```
   if ( policy != null) {
      Grantee gtee = new Grantee( (Principal) user);
      java.io.FilePermission fileperm =
                           new java.io.FilePermission("foo.txt", "read");
      policy.grant( gtee, fileperm);
   }

   out.println("</BODY>");
   out.println("</HTML>");
}
```

See Also:

- *Oracle Identity Management Application Developer's Guide* for details about JAAS policy management APIs.

Use OracleAS JAAS Provider JAAS Mode

In this example, the `doGet()` method shown in "Use J2EE Authorization" on page 5-7 is expanded to create and check permissions. Furthermore, assume the JAAS mode `doAsPrivileged`, which is set with configuration such as the following in the application `orion-application.xml` file (which specifies the file-based provider in this example):

```
<orion-application ... >
   ...
   <jazn provider="XML" jaas-mode="doAsPrivileged" />
   ...
</orion-application>
```

The code follows. Because of the JAAS mode setting, the action method, in this case `doGet()`, will be executed by OC4J within a `Subject.doAsPrivileged()` block for the authenticated subject.

```
    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       ServletOutputStream out = response.getOutputStream();

       response.setContentType("text/html");
       out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
       out.println("Time stamp: " + new Date().toString());
       out.println
           ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
       out.println("request.isUserInRole('ar_developer') = " +
                   request.isUserInRole("ar_developer") + "<br>");
       out.println
           ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

      //create Permission
      FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
      {
         //get current AccessControlContext
         AccessControlContext acc = AccessController.getContext();

         javax.security.auth.Policy currPolicy =
                                   javax.security.auth.Policy.getPolicy();

         // Query policy now
         out.println("Policy permissions for this subject are " +
                     currPolicy.getPermissions(Subject.getSubject(acc),null));
```

```
        //Check Permissions
        out.println("Policy.impiles permission: "+ perm +" ? " +
        currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));
      }
      out.println("</BODY>");
      out.println("</HTML>");
  }
```

Using the Java Authorization Contract for Containers

This section describes what you must do to use the Oracle Java ACC provider in OC4J, covering the following topics:

- System Properties to Enable Java ACC Features
- System Properties to Specify the Java ACC Provider

Note: Java ACC is supported only for the file-based provider. Generated policies are stored in `system-jazn-data.xml`.

See Also:

- "Implementation of Java Authorization Contract for Containers" on page 2-12 for an overview

System Properties to Enable Java ACC Features

By default, Java ACC is disabled in OC4J. It can be enabled with the following system property setting at OC4J startup:

```
-Doracle.oc4j.security.jacc=true
```

Java ACC trace logging can be enabled with the following system property setting at OC4J startup:

```
-Doracle.oc4j.security.jacc.debug=true
```

System Properties to Specify the Java ACC Provider

To employ a Java ACC provider, the system properties described in Table 5–1 must be set appropriately at application server startup. For the Oracle Java ACC provider, this happens automatically when you enable Java ACC, with the properties being set as shown in parentheses.

Table 5–1 System Properties for the Java ACC Provider

Property	Description
javax.security.jacc.policy.provider	Class name of the policy provider (oracle.security.jacc.provider.J2SEPolicy)
javax.security.jacc.policy. PolicyConfigurationFactory.provider	Class name of the policy mapping configuration factory (oracle.security.jacc.provider. JACCPolicyConfigurationFactory)
oracle.security.jacc.provider. RoleMappingConfigurationFactory.provider	Class name of the role mapping configuration factory (oracle.security.jacc.provider. JACCRoleMappingConfigurationFactoryImpl)

Packaging Considerations for OC4J Configuration Files

This section discusses packaging considerations for OC4J-specific configuration files:

- Configuration Tasks and Considerations in the Deployment Descriptors
- Supplying an Application-Specific jazn-data.xml File

Configuration Tasks and Considerations in the Deployment Descriptors

When you package your application, you will include standard deployment descriptors such as `application.xml`, `web.xml`, and `ejb-jar.xml`. You can also optionally provide OC4J descriptors such as `orion-application.xml`, `orion-web.xml`, and `orion-ejb-jar.xml`, or they may be provided for you, such as according to your settings in Application Server Control Console when you deploy.

This section summarizes some considerations for your deployment descriptors before you package your application.

Configuration to Use the Instance-Level File-Based Provider

If there is no `<jazn>` element in the `orion-application.xml` file when you deploy your application (neither placed there automatically by Application Server Control or other deployment processing, nor placed there manually), the default is as follows:

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

This results in use of the instance-level file-based provider with `system-jazn-data.xml` as the repository.

Alternatively, you can explicitly place this configuration in `orion-application.xml` in order to use the instance-level file-based provider.

Configuration to Automatically Create jazn-data.xml

If `orion-application.xml` is configured exactly as follows:

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

But the `jazn-data.xml` file is not packaged with the application, then one will be created during deployment.

Supplying an Application-Specific jazn-data.xml File

If you supply a `jazn-data.xml` file with your application, then you must specify its location through the `<jazn>` element `location` attribute in the `orion-application.xml` file for your application. For example:

1. In `orion-application.xml`, specify the following:

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

2. Package the `jazn-data.xml` file in the `/META-INF` directory of the EAR file.

Deployment Tasks and Guidelines for Security

This section discusses security issues to consider when deploying your application, covering the following topics:

- Overview of Deployment Considerations
- Deploying an Application

- Specifying a Security Provider
- Mapping J2EE Security Roles to JAAS Roles

See Also:

- *Oracle Containers for J2EE Deployment Guide* for a full discussion of deployment and related considerations

Overview of Deployment Considerations

The security 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 security provider also supports programmatic security in the same manner that JAAS is used in any J2SE environment.

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

- Decide which security provider you want to use. The Oracle Application Server includes Oracle Identity Management, which uses the LDAP-based Oracle Internet Directory as the repository, and the file-based provider, which uses an XML file as the repository. OC4J also supports external (third-party) LDAP providers, custom security providers (custom login modules), and, beginning in the OC4J 10.1.3 implementation, the COREid Access provider.
- 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.
- Map the security roles (including the application-specific roles, if they exist) to users and roles defined by the OracleAS JAAS Provider. For example, the J2EE logical role called `hr_manager` may be mapped to a given set of users defined by the OracleAS JAAS Provider.
- Consider whether you have any code that you will want to load as shared libraries (login modules, for example).

Deploying an Application

This section discusses how to deploy an application, focusing on the functionality of the Application Server Control Console.

Deploying an Application through Application Server Control

Details about deploying an application to OC4J, including information about deployment plans, are provided in the *Oracle Containers for J2EE Deployment Guide*. This section reviews the basic steps:

1. In the OC4J Home page, select the **Applications** tab.
2. In the resulting Applications page, choose **Deploy**.

3. In the resulting Deploy: Select Archive page (page 1 of 3), specify the archive file to deploy and your desired choice for a deployment plan.
4. In the Deploy: Application Attributes page (page 2 of 3), specify the desired application name, parent application, Web site, and context root.
5. In the Deploy: Deployment Settings page (page 3 of 3), you can choose any of the following tasks:
   - Map Environment References (if applicable)
   - Select Security Provider
   - Map Security Roles (if applicable)
   - Configure EJBs (if applicable)
   - Configure Clustering
   - Configure Class Loading

   For security, selecting a security provider and mapping security roles are of particular interest. You may also want to configure class loading in order to import shared libraries, such as if you have login modules that you want to load as shared libraries.
6. In the Deploy: Deployment Settings page, when you are done with any applicable tasks mentioned in the previous step, select **Deploy**.

See Also:

- "Specifying the Security Provider through Application Server Control" on page 5-13
- "Specifying Security Role Mapping through Application Server Control" on page 5-14
- "Using Login Modules as OC4J Shared Libraries" on page 8-6

Specifying a Security Provider

This section discusses how to specify the security provider using Application Server Control Console, as well as considerations for using the file-based provider versus the LDAP-based provider.

Considering the File-Based Provider Versus Oracle Identity Management

Generally, use the file-based provider in development environments and in deployed applications with a small user population. Use Oracle Identity Management in larger production environments.

Compared to the file-based provider, Oracle Identity Management offers better security and performance. The centralized Oracle Internet Directory server scales well as the number of applications and users grows, and enables you to configure cache parameters to improve overall performance of authentication and authorization.

In addition, Oracle Internet Directory offers features such as centralized account creation and management, single passwords, and credential management.

Note: To use Oracle Identity Management, the OC4J instance must have been previously associated with Oracle Internet Directory through Application Server Control.

Specifying the Security Provider through Application Server Control

In Application Server Control Console, specify the security provider during deployment, from the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page), as follows:

1. Choose the Select Security Provider task.
2. In the resulting Deployment Settings: Select Security Provider page, choose the desired security provider from the dropdown list. The choices are:
   - File-Based
   - Oracle Identity Management
   - Third Party LDAP Server (for an external LDAP provider)
   - Custom (for a custom login module)
3. Each type of security provider involves its own set of configuration tasks, documented in the following locations:
   - "Specifying Oracle Identity Management during Deployment" on page 6-14
   - "Configuring the File-Based Provider during Application Deployment" on page 7-3
   - "Specifying and Configuring an External LDAP Provider during Deployment" on page 9-3
   - "Specifying and Configuring a Custom Security Provider during Deployment" on page 8-7
4. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11.

Mapping J2EE Security Roles to JAAS Roles

This section discusses various aspects of the following:

- Defining security roles in an application and referencing them to logical J2EE security roles in the standard deployment descriptor
- Mapping J2EE security roles to JAAS security roles in the OC4J configuration, and how to accomplish this mapping in Application Server Control

The information is organized as follows:

> **Note:** Security role mappings are *not* inherited from a parent application.

See Also:

- "Overview of Security Role Mapping" on page 2-13
- "Web Application Security Role Configuration" on page 13-6
- "Mapping Logical Roles to Users and Roles" on page 14-7 (for EJBs)

Application Role Definitions and References

The process of role definitions and references includes the following steps:

1. During development, define roles as classes that implement the `java.security.Principal` interface.
2. In your standard deployment descriptor (`web.xml` or `ejb-jar.xml`), use `<security-role>` elements to define logical security roles, such as in the following example:

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

3. Use `<security-role-ref>` elements in the standard deployment descriptor to map from the roles you have developed in your application to the logical roles you have defined in the descriptors, such as in the following example (where `ar_developer` is defined in the application):

```
<security-role-ref>
   <role-name>ar_developer</role-name>
   <role-link>sr_developer</role-link>
</security-role-ref>
```

After these steps, mappings from the logical roles to roles defined in the OC4J container are defined in the OC4J descriptors (`orion-web.xml`, `orion-ejb-jar.xml`, or `orion-application.xml`). These files are updated, as appropriate, through the mappings you define when you deploy an application through Application Server Control, for example, and are reflected in `<security-role-mapping>` elements. These mappings are discussed in the next two sections, "Specifying Security Role Mapping through Application Server Control" and "Mapping J2EE Roles to JAAS Roles in OC4J Configuration Files".

See Also: The preceding discussion leaves out some details, which differ between Web applications and EJBs. Refer to the following for additional information:

- "Web Application Security Role Configuration" on page 13-6
- "Authenticating and Authorizing EJB Applications" on page 14-2

Specifying Security Role Mapping through Application Server Control

In Application Server Control Console, map security roles during deployment, from the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page), as follows:

1. Select the Map Security Roles task.
2. In the resulting Deployment Settings: Map Security Roles page, choose the Map Role task for each role you want to map. (You can also choose **Clear All Mappings**.)

3. In the resulting page for the role, you can do any of the following:
   - Map all users and groups to the role.
   - Map selected users to the role. Choose **Add Existing User**, then specify the desired users in the Select and Search: Users page, then choose **Select**. If **Add Existing User** does not list the desired user, then use the Add User feature.
   - Map selected groups to the role. Choose **Add Existing Group**, then specify the desired groups in the Select and Search: Groups page, then choose **Select**. If **Add Existing Group** does not list the desired group, then use the Add Group feature.
   - Choose **Continue** when you are finished mapping users and groups.
4. Back in the Deployment Settings: Map Security Roles page, choose **OK**.
5. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11

These actions create `<security-role-mapping>` elements in the applicable OC4J configuration file, such as `orion-application.xml`, `orion-web.xml`, `orion-ejb-jar.xml`, as shown in the next section, "Mapping J2EE Roles to JAAS Roles in OC4J Configuration Files".

> **Note:** There is no way to alter security mappings through Application Server Control after deployment. You would have to update the configuration manually (as shown in "OC4J Mapping of J2EE Roles to JAAS Roles" on page 13-7 and "Mapping Logical Roles to Users and Roles" on page 14-7) and then restart or redeploy the application.

Mapping J2EE Roles to JAAS Roles in OC4J Configuration Files

Portable J2EE security roles defined in a standard J2EE deployment descriptor are mapped to JAAS roles in OC4J through `<security-role-mapping>` settings in the `orion-application.xml` file (to apply throughout a J2EE application), `orion-web.xml` file (to apply to a particular Web application), or `orion-ejb-jar.xml` file (to apply to a particular EJB application).

In this example, the JAAS `sr_developer` security role is mapped to the OC4J `developer` role. Note that a `<group>` subelement under a `<security-role-mapping>` element corresponds to a role in the OracleAS JAAS Provider. You can also have `<user>` subelements to map to individual users.

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

This association permits the `developer` role to access resources that are accessible for the `sr_developer` role.

Consider a user `john`, for example, who is a member of the `developer` role. Because this role is mapped to the J2EE role `sr_developer`, `john` has access to the application resources available to the `sr_developer` role.

Using the OC4J PUBLIC Role to Allow General Access by Authenticated Users

For situations where you care only about authentication, not authorization, OC4J supports a mode where any authenticated user is allowed access to a given application or resource. This is supported through the `PUBLIC` role, and can be configured down to a per-URL or per-method basis as desired. This involves the following steps:

1. If you do not already have a logical role intended for public access, you can define such a role in `web.xml` (for a Web application) or in `ejb-jar.xml` (for an EJB).

   For example, in `web.xml`:

   ```
   <web-app>
      ...
      <security-role>
         <role-name>public_role</role-name>
      </security-role>
      ...
      <auth-constraint>
         <role-name>public_role</role-name>
      </auth-constraint>
      ...
   </web-app>
   ```

   Or, in `ejb-jar.xml`:

   ```
   <assembly-descriptor>
      ...
      <security-role>
         <role-name>public_role</role-name>
      </security-role>
      ...
      <method-permission>
         <role-name>public_role</role-name>
         <method>method</method>
      </method-permission>
      ...
   </assembly-descriptor>
   ```

2. Map your public role to the `PUBLIC` role in `orion-application.xml` (for a Web application) or `orion-ejb-jar.xml` (for an EJB).

   To map the role defined in `web.xml` above, include the following in `orion-application.xml`:

   ```
   <orion-application>
      ...
      <security-role-mapping name="public_role">
         <group name="{{PUBLIC}}"/>
      </security-role-mapping>
      ...
   </orion-application>
   ```

   Or, for an EJB, use the `<security-role-mapping>` element in `orion-ejb-jar.xml` instead (where it is a subelement of the `<assembly-descriptor>` element).

   Note: This example assumes the default setting of `"{{PUBLIC}}"` for the OC4J public group. This may be overridden through the OracleAS JAAS Provider `public.group` property.

Post-Deployment Considerations

This section discusses the following consideration for after you have deployed your application:

- Navigating to the Security Provider Page for Your Application

Navigating to the Security Provider Page for Your Application

After you have deployed your application, you can go to the Security Provider page for your application in the Application Server Control Console to examine or update the application-level security settings. Starting from the OC4J Home page for your OC4J instance:

1. Choose the **Administration** tab.
2. In the Administration page, go to the Security Providers task (under "Security").
3. In the Security Providers page, under "Application Level Security", go to the Edit task for your application.

This brings you to the Security Provider page, displaying information on the provider for your application and allowing you to update settings or change to a different security provider.

Tasks for DataSourceUserManager

As an alternative to using security providers documented elsewhere in this manual, the OC4J 10.1.3 implementation continues to support the `DataSourceUserManager` class, in package `com.evermind.sql`, for retrieving user data from a database. This section discusses features of this class, and how to configure your application to use it. The following topics are covered:

- DataSourceUserManager Properties
- Configuring an Application to Use DataSourceUserManager

Important:

- The `DataSourceUserManager` class is deprecated in the OC4J 10.1.3 implementation, but is still supported for backward compatibility. In future releases, it will be replaced with equivalent functionality using a custom login module.
- In the 10.1.3 implementation, `DataSourceUserManager` obtains group information only from the database, which differs from the behavior in previous implementations. Therefore, you must now map groups to users in the database, as applicable.

DataSourceUserManager Properties

When you configure `DataSourceUserManager` (as described later, in "Configuring an Application to Use DataSourceUserManager" on page 5-18), you can specify values for the properties described in Table 5–2, as appropriate. The `DataSourceUserManager` instance uses these properties to access the user-defined database table that lists the current users and their associated credentials.

Table 5–2 DataSourceUserManager Properties

Property	Description
dataSource	A JNDI location for the installed data source (database) to use
table	Name of the database table containing user data
usernameField	Name of the column for user names in the database table
passwordField	Name of the column for passwords in the database table
certificateIssuerField	An identifier for the certificate issuer, if applicable
certificateSerialField	The serial ID of the certificate issuer, if applicable
localeField	The locale, if applicable
defaultGroups	Comma-delimited list of groups that the users are members of
groupMembershipTableName	Name of an optional database table that maps users to groups, if the use of `defaultGroups` is not sufficient
groupMembershipUserNameFieldName	Name of the column for user names in the group membership database table, if applicable
groupMembershipGroupFieldName	Name of the column for group names in the group membership database table, if applicable
staleness	Number of milliseconds for which a fetched set of user data will be valid. The default setting is -1 (forever)
casing	Flag that controls how `DataSourceUserManager` handles character case for user names (but not group names) when trying to match a name against the list of known users in the database The default "`sensitive`" setting results in case-sensitive matching. For the "`toupper`" and "`tolower`" settings, the name is converted to all uppercase or all lowercase, respectively, for purposes of matching.
debug	Flag to enable output of debug information

Configuring an Application to Use DataSourceUserManager

To use `DataSourceUserManager`, configure it in a `<user-manager>` element in your `orion-application.xml` file. This is a subelement of `<orion-application>`, and must be configured manually. There is no `UserManager` support in the Application Server Control 10.1.3 implementation.

Specify the `DataSourceUserManager` fully qualified name in the `class` attribute of `<user-manager>`. Use a `<property>` subelement to specify the name and value of each property you want to set.

See Also:

- The preceding section, "DataSourceUserManager Properties"

Here is an example:

```
<orion-application ... >
   ...
```

```
   <user-manager class="com.evermind.sql.DataSourceUserManager">
      <property name="dataSource" value="jdbc/OracleCoreDS" />
      <property name="table" value="j2ee_users" />
      <property name="usernameField" value="username" />
      <property name="passwordField" value="password" />
      <property name="groupMembershipTableName" value="second_table" />
      <property name="groupMembershipGroupFieldName" value="group" />
      <property name="groupMembershipUserNameFieldName" value="userId" />
   </user-manager>
   ...
</orion-application>
```

6

Oracle Identity Management Security Provider

In Oracle Application Server, Oracle Identity Management with Oracle Internet Directory and (optionally) OracleAS Single Sign-On is the LDAP-based security provider.

This chapter is for those who use or plan to use Oracle Identity Management as the security provider, and covers the integration of Oracle Identity Management with OC4J. It begins with some overview of LDAP realm management, then discusses configuration and use of these features, covering the following topics:

Notes:

- Beginning with the OC4J 10.1.3 implementation, the LDAP-based provider is supported in standalone OC4J as well as in an Oracle Application Server environment.
- Be aware that with the LDAP-based provider, role comparisons for authorization are *not* case-sensitive.
- Managing users and roles in Oracle Internet Directory is beyond the scope of this document. Consult the *Oracle Identity Management Guide to Delegated Administration*.
- After you add or modify a user account in Oracle Internet Directory, you should be able to log in without restarting OC4J, assuming you have associated Oracle Internet Directory with OC4J as described in "Associate Oracle Internet Directory with OC4J" on page 6-9.
- OC4J provides a login module, `LDAPLoginModule`, for use with non-Oracle LDAP servers. Do not configure this login module for use with Oracle Internet Directory. Doing so would result in the loss of optimizations and integrations that are otherwise available. The only login module for use with Oracle Internet Directory is `RealmLoginModule`.

Realm Management in LDAP-Based Environments

A realm is a collection of users and roles. In the OC4J 10.1.3 implementation, manage users and roles in an LDAP-based (Oracle Internet Directory) realm by using administrative features of the Oracle Delegated Administration Services (DAS).

This section discusses the following topics for realm management in Oracle Internet Directory:

- LDAP-Based Realm Types
- LDAP-Based Realm Data Storage

LDAP-Based Realm Types

The OracleAS JAAS Provider supports the identity management realm for LDAP-based environments. (The external realm and application realm are deprecated.) A realm provides different user and role management capabilities.

A realm type consists of:

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

The identity management realm:

- Is created through provisioning tools.
- Is used in hosting environments, and is well suited for a hosting environment in which multiple customers or companies subscribe to shared services.
- Supports external, read-only user and role management.

When you use Oracle Internet Directory with OracleAS Single Sign-On, you must use the identity management realm.

Notes:

- Specifically, regarding external and application realms: in package `oracle.security.jazn.realm`, `APPLICATION_REALM` and `EXTERNAL_REALM` are deprecated in the `RealmType` class; `_extRealm` and `_appRealm` are deprecated in the `InitRealmInfo` class. External realms and application realms will be desupported in future releases.
- Use the DAS tool for user and role management with Oracle Internet Directory.

Figure 6–1 shows a sample LDAP DIT containing an identity management realm that is registered as an instance with the OracleAS JAAS Provider. The realm type is created below a realms container.

Figure 6–1 Simplified Directory Information Tree for the Identity Management Realm

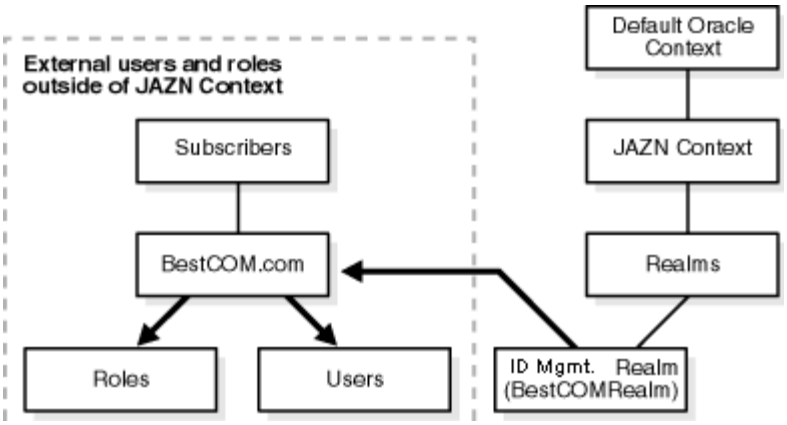


Table 6–1 describes the user and role management responsibilities of the identity management realm.

Table 6–1 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

About Distinguished Names

The term *distinguished name*, or DN, is used frequently in this chapter. This is a standard LDAP concept. A DN comprises a set of one or more relative distinguished names (RDNs) separated by commas. An RDN can be any of the following:

- DC (domain component)
- CN (common name)
- OU (organizational unit name)
- O (organization name)
- STREET (street address)
- L (locality name)
- ST (state or province)
- C (country)
- UID (user ID)

RDNs most often consist of common names or domain components in the discussion in this chapter. A common name could be something like "Jeff Smith" or "Oracle", for example.

See Also:

- The next section, "LDAP-Based Realm Data Storage"

LDAP-Based Realm Data Storage

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

By default, Oracle Internet Directory has one default identity management realm. OracleAS JAAS Provider creates a corresponding realm that it linked to it. For example (a typical scenario), an OracleAS JAAS Provider realm called "`us`" is linked to the default identity management realm in Oracle Internet Directory, which has the distinguished name "`dc=us,dc=oracle,dc=com`". Each time you create a new identity management realm in Oracle Internet Directory, a corresponding OracleAS JAAS Provider realm is created to link to it.

A realms container object is created under the site-wide JAAS context. For each registered realm instance, a corresponding realm entry is created under the realms container that stores the realm attributes. This directory hierarchy is known to the OracleAS JAAS Provider, which enables it to create new realm instances in the desirable directory location and find all the registered realms in runtime.

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

During runtime, the OracleAS JAAS Provider finds all the registered realms and their attributes (name, user manager implementation class, role manager implementation class, and their properties) from Oracle Internet Directory and instantiates the realm implementation class with the properties for initialization.

Realm Hierarchy

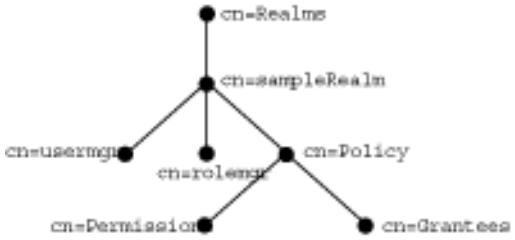
As Figure 6–2 illustrates, the OracleAS JAAS Provider stores its entries within the product container `cn=JAZNContext`. Beneath `cn=JAZNContext` is a `cn=Realms` container, which stores realm entries, and a `cn=Policy` container, which stores global OracleAS JAAS Provider policies. The `cn=Policy` container in turn stores two types of entries, `cn=Permissions` and `cn=Grantees`.

Note that the OracleAS JAAS Provider has its own `Groups` and `Users` containers. The `Groups` container contains the role `JAZNAdminGroup`. The `Users` container contains the users that populate this role.

Figure 6–2 Global JAZNContext Subtree

Figure 6–3 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 management. The policy-related entries under `cn=sampleRealm` store realm-specific policies.

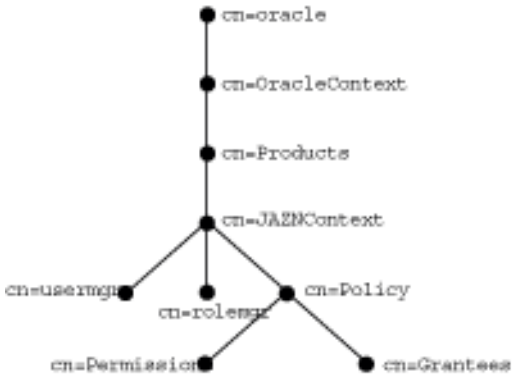
Figure 6–3 Realm-Specific Subtree



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

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

Figure 6–4 Subscriber JAZNContext Subtree



Access Control Lists and OracleAS JAAS Provider Directory Entries

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

Overview of Oracle Identity Management Key Components

Oracle Identity Management provides an enterprise infrastructure for securing distributed enterprise applications. It is an integrated package that includes the LDAP-based Oracle Internet Directory, Oracle Application Server Single Sign-On, and additional security and user management functionality.

To use Oracle Identity Management as your security provider, you must consider the underlying Oracle Internet Directory and OracleAS Single Sign-On. This section provides an overview of these features:

- Overview of Oracle Internet Directory

- Overview of Oracle Application Server Single Sign-On
- SSO-Enabled J2EE Environment: Typical Scenario

See Also:

- *Oracle Identity Management Administrator's Guide*

Overview of Oracle Internet Directory

Oracle Internet Directory provides Windows integration, password policy options, partial replication, and other important security features, including the following.

- Windows integration capabilities: Provides a preconfigured directory synchronization solution for Windows Active Directory Services. This feature allows users to have a single identity and password credential across Oracle and Windows environments. It also includes directory plug-ins that support mastering and changing passwords stored in the Windows environment, relieving customers of overhead and potential security concerns associated with synchronizing passwords across the two environments.
- Flexible password policy: Supports password policy options. In addition, Oracle Internet Directory plug-in support allows customers to implement an almost unlimited variety of site-specific password policies.
- Partial replication: Supports replication models, enabling improved scalability and performance in large network configurations.
- Other features include support for dynamic groups, an expanded Oracle Internet Directory Self-Service Console, easy synchronization of directory data with database tables, and features to permit user identity synchronization with the Oracle e-Business Suite Release 11*i*.

When using Oracle Internet Directory with the OC4J 10.1.3 implementation, the basic, digest, client-cert, username token, X.509 token, and SAML token authentication methods are supported.

See Also:

- *Oracle Internet Directory Administrator's Guide*

Overview of Oracle Application Server Single Sign-On

OracleAS Single Sign-On supports multilevel authentication. This allows customers to establish more than one authentication mechanism, and indicates the way in which a user is authenticated to single sign-on enabled applications. Applications can take advantage of this to grant different degrees of privilege to users, depending on how they authenticated.

For example, users may get partial privileges if they authenticate using a password, but more complete privileges if they use stronger authentication, such as X.509v3.

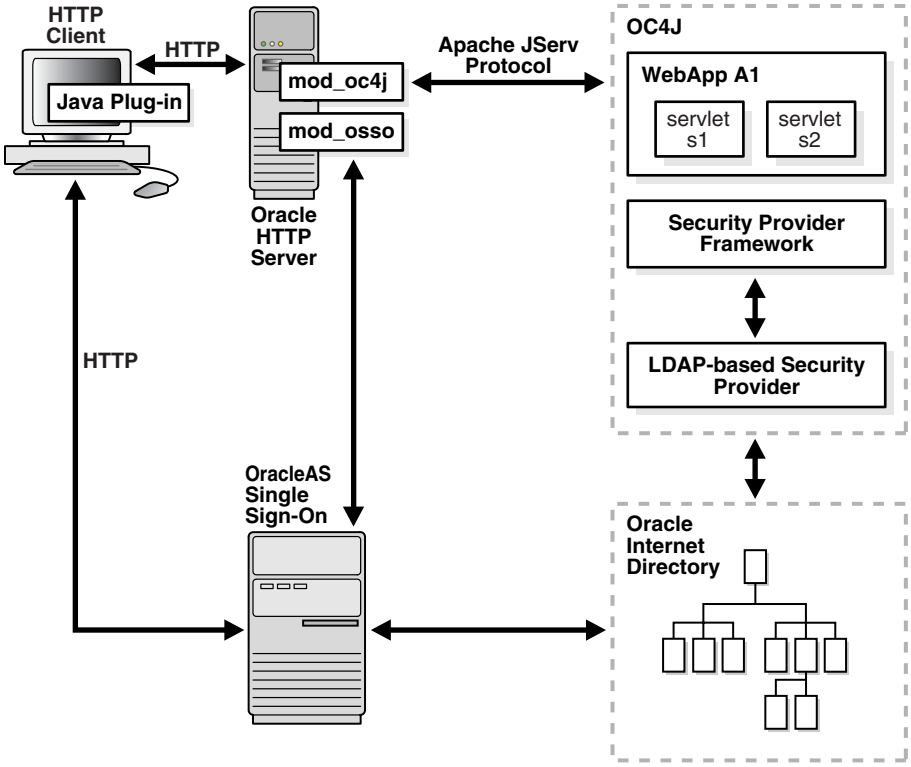
See Also:

- *Oracle Application Server Single Sign-On Administrator's Guide*

SSO-Enabled J2EE Environment: Typical Scenario

OracleAS Single Sign-On lets a user access multiple applications with a single set of login credentials. Figure 6–5 shows JAAS integration in an application running in an SSO-enabled J2EE environment.

Figure 6–5 OracleAS Single Sign-On and J2EE Environments



The following steps describe the responsibilities of Oracle components when an HTTP client request is initiated in a J2EE environment with OracleAS Single Sign-On enabled.

1. An HTTP client attempts to access a Web application, WebApp A1, hosted by OC4J (the Web container for executing servlets). Oracle HTTP Server (using an Apache listener) handles the request.
2. Oracle HTTP Server/`mod_osso` receives the request and:
   - Determines that WebApp A1 application requires Web-based OracleAS Single Sign-On for authenticating HTTP clients.
   - Redirects the HTTP client request to the Web-based OracleAS Single Sign-On (because it has not yet been authenticated).
3. The HTTP client is authenticated by OracleAS Single Sign-On through a user name and password or through a user certificate. OracleAS Single Sign-On then:
   - Validates the user's stored login credentials.
   - Sets the OracleAS Single Sign-On cookie (including the user's distinguished name and realm).
   - Redirects back to the WebApp A1 application (in OC4J).
4. The security provider retrieves the OracleAS Single Sign-On user.

> **Note:** For full details on OracleAS Single Sign-On, see the *Oracle Application Server Single Sign-On Administrator's Guide*.

Prerequisites: Oracle Application Server Infrastructure

Oracle Identity Management is part of the Oracle Application Server infrastructure. Using Oracle Identity Management as security provider requires the 10.1.2 or 9.0.4 infrastructure to be installed. This is in a separate *ORACLE_HOME* from OC4J.

For information about installing Oracle Application Server infrastructure, refer to the appropriate *Oracle Application Server Installation Guide* for your platform.

The rest of this section provides additional information, organized as follows:

Supported Versions for Oracle Internet Directory and OracleAS Single Sign-On

For using Oracle Identity Management (with Oracle Internet Directory) as the security provider under OracleAS JAAS Provider in the OC4J 10.1.3 implementation, the supported versions of the Oracle Application Server infrastructure are 10.1.2.0.2, 10.1.2.0.1, and 9.0.4.x.

Using OracleAS Single Sign-On as well, however, requires version 10.1.2.0.1 or 9.0.4.3 of the infrastructure. You must upgrade to the appropriate version if you want to use SSO and are currently using 10.1.2.0.0, or 9.0.4.2 or prior.

Considerations for 9.0.4.x Infrastructure: Access Control List Settings

Prior to the Oracle Internet Directory 10.1.2 implementation, access control list (ACL) features were not set up properly for `JAZNAdminGroup`. To use the Oracle Internet Directory 9.0.4 implementation with a 10.1.x OracleAS JAAS Provider implementation, place the following contents into a file, replacing `%s_MgmtRealmDN%` with the appropriate ID management realm (for example, `dc=us,dc=oracle,dc=com`), then execute the steps that follow.

```
dn: cn=JAZNContext,cn=Products,cn=OracleContext,%s_MgmtRealmDN%
changetype: modify
replace: orclaci
orclaci: access to entry
   by group= "cn=JAZNAdminGroup,cn=Groups,cn=JAZNContext,cn=Products,cn=OracleContext"
(browse, add, delete)
   by group= "cn=IASAdmins,cn=Groups,cn=OracleContext,%s_MgmtRealmDN%
added_object_constraint=(objectclass=orclApplicationEntity) (add, delete, browse)
   by * (none)
orclaci: access to attr=(*)
   by group= "cn=JAZNAdminGroup,cn=Groups,cn=JAZNContext,cn=Products,cn=OracleContext"
(search, read, write, compare)
   by group= "cn=IASAdmins,cn=Groups,cn=OracleContext,%s_MgmtRealmDN%"
(read, search, write, compare)
   by * (none)
```

1. Name the file with the `.ldif` extension, such as `jaznacl.ldif`.
2. Run the `ldapmodify` utility with the newly created file as input, specifying *oidport*, *oidhost*, *adminuser_dn*, *password*, and *filename*, as appropriate:

```
% ldapmodify -c -a -p oidport -h oidhost -D adminuser_dn -w password \
             -f filename.ldif
```

Note: The `ldapmodify` tool is a standard LDAP utility and is provided with Oracle Internet Directory in *`ORACLE_HOME`*`/bin` in your Oracle Application Server infrastructure installation.

See Also:

- "Access Control Lists and OracleAS JAAS Provider Directory Entries" on page 6-5

Steps to Use the Oracle Identity Management Security Provider

This section documents the steps involved in setting up Oracle Identity Management as your security provider, optionally using OracleAS Single Sign-On for authentication:

1. Associate Oracle Internet Directory with OC4J
2. Configure Oracle Identity Management as the Security Provider
3. Configure SSO (Optional)

See Also: For additional information about installing and using Oracle Application Server infrastructure:

- *Oracle Application Server Installation Guide* (appropriate version for your platform)
- *Oracle Application Server Administrator's Guide*

Associate Oracle Internet Directory with OC4J

This section discusses the step of associating an Oracle Internet Directory instance with your OC4J instance, which you must do before you can specify Oracle Identity Management as the security provider. It also shows the corresponding XML configuration. The following subtopics are covered:

- Associating Oracle Internet Directory with OC4J
- Changing the Oracle Internet Directory Association
- Required OC4J Accounts Created in Oracle Internet Directory
- Oracle Internet Directory Association in jazn.xml
- Associating the OC4J System Application with Oracle Internet Directory

Associating Oracle Internet Directory with OC4J

Use the Application Server Control Console to associate your OC4J instance with an instance of the LDAP-based Oracle Internet Directory, the repository for Oracle Identity Management. Here are the steps:

1. In the OC4J Home page for your instance, choose the **Administration** tab.
2. In the resulting Administration page, choose the Identity Management task (one of the Security tasks).
3. In the resulting Identity Management page, choose **Configure**. (This assumes no Oracle Internet Directory instance was previously associated with the OC4J instance, so that the Oracle Internet Directory host name and port are listed as "not configured". If a different Oracle Internet Directory instance was previously

associated with this OC4J instance, see the next section, "Changing the Oracle Internet Directory Association".)

4. In the resulting Configure Identity Management: Connection Information page, do the following:
   - Specify the fully qualified host name for the Oracle Internet Directory instance (`myoid.oracle.com`, for example).
   - Specify the distinguished name for the Oracle Internet Directory user, such as `cn=orcladmin` (see note below). The user specified here must belong to the `iASAdmins` role in the Oracle Internet Directory instance.
   - Specify the password for the Oracle Internet Directory user. This will also be set as the default password for the `oc4jadmin` user created in Oracle Internet Directory (unless the `oc4jadmin` account had previously been created, due to associating a different OC4J instance with the Oracle Internet Directory instance).
   - Specify whether to use SSL connections or non-SSL connections to the Oracle Internet Directory instance, and the appropriate port to use. The port for SSL is typically 636; for non-SSL it is typically 389. (To change the SSL or port setting later, you would have to redo the OC4J-OID association, as described in the next section, "Changing the Oracle Internet Directory Association".)
   - When you are done, go to the next page.
5. In the Configure Identity Management: Application Server Control page, you can specify whether Application Server Control uses Oracle Identity Management as its security provider. (If you do this, only users and roles defined in the Oracle Internet Directory instance will be able to access Application Server Control.)

   When you are done, go to the next page.
6. In the Configure Identity Management: Deployed Applications page, you can optionally specify Oracle Internet Directory, with or without SSO, as the security provider for each deployed application in the OC4J instance.

   When you are done, choose **Configure**. This completes the OC4J-OID association process and takes you back to the Identity Management page.

Notes:

- Because Oracle Internet Directory is associated at OC4J instance level, OracleAS JAAS Provider picks up the Oracle Internet Directory host, port, password, and SSL settings only from the `jazn.xml` file of a given OC4J instance, not from any application-level configuration.
- Each user in a directory must have a unique distinguished name.

Changing the Oracle Internet Directory Association

This section describes the steps to change the OC4J-OID association to use a different Oracle Internet Directory instance, or to change the port or SSL configuration. A new `JAZNAdminUser` account is created in Oracle Internet Directory.

1. As in the previous section, "Associating Oracle Internet Directory with OC4J", navigate to the Identity Management page.
2. In the Identity Management page, choose **Change**. (This is in the same place as **Configure** would be if there had been no previous OC4J-OID association.)

3. In the Change Identity Management page, as for the Configure Identity Management page in the previous section, specify the Oracle Internet Directory host name, the distinguished name and password of the Oracle Internet Directory user, whether to use SSL connections, and the port number for connections.
4. Choose **OK**. This completes the OC4J-OID reassociation process and brings you back to the Identity Management page.

Required OC4J Accounts Created in Oracle Internet Directory

The Oracle Internet Directory 10.1.2 implementation does not by default include certain accounts that are required by OC4J and Application Server Control 10.1.3 implementations. Therefore, the accounts listed below are created automatically as default accounts in Oracle Internet Directory, under the default identity management realm, as part of the OC4J-OID association process. This occurs the first time an OC4J instance is associated with the Oracle Internet Directory instance. On any subsequent associations of the same or any other OC4J instance with the same Oracle Internet Directory instance, these accounts are not changed.

- `oc4jadmin` user
- `oc4j-administrators` role
- `oc4j-app-administrators` role

(Also during OC4J-OID association, the `ascontrol_admin`, `ascontrol_appadmin`, and `ascontrol_monitor` roles are created for Application Server Control.)

Note: The file `oidConfigForOc4j.sbs` in directory *ORACLE_HOME*`/j2ee/home/jazn/install` contains the OC4J accounts and permissions for default users and roles that are created in Oracle Internet Directory the first time an OC4J instance is associated with that Oracle Internet Directory instance. Do not modify or delete this file, as these accounts are required for normal OC4J operations. Also, do not modify or delete any of these default accounts or their permissions once they are created.

See Also:

- "Predefined OC4J Accounts" on page 3-11 for additional information about the OC4J accounts
- "Activation of the oc4jadmin Account" on page 3-12

Oracle Internet Directory Association in jazn.xml

OC4J-OID association is effective at the level of the OC4J `home` instance. After you have associated OC4J with Oracle Internet Directory, the location, user, password, and LDAP protocol configurations are reflected in the bootstrap `jazn.xml` file. Here is a sample entry:

```
<jazn provider="LDAP" location="ldap://myoid.oracle.com:389" default-realm="us" >
   <property
      name="ldap.user"
      value="orclApplicationCommonName=jaznadmin1,cn=JAZNContext,cn=products,
             cn=OracleContext"/>
   <property name="ldap.password"
             value="{903}3o4PTHbgMzVlzbVfKITIO5Bgio6KK9kD"/>
   <property name="ldap.protocol" value="no-ssl"/>
</jazn>
```

The default realm "us" corresponds to the default identity management realm in Oracle Internet Directory. Supported ldap.protocol settings are "ssl" or "no-ssl", according to whether you use SSL connections. The default is to use SSL, so if you specify SSL when you use Application Server Control, this does not actually result in any ldap.protocol setting.

Note: In runtime, the LDAP-based provider connects as user jaznadmin to Oracle Internet Directory. This user is a member of JAZNAdminGroup.

See Also:

- "Configuring LDAP User and SSL Properties" on page 6-18

Associating the OC4J System Application with Oracle Internet Directory

There may be situations where, after associating OC4J with Oracle Internet Directory, you also need to specifically associate the OC4J system application with Oracle Internet Directory. This would be the case, for example, if you want to perform operations on your application (such as startup and shutdown) with admin_client.jar, which executes through the system application, and have admin_client.jar be aware of Oracle Internet Directory user accounts. This requires the following manual steps:

1. Copy the <jazn> element from the instance-level jazn.xml file (discussed in the preceding section, "Oracle Internet Directory Association in jazn.xml") to the system-application.xml file, overwriting the existing <jazn> element in system-application.xml. This results in the system application using Oracle Identity Management (instead of the default file-based provider) as its security provider.
2. Map or create an anonymous user in Oracle Internet Directory. You have two choices:
   - Map an anonymous user to an existing Oracle Internet Directory user.
   - Create an anonymous Oracle Internet Directory user.

   These procedures are described immediately below.

   See Also:

   - "OC4J System Application" on page 3-10

Map an Anonymous User You can map an anonymous user to an existing Oracle Internet Directory user through a <property> element under the <jazn> element in the instance-level jazn.xml file, for the anonymous.user property. For example, assuming there is a user myoiduser in Oracle Internet Directory: You can map an anonymous user to an existing Oracle Internet Directory user through a <property> element under the <jazn> element in the instance-level jazn.xml file, for the anonymous.user property. For example, assuming there is a user myoiduser in Oracle Internet Directory:

```
<jazn ... >
   <property name="anonymous.user" value="myoiduser" />
   ...
</jazn>
```

Create an Anonymous User You can use the `ldapmodify` utility to create an anonymous user account in Oracle Internet Directory.

First, create an LDIF (lightweight directory interchange format) file to use as input for `ldapmodify`. Here is an example of an appropriate LDIF file:

```
dn: cn=anonymous, cn=Users, yourDistinguishedName
changetype: add
uid: anonymous
givenName: anonymous
cn: anonymous
sn: anonymous
description: This entry is used as the identification for unauthenticated users.
orclisenabled: disabled
objectClass: top
objectclass: person
objectclass: organizationalPerson
objectClass: inetorgperson
objectClass: orcluser
objectClass: orcluserV2
```

Note that you must replace *yourDistinguishedName* by the distinguished name of the default identity management realm in Oracle Internet Directory.

After you have created the `anony.ldif` file, use `ldapmodify` to add the anonymous user, as follows:

```
% ORACLE_HOME/bin/ldapmodify -D cn=orcladmin -w password -h hostname -p port \
                             -f anony.ldif
```

When you issue this command, replace *password*, *hostname*, and *port* with the password, host name, and port for your installation.

Notes:

- The `anonymous` account is a special user account created in the Oracle Internet Directory server for OC4J server usage purpose only. Because this account is created without a password, this account cannot be used by an end user to log in to the applications.
- The `ldapmodify` tool is a standard LDAP utility and is provided with Oracle Internet Directory in *ORACLE_HOME*`/bin` in your Oracle Application Server infrastructure installation.

Configure Oracle Identity Management as the Security Provider

This section covers the step of specifying Oracle Identity Management as the security provider for your application, using the Application Server Control Console. The following subtopics are covered:

- Specifying Oracle Identity Management during Deployment
- Changing to Oracle Identity Management after Deployment

Notes:

- Procedures discussed throughout this section assume you are logged in to Application Server Control as a user with required administrative permissions (as `oc4jadmin`, for example).
- To enable fat client access to EJBs using RMI, you must grant RMI permission "login" to your user or role. When using the Oracle Identity Management security provider, you can accomplish this through the OracleAS JAAS Provider Admintool. For example:

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

Specifying Oracle Identity Management during Deployment

Assuming you have satisfied appropriate requirements discussed earlier, in "Steps to Use the Oracle Identity Management Security Provider" on page 6-9, you can specify Oracle Identity Management (the LDAP-based provider) when you deploy an application through Application Server Control.

From the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page):

1. Go to the Select Security Provider task.
2. In the resulting Deployment Settings: Select Security Provider page, choose Oracle Identity Management from the Security Provider dropdown list.
3. Under "Configuration of Oracle Identity Management Security Provider" (which appears after you choose Oracle Identity Management from the dropdown), do the following:
   - Confirm the Oracle Internet Directory host and port are correct, as established earlier when you associated the Oracle Internet Directory instance with your OC4J instance.
   - Optionally enable SSO authentication. This results in the configuration `auth-method="SSO"` in `orion-application.xml` for your application, as discussed in "Configuring OC4J for OracleAS Single Sign-On" on page 13-3.
4. Choose **OK** to finish the security provider selection.
5. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11.

Notes:

- Specifying Oracle Identity Management as the security provider for your application results in the setting `provider="LDAP"` in the `<jazn>` element in `orion-application.xml`.
- During deployment, there is no need to specify the Oracle Internet Directory location, since this was already specified when you associated OC4J with Oracle Internet Directory (and is reflected in the `<jazn>` element in `jazn.xml`).
- The default realm is the default Oracle Identity Management realm. This is determined when Oracle Internet Directory is installed.

Changing to Oracle Identity Management after Deployment

You can select a security provider for your application at deployment time, as described above. You can also change to a different security provider after deployment. Assuming you have completed the prerequisites outlined in "Steps to Use the Oracle Identity Management Security Provider" on page 6-9, you can change to Oracle Identity Management as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, choose "Change Security Provider".
3. In the Change Security Provider page, select Oracle Identity Management Security Provider from the Security Provider Type dropdown.
4. Under "Security Provider Attributes: Oracle Identity Management Security Provider" (which appears after you select Oracle Identify Management in the dropdown):
   - Confirm the Oracle Internet Directory host and port are correct, as established earlier when you associated the Oracle Internet Directory instance with your OC4J instance.
   - Optionally enable SSO authentication. This results in the configuration `auth-method="SSO"` for your application, as discussed in "Configuring OC4J for OracleAS Single Sign-On" on page 13-3.
5. Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you can examine the settings.

Configure SSO (Optional)

This step is required only if you want to use OracleAS Single Sign-On functionality with Oracle Identity Management. The following subtopics are covered:

1. Run the SSO Registration Tool
2. Transfer the osso.conf File to the OC4J Instance
3. Run the osso1013 Script
4. Synchronization of OracleAS JAAS Provider User Context with Servlet Sessions
5. Restart the Oracle HTTP Server and OC4J Instances

See Also:

- *Oracle Application Server Administrator's Guide*, which also documents these steps

Run the SSO Registration Tool

The first task in configuring OracleAS Single Sign-On is to register your application as a partner application with the single sign-on server in your infrastructure. This is a post-installation step. Accomplish this by running the `ssoreg` utility in your infrastructure installation (the SSO server system) to create an (obfuscated) `osso.conf` file.

The `ssoreg` utility is *ORACLE_HOME*`/sso/bin/ssoreg.sh` in a Linux installation or *ORACLE_HOME*`\sso\bin\ssoreg.bat` in a Windows installation.

Here is the 10.1.2.0.2 syntax for `ssoreg` options required for this usage, with options described in Table 6–2:

```
-oracle_home_path path
-site_name name
-config_mod_osso TRUE
-mod_osso_url url
-remote_midtier
-config_file path
```

Table 6–2 Key ssoreg Options

Option	Description
`oracle_home_path`	The absolute path to the *ORACLE_HOME* location in your infrastructure installation.
`site_name`	Name of the Web site, such as `www.example.com`.
`config_mod_osso`	A `TRUE` setting (which is what you want) indicates that `mod_osso`, the Apache mod for OracleAS Single Sign-On, is effectively the application being registered. (Actually, your application is being registered through `mod_osso`.) This results in an obfuscated `osso.conf` file being generated.
`mod_osso_url`	A URL consisting of the host name and port where your application will run: `http://www.example.com:7777`
`remote_midtier`	When present on the command line, specifies that the application being registered is on a remote middle tier. Because your OC4J installation is on a different tier (with a different *ORACLE_HOME*) than your infrastructure, including OracleAS Single Sign-On, you must include this option.
`config_file`	Desired location of the `osso.conf` file, typically something like: *ORACLE_HOME*`/Apache/Apache/conf/osso/osso.conf`

Here is a Linux example (assume that `$ORACLE_HOME` has been set properly).

```
% $ORACLE_HOME/sso/bin/ssoreg.sh -oracle_home_path $ORACLE_HOME \
  -site_name myhost.mydomain.com -config_mod_osso TRUE \
  -mod_osso_url http://myhost.mydomain.com:7777 -remote_midtier \
  -config_file $ORACLE_HOME/Apache/Apache/conf/osso/osso.conf
```

Important:

- To use OracleAS Single Sign-On with Oracle Identity Management as the security provider under OracleAS JAAS Provider in a 10.1.2.0.x infrastructure, you must upgrade to 10.1.2.0.1 or higher. In a 9.0.4.x infrastructure, you must upgrade to 9.0.4.3. (This is also noted in "Supported Versions for Oracle Internet Directory and OracleAS Single Sign-On" on page 6-8.) Older versions do not support the `-remote_midtier` option, and ignoring this option may cause unintended changes in Oracle Application Server Distributed Configuration Management (DCM) on the host where you run the command.
- In a 9.0.4 infrastructure, the utility is `ossoreg.jar`, but the functionality is essentially the same. You must still have a version that supports `-remote_midtier`.
- In a 9.0.4 infrastructure, you must include a `-u` setting for `ssoreg`, to set the user ID to start the Apache process:

```
-u SYSTEM        (for Windows)
-u root          (for Linux)
```

See Also:

- *Oracle Application Server Single Sign-On Administrator's Guide* for additional information about the `ssoreg` utility, including options not mentioned here

Transfer the osso.conf File to the OC4J Instance

Transfer, such as by FTP, the `osso.conf` file produced during SSO registration (at your infrastructure installation, after installation) to a desired location on the OC4J middle tier.

Run the osso1013 Script

At your OC4J installation, run a script called `osso1013` to complete the SSO registration process, specifying the location where you placed the `osso.conf` file.

```
% osso1013 path/osso.conf
```

This script is located in the *ORACLE_HOME*`/Apache/Apache/bin` directory.

On Windows, you may have to run it through Perl:

```
% perl osso1013 path/osso.conf
```

Synchronization of OracleAS JAAS Provider User Context with Servlet Sessions

For situations where a Web application is used with the Oracle Identity Management security provider and with OracleAS Single Sign-On (acting as the login, timeout, and logout service), the OC4J 10.1.3 implementation supports synchronization between the OracleAS JAAS Provider user context and the servlet session.

With this synchronization, if there is an SSO logout or timeout, after which the user tries to access a protected resource, he or she receives the SSO login prompt again. (This does not occur if the user is only trying to access a public resource.)

This synchronization is disabled by default. You can enable it (or explicitly disable it) through the property `sso.session.synchronize` under the `<jazn-web-app>` element in the `orion-application.xml` file. The following example enables it:

```
<orion-application ... >
   ...
   <jazn ... >
      ...
      <jazn-web-app ... >
         <property name="sso.session.synchronize" value="true" />
      </jazn-web-app>
      ...
   </jazn>
   ...
</orion-application>
```

Notes:

- The `orion-web.xml` file, used to configure a single Web application, also supports the `<jazn-web-app>` element, as a subelement of `<orion-web-app>`. In the event of competing settings, the `orion-web.xml` setting takes precedence for the particular Web application.
- For SSO timeout to work, you must enable the SSO timeout header in OracleAS Single Sign-On. Refer to the *Oracle Application Server Single Sign-On Administrator's Guide* for details.

Restart the Oracle HTTP Server and OC4J Instances

You must restart Oracle HTTP Server and OC4J for the registration to take effect.

LDAP-Based Provider Settings in OC4J Configuration Files

This section describes how to configure aspects of the LDAP-based Oracle Internet Directory, covering the following topics:

- Configuring LDAP User and SSL Properties
- Configuring LDAP Connection Properties
- Configuring LDAP Caching Properties

See Also:

- *Oracle Identity Management Guide to Delegated Administration* for information about creating users and roles, through the Oracle Delegated Administration Services (DAS), when using Oracle Identity Management

Configuring LDAP User and SSL Properties

Table 6–3 summarizes LDAP user and SSL properties, supported through `<property>` subelements under the `<jazn>` element in the bootstrap `jazn.xml` file. These parameters are set as appropriate through your configuration in Application Server Control Console when you associate OC4J with Oracle Internet Directory, described earlier in this chapter.

The resulting configuration is as follows:

```
<jazn ... >
   ...
   <property name="propname" value="propvalue" />
   ...
</jazn>
```

You must restart OC4J for the changes to take effect.

Table 6–3 LDAP SSL Properties and Related Properties

Property Name	Property Definition
`ldap.user`	LDAP user name or distinguished name. This element is populated automatically; you should not change the contents. For example: `orclApplicationCommonName=jaznadmin1,cn=JAZNContext, cn=products,cn=OracleContext`
`ldap.password`	Obfuscated password for the LDAP user name. For example: `{903}oZZYqmGc/iyCaDrD4qs2FHbXf3LAWtMN` **See Also**: "Password Obfuscation in OC4J Configuration Files" on page 5-3 for details on obfuscation.
`ldap.protocol`	Determines whether to use SSL. (By default, SSL is used.) Supported settings are "`ssl`" (typically used with port 636) or "`no-ssl`" (typically used with port 389). **Note**: As an alternative to the "`ssl`" setting, you can use the protocol "`ldaps://`" in the LDAP URL.

Oracle Internet Directory supports NULL authentication for SSL communication. Data are encrypted with the Anonymous Diffie-Hellman cipher suite, but no certificates are used for authentication.

See Also:

- Table 12–1, " Cipher Suites Supported by OracleSSL" for a list of supported cipher suites

Here is a sample configuration:

```
<?xml version = '1.0' encoding = 'UTF-8'?>
<jazn provider="LDAP" location="ldap://www.example.com:389" default-realm="us">
   <property name="ldap.protocol" value="no-ssl"/>
</jazn>
```

See Also:

- *Oracle Internet Directory Administrator's Guide*

Configuring LDAP Connection Properties

Table 6–4 summarizes LDAP connection properties. Table 6–5 summarizes properties for the LDAP JNDI connection pool. You can set these properties in `<property>` subelements under the `<jazn>` element in the instance-level `jazn.xml` file, as follows:

```
<jazn ... >
   ...
   <property name="propname" value="propvalue" />
   ...
</jazn>
```

You must restart OC4J for the changes to take effect.

Table 6–4 LDAP Connection Properties

Property Name	Property Definition	Default Value
`ldap.connect.max.retry`	Number of times the security provider attempts to create an LDAP connection before giving up	5
`ldap.connect.sleep.time`	Number of milliseconds the security provider waits before retrying a failed LDAP connection attempt	5000

Table 6–5 LDAP JNDI Connection Pool Properties

Property Name	Property Definition	Default Value
`jndi.ctx_pool.init_size`	Initial size for the LDAP JNDI connection pool	5
`jndi.ctx_pool.inc_size`	Pool increment size for the LDAP JNDI connection pool—number of connections added to pool whenever the supply of connections in the pool is exhausted	10

Note: The configurations discussed here must be performed manually; there is currently no support for these in Application Server Control.

Configuring LDAP Caching Properties

Oracle Internet Directory 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 (available only to Web-based clients with cookies enabled)

The caching service maintains a global hashmap (`java.util.HashMap` instance) that is used to store and retrieve cached objects. Expired objects in the hashmap are periodically invalidated and cleaned up automatically, as appropriate. Objects in the cache expire based on a time-to-live algorithm; expiration time can be set through the cache properties described below.

Note: Only Oracle Internet Directory supports these caches. The file-based provider defaults to caching the entire XML document.

Table 6–6 describes LDAP caching properties and their default values. You can set these properties in `<property>` subelements under the `<jazn>` element in the instance-level `jazn.xml` file, as follows:

```
<jazn ... >
   ...
   <property name="propname" value="propvalue" />
   ...
```

```
</jazn>
```

Table 6–6 LDAP Cache Properties

Property	Description	Default
`ldap.cache.policy.enable`	If set to `true`, enables policy cache; if set to `false`, disables cache.	`true`
`ldap.cache.realm.enable`	If set to `true`, enables realm cache; if set to `false`, disables cache.	`true`
`ldap.cache.session.enable`	If set to `true`, enables session cache; if set to `false`, disables cache.	`true`
`ldap.cache.initial.capacity`	Initial capacity for the hashmap. This property affects performance; it is important to not set it too low.	20
`ldap.cache.load.factor`	Load factor for the hashmap. This is a measure of how full to allow the cache to get before the capacity is automatically increased. This property affects performance; it is important to not set it too high.	0.7
`ldap.cache.purge.initial.delay`	String containing an integer that represents the number of milliseconds the daemon thread waits before it starts checking for expired objects.	3600000 (one hour)
`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 (one hour)

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 example disables all three caches:

```
<jazn provider="LDAP" location="ldap://myhost.example.com:636" >
   ...
   <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>
```

Or, as startup parameter settings:

```
-Dldap.cache.session.enable=false
-Dldap.cache.realm.enable=false
-Dldap.cache.policy.enable=false
```

The following example leaves all caches enabled, and sets a cache size of 100 and a 10,000-millisecond timeout:

```
< jazn provider="LDAP" location="ldap://myhost.example.com:636" >
   < property name="ldap.cache.initial capacity" value="100" />
   < property name="ldap.cache.purget.timeout" value="10000" />
</jazn>
```

Notes:

- The OracleAS JAAS Provider Admintool automatically disables caching while it is in operation, then reenables caching when it finishes.
- The configurations discussed here must be performed manually; there is currently no support for these in Application Server Control.

7

File-Based Security Provider

OC4J supplies a file-based security provider, where an XML-based file is used as the repository for users, roles, and policies. This is the default security provider. Specifically, OracleAS JAAS Provider supports the following tasks for the file-based (XML-based) provider:

- 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 repository, typically, `system-jazn-data.xml`, although you have the option of using an application-specific `jazn-data.xml` file instead.

This chapter discusses basic user, role, and realm management tasks for the file-based provider, focusing on features of the Application Server Control Console.

The chapter is divided into the following sections:

Notes:

- Be aware that with the file-based provider, role comparisons for authorization are case-sensitive.
- By default, the file-based provider is the security provider, the `system-jazn-data.xml` file is the repository, and `jazn.com` is the default realm. The `system-jazn-data.xml` file is located in the *`ORACLE_HOME`*`/j2ee/`*`instance_name`*`/config` directory. Changes made to this repository are visible to all applications that use it.

Tools for File-Based Provider Policy and Realm Management

To manage users and roles for the file-based provider, use Application Server Control Console, as described in "Managing Application Realms through Application Server Control" on page 7-4. This updates the user repository, either

system-jazn-data.xml or an application-specific jazn-data.xml file that you provide.

To manage policies for the file-based provider, use the OracleAS JAAS Provider Admintool. Refer to the policy options listed in "Summary of Admintool Command-Line Syntax and Options" on page C-2.

Generally avoid direct manipulation of the system-jazn-data.xml or jazn-data.xml file.

Note: There is one exception regarding the tool for policy management: Granting RMI permission or Administration permission to a role in the file-based provider is something you can do as part of editing or adding the role through Application Server Control, as described later in this chapter.

Note that to enable fat client access to EJBs using RMI, you must grant RMI permission "login" to your user or role. If you do not enable this through Application Server Control, you can use the OracleAS JAAS Provider Admintool. For example:

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

See Also:

- Appendix C, "OracleAS JAAS Provider Admintool Reference"

Configuring the File-Based Provider in Application Server Control

This section covers the following administration tasks, using the Application Server Control Console, for an application using the file-based provider. There is also a section at the end for instance-level administration.

- Configuring the File-Based Provider during Application Deployment
- Changing to the File-Based Provider after Deployment
- Managing Application Realms through Application Server Control
- Managing Application Users through Application Server Control
- Managing Application Roles through Application Server Control
- Administering Instance-Level Security through Application Server Control

---

Note:

- Procedures discussed throughout this section assume you are logged in to Application Server Control as a user with required administrative permissions (as oc4jadmin, for example).
- Security provider settings, optionally including specification of the repository file, affect settings in the <jazn> element of the orion-application.xml file. Realm, user, and role settings affect settings under the <jazn-realm> element in the repository file. Examples of the XML settings are in "File-Based Provider Settings in OC4J Configuration Files" on page 7-9.

See Also:

- "File-Based Provider Settings in OC4J Configuration Files" on page 7-9, for examples of the XML configuration that results from the steps described in this section

Configuring the File-Based Provider during Application Deployment

You can specify the file-based provider when you deploy an application through Application Server Control. Optionally, you can also specify a `jazn-data.xml` file location and a default realm.

From the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page):

1. Go to the Select Security Provider task.
2. In the resulting Deployment Settings: Select Security Provider page, choose File-Based from the Security Provider dropdown list.
3. Under "Configuration of File-Based Security Provider" (which appears after you choose the file-based provider in the dropdown), you can accomplish the following:
   - Specify the location of your repository, optionally an application-specific `jazn-data.xml` file for user and role configuration. Otherwise, the `system-jazn-data.xml` file will be used, unless `jazn.xml` also specifies the file-based provider and has a location setting for a `jazn-data.xml` file.
   - Specify a default realm. Otherwise, the default realm is `jazn.com`, unless there is a different setting in the instance-level `jazn.xml` file.
4. Choose **OK** to finish the security provider selection.
5. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11.

Changing to the File-Based Provider after Deployment

You can select a security provider for your application at deployment time, as described above. You can also change to a different security provider after deployment. You can change to the file-based provider as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, choose "Change Security Provider".
3. In the Change Security Provider page, select File-Based Security Provider from the Security Provider Type dropdown.
4. Under "Security Provider Attributes: File-Based Security Provider" (which appears after you select "File-Based Security Provider"):
   - Optionally specify the location of your repository file, such as an application-specific `jazn-data.xml` file. Otherwise, the `system-jazn-data.xml` file will be used, unless `jazn.xml` also specifies the file-based provider and has a location setting for a `jazn-data.xml` file.
   - Optionally specify a default realm. Otherwise, the default realm is `jazn.com`, unless there is a different setting in the instance-level `jazn.xml` file.

5. Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you can examine your settings.

Managing Application Realms through Application Server Control

This section describes how to configure realms for the file-based provider.

The first step for any of these instructions is to go to the Application Server Control Console Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.

The tasks here create or modify subelements under the `<jazn-realm>` element in your repository file. There is a `<realm>` subelement under `<jazn-realm>` for each realm.

> **Note:** There is no "Edit" task for realms. Editing a realm includes updating users, roles, or both, as described in "Managing Application Users through Application Server Control" on page 7-5 and "Managing Application Roles through Application Server Control" on page 7-6.

Search for a Realm

From the Security Provider page for your application, execute the following steps to search for a realm:

1. Choose the **Realms** tab.
2. In the Realms page, under "Search", specify a search string then choose **Go**.
3. Realms matching the search string appear under "Results". (An empty search string displays all existing realms.)

Create a Realm

From the Security Provider page for your application, execute the following steps to create a realm:

1. Choose the **Realms** tab.
2. Above the list of existing realms, choose **Create**.
3. In the resulting Add Realm page:
   - Specify the desired name of the realm.
   - Specify the desired name for the administrator user of the realm.
   - Specify and confirm the desired password for the administrator user.
   - Specify the desired administrator role of the realm. The administrator user you specified will belong to this realm.
4. Choose **OK** to create the realm.

This takes you back to the Security Provider page, where you can see the new realm in the list of realms.

Delete a Realm

From the Security Provider page for your application, execute the following steps to delete a realm:

1. In the list of existing realms, choose the Delete task for the realm you want to delete.
2. In the resulting Confirmation page, choose **Yes** to delete the realm.

This takes you back to the Security Provider page.

Managing Application Users through Application Server Control

This section describes how to configure users for the file-based provider.

The first step for any of these instructions is to go to the Application Server Control Console Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.

The tasks here create or modify subelements under a `<users>` element in your repository file. Each `<realm>` element has a `<users>` subelement for the users in that realm.

Search for a User

From the Security Provider page for your application, execute the following steps to search for a user:

1. Choose the **Realms** tab.
2. In the Realms page, under "Users" in the list of realms, and in the row for the realm of interest, select the number that shows how many users are in the realm. This is a link to the Users page for the realm.
3. In the Users page, under "Search", specify a search string then choose **Go**.
4. Users matching the search string appear under "Results". (An empty search string displays all users in the realm.)

Create a User

From the Security Provider page for your application, execute the following steps to create a user:

1. Choose the **Realms** tab.
2. In the Realms page, under "Users" in the list of realms, and in the row for the realm of interest, select the number that shows how many users are in the realm. This is a link to the Users page for the realm.
3. In the Users page, above the list of existing users in the realm, choose **Create**.
4. In the resulting Add User page:
   - Specify the desired user name.
   - Specify and confirm the desired password for the user.
   - Under "Assign Roles", for any available role you want the user to belong to, move the role name into the "Selected Roles" column.
   - Choose **OK** to add the user.

This takes you back to the Users page, where you can see the new user in the list of users.

Note: Do not create user names that contain slash (/) characters, as in a/b/c.

Delete a User

From the Security Provider page for your application, execute the following steps to delete a user:

1. Choose the **Realms** tab.
2. In the Realms page, under "Users" in the list of realms, and in the row for the realm of interest, select the number that shows how many users are in the realm. This is a link to the Users page for the realm.
3. In the Users page, choose the Delete task for the user you want to delete.
4. In the resulting Confirmation page, choose **Yes** to delete the user.

This takes you back to the Users page.

Edit a User

From the Security Provider page for your application, execute the following steps to edit the properties of a user:

1. Choose the **Realms** tab.
2. In the Realms page, under "Users" in the list of realms, and in the row for the realm of interest, select the number that shows how many users are in the realm. This is a link to the Users page for the realm.
3. In the Users page, select the user you want to edit.
4. In the resulting User page:
   - If you want to change the user password, enter the old password, then specify and confirm the desired new password.
   - If you want to add the user to any roles or remove the user from any roles, under "Assign Roles", move role names into or out of the "Selected Roles" column as desired.
   - Choose **Apply** to edit the user.

This takes you back to the Users page.

Note: You can also reach the User page for a given user from the Role page (see "Edit a Role" on page 7-8) for any role that the user belongs to. In the Role page, under "Users", select the user of interest.

Managing Application Roles through Application Server Control

This section describes how to configure roles for the file-based provider.

The first step for any of these instructions is to go to the Application Server Control Console Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.

The tasks here create or modify subelements under a <roles> element in your repository file. Each <realm> element has a <roles> subelement for the roles in that realm.

Search for a Role

From the Security Provider page for your application, execute the following steps to search for a role:

1. Choose the **Realms** tab.
2. In the Realms page, under "Roles" in the list of realms, and in the row for the realm of interest, select the number that shows how many roles are in the realm. This is a link to the Roles page for the realm.
3. In the Roles page, under "Search", specify a search string then choose **Go**.
4. Roles matching the search string appear under "Results". (An empty search string displays all roles in the realm.)

Create a Role

From the Security Provider page for your application, execute the following steps to create a role:

1. Choose the **Realms** tab.
2. In the Realms page, under "Roles" in the list of realms, and in the row for the realm of interest, select the number that shows how many roles are in the realm. This is a link to the Roles page for the realm.
3. In the Roles page, above the list of existing users in the realm, choose **Create**.
4. In the resulting Add Role page:
   - Specify the desired role name.
   - Choose the permissions you want to grant to the role (essentially, to users or other entities belonging to the role)—RMI permission, administration permission, neither, or both.

     A user needs RMI (remote method invocation) permission to be able to access objects on OC4J through RMI, such as when using a remote EJB client.

     A user needs administration permission to perform administrative functions such as startup, shutdown, and configuration changes.
   - Under "Assign Roles", for any available role you want the new role to inherit from, move the role name into the "Selected Roles" column.
   - Choose **OK** to add the role.

This takes you back to the Roles page, where you can see the new role in the list of roles.

Delete a Role

From the Security Provider page for your application, execute the following steps to delete a role:

1. Choose the **Realms** tab.
2. In the Realms page, under "Roles" in the list of realms, and in the row for the realm of interest, select the number that shows how many roles are in the realm. This is a link to the Roles page for the realm.
3. In the Roles page, choose the Delete task for the role you want to delete.
4. In the resulting Confirmation page, choose **Yes** to delete the role.

This takes you back to the Roles page.

Edit a Role

From the Security Provider page for your application, execute the following steps to edit the properties of a role:

1. Choose the **Realms** tab.
2. In the Realms page, under "Roles" in the list of realms, and in the row for the realm of interest, select the number that shows how many roles are in the realm. This is a link to the Roles page for the realm.
3. In the Roles page, select the role you want to edit.
4. In the resulting Role page:
   - Update permissions for the role as desired, by selecting or unselecting RMI permission and administration permission.
   - Under "Assign Roles", move role names into or out of the "Selected Roles" column, depending on which roles you want this role (the role you are editing) to inherit from.
   - Choose **Apply** to edit the role.

This takes you back to the Roles page.

See Also:

- "Edit a User" on page 7-6 for how to add a user to a role

Administering Instance-Level Security through Application Server Control

In the OC4J 10.1.3 implementation, instance-level security uses the file-based provider for realm settings (users and roles). This is according to settings in the `<jazn>` element of the OC4J `system-application.xml` file, which points to the `system-jazn-data.xml` file for the user and role repository. You can administer the file-based provider for instance-level security in much the same was as you would administer the file-based provider for an application. You can navigate to the Application Server Control Console Instance Level Security page as follows:

1. From the OC4J Home page for the OC4J instance, choose the **Administration** tab.
2. In the Administration page, choose the Security Providers task (under "Security").
3. In the Security Providers page, choose **Instance Level Security**.
4. From the resulting Instance Level Security page, you can manage instance-level realms, users, and roles using essentially the same steps as documented earlier in this chapter, in "Managing Application Realms through Application Server Control" on page 7-4, "Managing Application Users through Application Server Control" on page 7-5, and "Managing Application Roles through Application Server Control" on page 7-6.

Instance-level security settings here will always affect the `system-jazn-data.xml` file (as opposed to application-level security settings, which would affect an application-level `jazn-data.xml` file if the user has specified one).

Note: Be aware that OC4J has some dependencies on the instance-level security provider settings in `system-application.xml` and `system-jazn-data.xml`. For example, `admin_client.jar` uses accounts in `system-jazn-data.xml`. Do not delete or alter default settings in these files regarding the instance-level security provider and related accounts.

File-Based Provider Settings in OC4J Configuration Files

This section provides reference information for important security configuration for the file-based provider in key OC4J configuration files. In general, you should use the Application Server Control Console or OracleAS JAAS Provider Admintool (both discussed earlier in this chapter) for configuration and administration, instead of manipulating the files directly. Using these tools results in the appropriate entries automatically being made in the configuration files.

The rest of this discussion covers the following:

- Scenarios for <jazn> Settings in orion-application.xml
- Realm Configuration in the Repository File
- Policy Configuration in the Repository File
- Predefined OC4J Accounts in system-jazn-data.xml

Scenarios for <jazn> Settings in orion-application.xml

The `<jazn>` element, which appears in both the `jazn.xml` file and the `orion-application.xml` file, includes configuration for the security provider, repository, and default realm. By default, the `system-jazn-data.xml` file is the repository for user, role, and policy configuration for the file-based provider, but OC4J can be configured to use an application-specific `jazn-data.xml` file instead.

There are three typical deployment scenarios for an application, as determined by `<jazn>` element settings in the `orion-application.xml` file and instance-level `jazn.xml` file, in using the file-based provider:

- Delegate to the instance-level `jazn.xml` file for the repository and default realm. If the `<jazn>` element in `jazn.xml` has the setting `provider="XML"`, then its settings for the repository (`location` attribute) and default realm (`default-realm` attribute) are used if the `orion-application.xml` file has the following `<jazn>` element:

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

  Or, if the `jazn.xml` file has no `location` and `default-realm` settings, this would use the default repository `system-jazn-data.xml` and the default realm `jazn.com`.

  Note: This becomes the default `<jazn>` setting if there is no `<jazn>` element in `orion-application.xml` when the application is deployed.

- Delegate to the instance-level `jazn.xml` file for the repository. If the `<jazn>` element in `jazn.xml` has the setting `provider="XML"`, then its setting for the

repository (`location` attribute) is used, but the `orion-application.xml` file setting for the default-realm (`default-realm` attribute) is used, if `orion-application.xml` has a `<jazn>` element such as the following:

```
<jazn provider="XML" default-realm="abc.com" />
```

Or, if the `jazn.xml` file has no `location` setting, this would use the default repository `system-jazn-data.xml`.

> **Note:** This example assumes the `abc.com` realm is defined in the `system-jazn-data.xml` repository.

- Do not delegate; specify both the repository and the default realm in `orion-application.xml`. In this example, `orion-application.xml` specifies the repository `jazn-data.xml` and the default realm `myrealm`:

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

> **Notes:** Note the following for situations where the application uses the file-based provider (`provider="XML"` in `orion-application.xml`) but the `jazn.xml` file has the setting `provider="LDAP"`:
>
> - If `orion-application.xml` specifies no repository file, then `system-jazn-data.xml` will be the repository.
> - If `orion-application.xml` specifies no default realm, then `jazn.com` file will be the default realm.

Realm Configuration in the Repository File

This section shows configuration for users and roles in the `system-jazn-data.xml` file for the `jazn.com` realm. The general structure would be the same for configuration of any realm in `system-jazn-data.xml` or a `jazn-data.xml` file. This configuration is created automatically when you manage realms through Application Server Control.

```
<jazn-realm>
  <realm>
    <name>jazn.com</name>
    <users>
      <user deactivated="true">
        <name>anonymous</name>
        <description>The default guest/anonymous user</description>
      </user>
      <user deactivated="true">
        <name>oc4jadmin</name>
        <display-name>OC4J Administrator</display-name>
        <description>OC4J Administrator</description>
        <credentials>!welcome</credentials>
      </user>
      <user>
        <name>JtaAdmin</name>
        <display-name>JTA Recovery User</display-name>
        <description>Used to recover propagated OC4J transactions</description>
        <credentials>!defaultJtaPassword</credentials>
      </user>
    </users>
```

```
    <roles>
      <role>
        <name>oc4j-administrators</name>
        <display-name>OC4J Admin Role</display-name>
        <description>Administrative role for OC4J</description>
        <members>
          <member>
            <type>user</type>
            <name>oc4jadmin</name>
          </member>
          <member>
            <type>user</type>
            <name>JtaAdmin</name>
          </member>
        </members>
      </role>
      <role>
        <name>oc4j-app-administrators</name>
        <display-name>OC4J Application Administrators</display-name>
        <description>OC4J application-level administrators</description>
        <members>
        </members>
      </role>
      <role>
        <name>users</name>
        <display-name>users</display-name>
        <description>users role for rmi/ejb access</description>
        <members>
        </members>
      </role>
    </roles>
  </realm>
</jazn-realm>
```

Policy Configuration in the Repository File

You can use the OracleAS JAAS Provider Admintool to grant JAAS permissions to custom principals, using the `-grantperm` option, as described in "Granting and Revoking Permissions" on page C-11.

Policy data is stored in the file `system-jazn-data.xml`. In the following example, a segment of this file grants the `admin` principal permission to log in.

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <class>oracle.security.jazn.samples.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>
```

Predefined OC4J Accounts in system-jazn-data.xml

The following accounts are predefined in `system-jazn-data.xml` for the file-based provider:

- `oc4jadmin` user (initially deactivated in standalone OC4J)
- `oc4j-administrators` role
- `oc4j-app-administrators` role
- `anonymous` user, initially deactivated
- `users` role
- `jtaadmin` user

See Also:

- "Predefined OC4J Accounts" on page 3-11 for additional information about these accounts
- "Activation of the oc4jadmin Account" on page 3-12

OracleAS JAAS Provider Migration Tool

OC4J includes a tool for migrating from a file-based repository to either an Oracle Internet Directory repository or an alternative file-based repository. (Do not confuse this with the tool for migrating from `principals.xml`; that is separate, and is documented later in this chapter.)

When migrating to an Oracle Internet Directory repository, the output is an LDIF file, which can be imported into Oracle Internet Directory using commands such as `ldapmodify` or `bulkload`.

Overview of the Migration Tool

The migration tool supports the migration of users, roles, role memberships, and policies (permissions granted to roles, users, custom principals, or codebases).

There are three modes for migration:

- *Realm mode* migrates only users and roles. All users and roles in the source realm, other than deactivated users, are migrated. Migrated roles include membership information.
- *Policy mode* migrates grantees and the permissions that have been granted to them. Grantees can be realm grantees, such as users and roles, or non-realm grantees, such as custom principals and codebases. When migrating to Oracle Internet Directory, realm grantees and their permissions are migrated to the policy that is specific to the destination realm, while non-realm grantees and their permissions are migrated to the global policy.
- *All mode* combines realm mode and policy mode.

Notes:

- When output to an LDIF file is generated, passwords are in clear text. It is your responsibility to take proper care in protecting this information.
- When migrating to Oracle Internet Directory, passwords may have to be modified to conform to Oracle Internet Directory requirements (such as having at least one numeric character).
- If you are migrating custom permissions, the JAR file containing the class files for the custom permissions must be available in the classpath.
- The migration tool is not intended for migration of indirect password accounts to Oracle Internet Directory.
- Be aware of the possibility of conflict—migrated users and roles may already exist in the destination realm. When migrating to Oracle Internet Directory, for example, commands such as `ldapmodify` and `bulkload` can be used in conjunction with standard JDK logging to obtain information that will help you to recover from conflicts.

Migration Tool Command Syntax

Command-line options and syntax of the migration tool are as follows:

```
% java JAZNMigrationTool [-st xml] [-dt ldap|xml]
                         [-D binddn] [-w passwd] [-h ldaphost] [-p ldapport]
                         [-sf sourcefilename] [-df destfilename]
                         [-sr source_realm] [-dr dest_realm]
                         [-m policy|realm|all]
                         [-help]
```

Table 7–1 describes these options.

Table 7–1 OracleAS JAAS Provider Migration Tool Options

Option	Description	Default (where applicable)
`-help`	To display option information	
`-st`	Type of provider at the source Currently only the setting `xml` is supported, for migrating from a file-based provider.	`xml`
`-dt`	Type of provider at the destination—either `xml` (to migrate to a file-based repository) or `ldap` (to migrate to Oracle Internet Directory)	`ldap`
`-D`	Oracle Internet Directory user name (for migration to Oracle Internet Directory only)	
`-w`	Oracle Internet Directory user password (for migration to Oracle Internet Directory only)	
`-h`	Oracle Internet Directory host system (for migration to Oracle Internet Directory only)	According to `<jazn>` element `location` setting in `jazn.xml`
`-p`	Oracle Internet Directory port (for migration to Oracle Internet Directory only)	According to `<jazn>` element `location` setting in `jazn.xml`

Table 7–1 (Cont.) OracleAS JAAS Provider Migration Tool Options

Option	Description	Default (where applicable)
`-sf`	Source file—path to the file-based repository you are migrating from	*ORACLE_HOME*`/j2ee/home/config/system-jazn-data.xml`
`-df`	Destination file—path to the LDIF output file (if migrating to Oracle Internet Directory) or to the destination file-based repository (if migrating to file-based)	If migrating to a file-based repository, *ORACLE_HOME*`/j2ee/home/config/system-jazn-data.xml` (otherwise no default)
`-sr`	Source realm—the realm you are migrating from	Name of the realm in the source repository, if there is only one realm
`-dr`	Destination realm—the realm you are migrating to	If migrating to a file-based repository, name of the realm in the destination repository, if there is only one realm; if migrating to Oracle Internet Directory, the default subscriber realm
`-m`	The desired migration mode—realm mode (`realm`), policy mode (`policy`), or both (`all`)	`all`

The following example migrates in `all` mode to the default subscriber realm in Oracle Internet Directory on the specified host:

```
% java oracle.security.jazn.tools.JAZNMigrationTool -D cn=orcladmin -w welcome1 \
       -h myhost.example.com -p 389 -sf /tmp/jazn-data.xml -df /tmp/dest.ldif \
       -sr jazndemo.com
```

Migration Tool APIs

You can also invoke the migration tool (class `JAZNMigrationTool` in package `oracle.security.jazn.tools`) from an application. Oracle provides the following APIs:

```
/**
 * Create an instance with the provided parameters. These parameters are
 * equivalent to the options supported by the executable utility version.
 */
public JAZNMigrationTool(Map params)

/**
 * Perform the migration operation
 */
public  void migrateData() throws JAZNException
```

The `params` parameter in the constructor supports the same options as described in Table 7–1 in the preceding section, with the same defaults. Parameter keys are defined as constants in the `JAZNMigrationTool` class. Table 7–2 shows the correlation between constants defined in `JAZNMigrationTool` and command-line options.

Table 7–2 JAZNMigrationTool Constants

Key Constant	Corresponds to Option
`SRC_TYPE`	`-st`
`DEST_TYPE`	`-dt`
`OID_USER`	`-D`
`OID_PASSWORD`	`-w`

Table 7–2 (Cont.) JAZNMigrationTool Constants

Key Constant	Corresponds to Option
OID_HOST	-h
OID_PORT	-p
SRC_FILE	-sf
DEST_FILE	-df
SRC_REALM	-sr
DEST_REALM	-dr
MIGRATE_OPT	-m

Migrating Principals from the principals.xml File

Use the OracleAS JAAS Provider Admintool `convert` option to migrate your data out of the `principals.xml` file, which is deprecated.

`-`**`convert`** *`filename realm`*

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

The migration converts `principals.xml` users to JAAS users and converts `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.

> **Note:** The `principals.xml` file is deprecated in the OC4J 10.1.3 implementation and will be desupported in a future release.

Before you convert `principals.xml`, you must make sure that you have an administrative 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.
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
   ```

   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` option does not migrate duplicate users, and migrates duplicate roles by overwriting the old one.
3. Migrate `principals.xml` to the `principals.com` realm, as in:

   ```
   % java -jar jazn.jar -convert config/principals.xml principals.com
   ```

4. Change the `<default-realm>` to `principals.com`; see "Scenarios for <jazn> Settings in orion-application.xml" on page 7-9.
5. Stop OC4J and restart it.

8

Login Modules

This chapter discusses how to configure the default login module, or how to implement, install, and configure a custom login module. The following topics are covered:

- Configuring RealmLoginModule
- Introducing Custom JAAS Login Modules
- Packaging and Deploying Login Modules
- Configuring the Custom Security Provider in Application Server Control
- Configuring Login Modules through the Admintool
- Login Module Configuration in OC4J Configuration Files
- Simple Login Module J2EE Integration
- Custom Login Module Example

Notes:

- Note the setting `provider="XML"` is used for custom providers (custom login modules) as well as for the file-based provider.
- Be aware that when you use a custom login module, role comparisons for authorization are *not* case-sensitive unless you add the following property setting to the `<jazn>` element in `orion-application.xml`:

  ```
  <property name="role.compare.ignorecase" value="false" />
  ```

- Because the JAAS specification does not cover user management, when you configure your application to use a custom login module, the use of the `UserManager` API within your application is not supported. The J2EE API, however, will continue to function within your application.
- OC4J provides a login module, `LDAPLoginModule`, for use with external LDAP providers, as noted in "External LDAP Provider Settings in system-jazn-data.xml" on page 9-5.
- OC4J provides a login module, `CoreIDLoginModule`, for use with Oracle COREid Access and Identity, as discussed in "Configure the COREid JAAS Login Module" on page 10-16.

See Also:

- "Login Module Authentication" on page 1-7

Configuring RealmLoginModule

The `RealmLoginModule` class is the default login module, for use with the file-based provider or Oracle Identity Management, and is configured through the `system-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-based, and so on).

You can configure `RealmLoginModule` by using Application Server Control or the OracleAS JAAS Provider Admintool.

The `<login-module>` element, under the `<jazn-loginconfig>` element in `system-jazn-data.xml`, supports the options shown in Table 8–1 for `RealmLoginModule` (through `<name>` and `<value>` subelements of an `<option>` element).

Notes:

- You do not need to enable the `RealmLoginModule` class if your application uses OracleAS Single Sign-On authentication.
- The use of `RealmLoginModule` as a custom login module—in other words, as a custom security provider—is not supported. However, as already noted, it is used by default as the login module when an application is configured to use the file-based provider (using `system-jazn-data.xml` or `jazn-data.xml` as the user repository) or Oracle Identity Management (using Oracle Internet Directory as the user repository).

See Also:

- "Login Module Settings in system-jazn-data.xml" on page 8-12
- "Configuring Login Modules through the Admintool" on page 8-11 for information on using the Admintool

Table 8–1 RealmLoginModule Options

Name	Meaning	Default
`debug`	If set to `true`, debugging messages are printed.	`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 example, password credentials) to the subject after successful authentication.	`false`

Table 8–1 (Cont.) RealmLoginModule Options

Name	Meaning	Default
supportCSIv2	If set to true, the RealmLoginModule supports CSIv2. **See Also**: Chapter 15, "Common Secure Interoperability Protocol" for details.	false
supportNullPassword	(Oracle Identity Management only) 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

Here is sample configuration of RealmLoginModule, in system-jazn-data.xml. (We recommend that you not alter RealmLoginModule configuration manually; this example is just for illustrative purposes.)

```
<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>
```

Introducing Custom JAAS Login Modules

Because OC4J 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 `RealmLoginModule` class as its default login module implementation. This class combines J2EE security constraints with either the file-based provider or Oracle Identity Management.)

A custom login module may be desirable, for example, when users and roles are defined in an external repository. When you create a custom login module, consider the following preliminary questions:

- Development: Do you want to take advantage of J2EE security constraints?
- Debugging: Do you want the login module to support a debugging option for use during development? (As noted previously, `RealmLoginModule`, for example, supports a `debug` option that provides diagnostic output. Also, "Custom Login Module Example" on page 8-16 includes debugging functionality.)
- Packaging and deployment: Are you using the login modules that come with J2SE 1.4? Or are you deploying custom or third-party login modules?

When you associate a custom login module with an application, the subject and the principals it contains are used as the sole basis for all authorization tasks, including evaluating J2EE security constraints. To ensure that all relevant principals are considered during authorization, the login module must add the relevant principals, including all roles that the authenticated user participates in, to the subject during the commit phase of the authentication process. (The `role.mapping.dynamic` property, discussed in "Settings in <jazn> for Login Modules" on page 8-13, is related to subject-based authorization.)

The custom login module framework supports the J2EE declarative security model. This means that subject-based authorization enforces the J2EE security constraints declared in application deployment descriptors (`web.xml` and `ejb-jar.xml`, for example).

Custom login modules are configured through the OC4J `system-jazn-data.xml` file, which can be updated automatically through use of tools such as Application Server Control Console and OracleAS JAAS Provider Admintool.

See Also:

- Sun Microsystems JAAS documentation for general information about login modules:

  `http://java.sun.com/j2se/1.4.2/docs/guide/security/jaas/JAASLMDevGuide.html`

Packaging and Deploying Login Modules

If you are using one or more of the default login modules provided with the J2SE (such as `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 one or more custom login modules, then you must deploy the login modules and configure the OracleAS JAAS Provider properly so that the module can be found at runtime. The following sections discuss ways to accomplish this:

- Deploying Login Modules within the J2EE Application
- Deploying Login Modules as Optional Packages
- Using Login Modules as OC4J Shared Libraries

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

Deploying Login Modules within the J2EE Application

If your login modules are used by only a single J2EE application, then you can simply package the login modules as part of your application by including the login module JAR file in the application EAR file.

The login modules must be configured through `<jazn-loginconfig>` settings, in one of two places:

- In the `system-jazn-data.xml` file, as discussed in "Login Module Settings in system-jazn-data.xml" on page 8-12
- In the `orion-application.xml` file in your application EAR file, as discussed in "Settings in <jazn-loginconfig> in orion-application.xml" on page 8-13

Using the Application Server Control Console, you can configure custom login modules as you deploy an application, or later if you change the security provider to custom. This results in `system-jazn-data.xml` being updated automatically.

Administering custom login modules through the OracleAS JAAS Provider Admintool will also update `system-jazn-data.xml` settings for you.

Note: If a different application needs the same login module, you must repackage the login module and any relevant classes with the new application.

See Also:

- "Configuring the Custom Security Provider in Application Server Control" on page 8-6
- "Configuring Login Modules through the Admintool" on page 8-11

Deploying Login Modules as Optional Packages

If you deploy your login modules as an *optional package* (formerly known as a "standard extension"), the OracleAS JAAS Provider will be able to find them. No additional configuration is necessary. Deploying login modules as an optional package allows multiple applications to share them.

There are two ways to use the optional package mechanism:

- Use the login module classes as an *installed optional package*. Place the login module JAR file in `jre/lib/ext` directory. Classes in JAR files in this directory can be used by applications without having to be included in the classpath.
- Use the login module classes as a *download optional package*. Specify the login module JAR file in the `Class-Path` header field in the manifest of other JAR files, as desired. In this way, classes in the login module JAR file can be used by classes in the other JAR files that reference it.

The login modules must also be configured in `system-jazn-data.xml`, as discussed in "Login Module Settings in system-jazn-data.xml" on page 8-12.

See Also:

- For general information about the standard "optional package" mechanism:

  http://java.sun.com/j2se/1.4.2/docs/guide/extensions

Using Login Modules as OC4J Shared Libraries

The OracleAS JAAS Provider is integrated with the OC4J class loading architecture. Because of this, you can make login modules available to applications by loading them as OC4J shared libraries. There are two main steps to this (considering functionality of the Application Server Control Console in particular):

1. Load the library as an OC4J shared library. From the Application Server Control Console **Administration** tab for the OC4J instance, use the Shared Libraries task.

   This results in configuration such as the following in the OC4J `server.xml` file:

```
<application-server ... >
   ...
   <shared-library name="mylib.lib" version="1.0" library-compatible="true">
      <code-source path="../mypath" />
   </shared-library>
   ...
</application-server>
```

2. Import the library into your application. In deploying an application through Application Server Control, when you reach the Deploy: Deployment Settings page (as discussed in "Deploying an Application through Application Server Control" on page 5-11), you have the opportunity to import shared libraries.

   This results in configuration such as the following in your application `orion-application.xml` file:

```
<orion-application ... >
   ...
   <imported-shared-libraries>
      <import-shared-library name="mylib.lib" />
      ...
   </imported-shared-libraries>
   ...
</orion-application>
```

Note: The `<library>` element and *ORACLE_HOME*`/j2ee/home/applib` location are still supported for OC4J shared libraries, but are discouraged.

See Also:

- *Oracle Containers for J2EE Developer's Guide* for more information about OC4J class loading and shared libraries.

Configuring the Custom Security Provider in Application Server Control

This section discusses the following administration tasks for login modules using the Application Server Control Console:

- Specifying and Configuring a Custom Security Provider during Deployment

- Changing to a Custom Security Provider after Deployment
- Adding a Login Module to the Custom Security Provider
- Updating a Login Module in the Custom Security Provider
- Deleting a Login Module in the Custom Security Provider

> **Note:** Procedures discussed throughout this section assume you are logged in to Application Server Control as a user with required administrative permissions (as `oc4jadmin`, for example).

Specifying and Configuring a Custom Security Provider during Deployment

When you plan to use the custom security provider and you deploy an application through Application Server Control, you have the opportunity to configure your custom login modules during deployment.

From the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page):

1. Go to the Select Security Provider task.
2. In the resulting Deployment Settings: Select Security Provider page, choose Custom from the Security Provider dropdown list.
3. Under "Configuration of Custom Security Provider" (which appears after you choose Custom), you can edit or delete any custom login module that is found with your application, or add a new custom login module.
   - To add a new custom login module, choose **Add Login Module**. See "Adding a Custom Login Module during Deployment" on page 8-9.
   - To edit an existing custom login module, choose the Edit task for the appropriate module. See "Editing a Custom Login Module Configuration during Deployment" on page 8-8.
   - To delete an existing custom login module, choose the Delete task for the appropriate module.
4. Still in the Deployment Settings: Select Security Provider page, choose **OK** to finish the security provider selection.
5. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11.

Deploying with a custom login module results in the following automatic settings in the `orion-application.xml` file:

```
<jazn provider="XML">
   <property name="role.mapping.dynamic" value="true" />
   <property name="custom.loginmodule.provider" value="true" />
</jazn>
```

Notes:

- Grants for custom login modules, which are stored in `system-jazn-data.xml`, cannot be configured through Application Server Control Console.
- Custom login modules, as well as the file-based provider, use a setting of `provider="XML"`.
- Custom login module configuration settings are reflected under the `<jazn-loginconfig>` element in the `system-jazn-data.xml` file, as shown in "Login Module Settings in system-jazn-data.xml" on page 8-12
- The properties `role.mapping.dynamic` and `custom.loginmodule.provider` must be set to `true` for any application that uses custom login modules.

See Also:

- "Login Module Configuration in OC4J Configuration Files" on page 8-12 and "Simple Login Module J2EE Integration" on page 8-14 for information and examples regarding the resulting XML configuration for login modules

Editing a Custom Login Module Configuration during Deployment

To edit a custom login module while deploying an application using the Custom Security Provider, take the following steps, starting under "Configuration of Custom Security Provider" in the Deployment Settings: Select Security Provider page (see "Specifying and Configuring a Custom Security Provider during Deployment" on page 8-7 for how to get to this point):

1. Choose the Edit task for the appropriate login module in the list of login module classes.
2. In the Deployment Settings: Select Security Provider: Edit Login Module page:
   - Specify a value for the login module control flag (from the dropdown list): Required, Requisite, Optional, or Sufficient. Table 8–2 describes these settings.
   - As desired, choose **Add Another Row** to create properties.
   - As desired, edit the name or value of any property in the Properties list.
   - As desired, use the Delete task for a property to remove that property.
   - Choose **Continue** to go back to the Deployment Settings: Select Security Provider page to continue the deployment steps in "Specifying and Configuring a Custom Security Provider during Deployment" on page 8-7.

Table 8–2 Login Module Control Flags

Flag	Meaning
`Required`	The login module must succeed. Whether or not it succeeds, authentication proceeds down the login module list.
`Requisite`	The login module must succeed. If it succeeds, authentication continues down the login module list. If it fails, control immediately returns to the application (authentication does not continue down the login module list).

Table 8–2 (Cont.) Login Module Control Flags

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

Adding a Custom Login Module during Deployment

To add a custom login module while deploying an application using the Custom Security Provider, take the following steps, starting under "Configuration of Custom Security Provider" in the Deployment Settings: Select Security Provider page (see "Specifying and Configuring a Custom Security Provider during Deployment" on page 8-7 for how to get to this point):

1. Choose **Add Login Module**.
2. In the Deployment Settings: Select Security Provider: Add Login Module page:
   - Specify your login module package and class name.
   - Specify a value for the login module control flag (from the dropdown list): Required, Requisite, Optional, or Sufficient. See the preceding section, "Editing a Custom Login Module Configuration during Deployment", for information about these settings.
   - As desired, choose **Add Another Row** to create properties.
   - As desired, edit the name or value of any property in the Properties list.
   - As desired, use the Delete task for a property to remove that property.
   - Choose **Continue** to go back to the Deployment Settings: Select Security Provider page to continue the deployment steps in "Specifying and Configuring a Custom Security Provider during Deployment" on page 8-7.

Changing to a Custom Security Provider after Deployment

You can select a security provider for your application at deployment time, as described above. You can also change to a different security provider after deployment. You can change to a custom security provider as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, choose "Change Security Provider".
3. In the Change Security Provider page, select "Custom Security Provider" from the dropdown.
4. Under "Login Modules" (which appears after you select Custom Security Provider in the dropdown), specify the first login module to be used, as follows. Later you can go back to the Security Provider to add more login modules, as described in the next section, "Adding a Login Module to the Custom Security Provider".
   - Specify your login module package and class name.
   - Specify a value for the login module control flag (from the dropdown list)—Required, Requisite, Optional, or Sufficient. See "Editing a Custom

Login Module Configuration during Deployment" on page 8-8 for information about these settings.

- As desired, choose **Add Another Row** to create properties.
- As desired, edit the name or value of any property in the Properties list.
- As desired, use the Delete task for a property to remove that property.
- Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you can examine the settings.

Adding a Login Module to the Custom Security Provider

Once you have established a custom security provider, either during or after deployment, you can add custom login modules as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, under "Login Modules", choose **Create**.
3. In the Add Login Module page:
   - Specify your login module package and class name.
   - Specify a value for the login module control flag (from the dropdown list): Required, Requisite, Optional, or Sufficient. See "Editing a Custom Login Module Configuration during Deployment" on page 8-8 for information about these settings.
   - As desired, choose **Add Another Row** to create properties.
   - As desired, edit the name or value of any property in the Properties list.
   - As desired, use the Delete task for a property to remove that property.
   - Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you can examine the settings.

Updating a Login Module in the Custom Security Provider

Once you have established a custom security provider, either during or after deployment, you can update custom login modules as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, in the list of login module classes, choose the Edit task for the login module you want to update.
3. In the Edit Login Module page:
   - As desired, update the value for the login module control flag (from the dropdown list): Required, Requisite, Optional, or Sufficient. See "Editing a Custom Login Module Configuration during Deployment" on page 8-8 for information about these settings.
   - As desired, choose **Add Another Row** to create properties.
   - As desired, edit the name or value of any property in the Properties list.
   - As desired, use the Delete task for a property to remove that property.
   - Choose **Apply** to finish the change.

This leaves you in the Edit Login Module page. You can use the breadcrumbs at the top of the page to go back to the Security Provider page.

Deleting a Login Module in the Custom Security Provider

Once you have established a custom security provider, either during or after deployment, you can delete custom login modules as follows:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, in the list of login module classes, choose the Delete task for the login module you want to delete.
3. Choose **Yes** in the Confirmation page.

This takes you back to the Security Provider page, where you can see that the login module was deleted.

Configuring Login Modules through the Admintool

Although Application Server Control is the preferred and recommended tool for adding and configuring custom login modules, it is also possible to use the OracleAS JAAS Provider Admintool. The following information is presented for reference:

- `-addloginmodule`: Configures a new login module for the named application. This includes specifying a control flag: one of `required`, `requisite`, `sufficient` or `optional`, as specified by `javax.security.auth.login.Configuration` and in "Editing a Custom Login Module Configuration during Deployment" on page 8-8.

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

  For example, to add `MyLoginModule`, which we will assume supports a `debug` option, to the application `myapp` as a required module with `debug` set to `true`:

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

- `-remloginmodule`: Removes the specified login module.

  To remove `MyLoginModule` from `myapp`:

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

- `-listloginmodules`: Displays all login modules for a specified application, or, if no application name is specified, displays login modules for all applications. Specifying a login module class after the application name displays information on only the specified class within the application.

  For example, to display all login modules for the application `myapp`:

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

You can also execute these commands from the Admintool shell.

Also note that to access an EJB using a custom login module, you must grant "login" permission to the user (`JDOE_ENDUSER`, for example). To grant login permission to the user through the Admintool:

```
% java -jar jazn.jar -grantperm login -user JDOE_ENDUSER com.evermind.server.rmi.RMIPermission
```

Important: Restart OC4J for changes to take effect.

See Also:

- "Adding and Removing Login Modules" on page C-8
- "Listing Login Modules" on page C-12

Login Module Configuration in OC4J Configuration Files

This section discusses files that contain configuration for custom login modules:

- Login Module Settings in system-jazn-data.xml
- Login Modules Settings in orion-application.xml
- Configuring oc4j-ra.xml for Login Modules (J2EE Connector Architecture)

Login Module Settings in system-jazn-data.xml

The `system-jazn-data.xml` file is the repository for login module configuration.

Note that settings in `system-jazn-data.xml` are updated automatically when you administer login modules through Application Server Control or the OracleAS JAAS Provider Admintool.

Note: Where there are multiple OC4J instances, login module configuration is added to the instance-specific `system-jazn-data.xml` file, not *ORACLE_HOME*`/j2ee/home/system-jazn-data.xml`.

The `<jazn-loginconfig>` element contains information that associates applications with login modules.

If this information is in the `orion-application.xml` file during deployment, as discussed in "Settings in <jazn-loginconfig> in orion-application.xml" on page 8-13, the `system-jazn-data.xml` file will be updated accordingly.

Example 8–1 Example jazn-loginconfig element

```
<jazn-loginconfig>
  <application>
    <name>sampleLM</name>
    <login-modules>
      <login-module>
        <class>oracle.security.jazn.samples.SampleLoginModule</class>
        <control-flag>required</control-flag>
        <options>
          <option>
            <name>debug</name>
            <value>true</value>
          </option>
        </options>
      </login-module>
    </login-modules>
  </application>
</jazn-loginconfig>
```

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

> **Note:** Do not remove login configuration information for `RealmLoginModule` from the `system-jazn-data.xml` file.

See Also:

- "The system-application.xml File" on page 3-6
- "Configuring the Custom Security Provider in Application Server Control" on page 8-6
- "Configuring Login Modules through the Admintool" on page 8-11

Login Modules Settings in orion-application.xml

This section discusses particular settings for login modules in the OC4J application-level descriptor `orion-application.xml`. The following topics are covered:

- Settings in <jazn-loginconfig> in orion-application.xml
- Settings in <jazn> for Login Modules
- Settings in <namespace-access> for Login Modules

> **Note:** This section discusses only elements relevant to security. For a full discussion of this file, see the *Oracle Containers for J2EE Developer's Guide*.

Settings in <jazn-loginconfig> in orion-application.xml

Settings in the `<jazn-loginconfig>` element in `system-jazn-data.xml` were documented earlier, in "Login Module Settings in system-jazn-data.xml" on page 8-12. You can add this element to `orion-application.xml` for deployment, and the settings will be written to the `system-jazn-data.xml` file automatically.

In addition, when you undeploy the application, the `<jazn-loginconfig>` settings will be removed from `system-jazn-data.xml` automatically.

Settings in <jazn> for Login Modules

The following `<jazn>` properties are specific to login module configuration:

- `role.mapping.dynamic`

  This property, when set to `true`, instructs the OracleAS JAAS Provider to base authorization checks on the authenticated subject instead of basing checks on the users and roles defined in `system-jazn-data.xml` or the application-specific `jazn-data.xml` file.

  A `LoginModule` instance must ensure that the appropriate principals (users or roles) 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.

- `custom.loginmodule.provider`

  This property, when set to `true`, instructs Application Server Control that the security provider is the custom provider. Without this setting, because the custom security provider uses the setting `provider="XML"`, Application Server Control would mistakenly report that the security provider is the file-based provider (although custom login modules you provide in your EAR file would still work).

These properties are automatically set to "`true`" in `orion-application.xml`, as shown in the following example, when you have a `<jazn-loginconfig>` element in `orion-application.xml`.

```
<jazn provider="XML" ... >
   <property name="role.mapping.dynamic" value="true" />
   <property name="custom.loginmodule.provider" value="true" />
</jazn>
```

Settings in <namespace-access> for Login Modules

To access an EJB using a custom login module, you must also grant namespace access to the user in `orion-application.xml`, as in the following example for `JDOE_ENDUSER`:

```
<namespace-access>
  <read-access>
    <namespace-resource root="">
      <security-role-mapping>
        <user name="JDOE_ENDUSER" />
      </security-role-mapping>
    </namespace-resource>
  </read-access>
</namespace-access>
```

Configuring oc4j-ra.xml for Login Modules (J2EE Connector Architecture)

When you configure resource adapters, each `<connector-factory>` element in the `oc4j-ra.xml` file can specify a different JAAS login module, as in the following example. This also shows `<config-property>` setup to connect to a database through Oracle JDBC.

```
  <connector-factory connector-name="myBlackbox" location="eis/myEIS1">
     <config-property name="connectionURL"
                      value="jdbc:oracle:thin:@localhost:5521/myservice" />
     <security-config use="jaas-module">
        <jaas-module>
           <jaas-application-name>JAASModuleDemo</jaas-application-name>
        </jaas-module>
     </security-config>
  </connector-factory>
```

Simple Login Module J2EE Integration

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

Development of Simple Login Module

Develop a JAAS-compliant `LoginModule` implementation according to the JAAS service provider interface.

See Also:

- `javax.security.auth.spi.LoginModule` Javadoc:

  `http://java.sun.com/j2se/1.4.2/docs/api/`

- "Custom Login Module Example" on page 8-16

Packaging of Simple Login Module

Package your login module classes in one of the ways described in "Packaging and Deploying Login Modules" on page 8-4.

Deployment of Simple Login Module

Configuration for login modules is specified in the `system-jazn-data.xml` file:

1. Register your application login module. This occurs automatically when you deploy an application through Application Server Control.

   The following fragment from `system-jazn-data.xml` shows the registration of the login module `oracle.security.jazn.samples.SampleLoginModule`, to be used for authenticating users accessing the `sampleLM` application.

```
<jazn-loginconfig>
   <application>
       <name>sampleLM</name>
       <login-modules>
          <login-module>
             <class>oracle.security.jazn.samples.SampleLoginModule</class>
             <control-flag>required</control-flag>
             <options>
                <option>
                   <name>debug</name>
                   <value>true</value>
                </option>
             </options>
          </login-module>
       </login-modules>
    </application>
    ...
 </jazn-loginconfig>
```

2. **Optional**. Grant relevant permissions to your users and roles. You can use the OracleAS JAAS Provider Admintool to accomplish this. For example, if the principal `developer` needs EJB access, then you must grant the permission `com.evermind.server.rmi.RMIPermission` to `developer`.

   This results in configuration such as the following in `system-jazn-data.xml`:

```
<jazn-policy>
  <grant>
    <grantee>
      <principals>
        <principal>
          <class>oracle.security.jazn.samples.SampleUser</class>
          <name>developer</name>
        </principal>
      </principals>
    </grantee>
    <permissions>
      <permission>
```

```
        <class>com.evermind.server.rmi.RMIPermission</class>
        <name>login</name>
      </permission>
    </permissions>
  </grant>
  ...
</jazn-policy>
```

To deploy your login module, ensure the following settings are in the `orion-application.xml` file:

1. The `<jazn>` properties `role.mapping.dynamic` and `custom.loginmodule.provider` (described in "Settings in <jazn> for Login Modules" on page 8-13) are set to "`true`":

   ```
   <jazn provider="XML" ... >
      <property name="role.mapping.dynamic" value="true" />
      <property name="custom.loginmodule.provider" value="true" />
   </jazn>
   ```

   This occurs automatically if there is a `<jazn-loginconfig>` element in `orion-application.xml`.

2. Appropriate `<security-role-mapping>` entries exist:

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

   You can set these mappings through Application Server Control.

Custom Login Module Example

This section gives source code for a simple custom login module to be used by the `CallerInfo` example.

Example 8–2 SampleLoginModule.java

```
package oracle.security.jazn.samples;

import java.util.Set;
import java.util.Iterator;
import java.util.Map;
import java.security.Principal;
import javax.security.auth.Subject;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.login.LoginException;
import javax.security.auth.spi.LoginModule;

public class SampleLoginModule implements LoginModule {

    // initial state
    protected Subject _subject;
    protected CallbackHandler _callbackHandler;
    protected Map _sharedState;
```

```
protected Map _options;

// configuration options
protected boolean _debug;

// the authentication status
protected boolean _succeeded;
protected boolean _commitSucceeded;

// username and password
protected String _name;
protected char[] _password;

protected Principal[] _authPrincipals;


/**
 * Initialize this <code>LoginModule</code>.
 * <p/>
 * <p/>
 *
 * @param subject         the <code>Subject</code> to be authenticated. <p>
 * @param callbackHandler a <code>CallbackHandler</code> for communicating
 *                        with the end user (prompting for usernames and
 *                        passwords, for example). <p>
 * @param sharedState     shared <code>LoginModule</code> state. <p>
 * @param options         options specified in the login
 *                        <code>Configuration</code> for this particular
 *                        <code>LoginModule</code>.
 */
public void initialize(Subject subject,
                       CallbackHandler callbackHandler,
                       Map sharedState,
                       Map options) {
    this._subject = subject;
    this._callbackHandler = callbackHandler;
    this._sharedState = sharedState;
    this._options = options;

    // initialize any configured options
    _debug = "true".equalsIgnoreCase((String) _options.get("debug"));

    if (debug()) {
        printConfiguration(this);
    }
}


final public boolean debug() {
    return _debug;
}


protected Principal[] getAuthPrincipals() {
    return _authPrincipals;
}


/**
 * Authenticate the user by prompting for a username and password.
```

```
     * <p/>
     * <p/>
     *
     * @return true if the authentication succeeded, or false if this
     *         <code>LoginModule</code> should be ignored.
     * @throws FailedLoginException if the authentication fails. <p>
     * @throws LoginException       if this <code>LoginModule</code>
     *                              is unable to perform the authentication.
     */
    public boolean login() throws LoginException {
        if (debug())
            System.out.println("\t\t[SampleLoginModule] login");

        if (_callbackHandler == null)
            throw new LoginException("Error: no CallbackHandler available " +
                    "to garner authentication information from the user");

        // Setup default callback handlers.
        Callback[] callbacks = new Callback[] {
            new NameCallback("Username: "),
            new PasswordCallback("Password: ", false)
        };

        try {
            _callbackHandler.handle(callbacks);
        } catch (Exception e) {
            _succeeded = false;
            throw new LoginException(e.getMessage());
        }


        String username = ((NameCallback)callbacks[0]).getName();
        String password = new
                          String(((PasswordCallback)callbacks[1]).getPassword());
        if (debug())
        {
            System.out.println("\t\t[SampleLoginModule] username : " + username);
        }

        // Authenticate the user. On successfull authentication add principals
        // to the Subject. The name of the principal is used for authorization by
        // OC4J by mapping it to the value of the name attribute of the group
        // tag in the security-role-mapping for the application.
        if(username.equals("developer") && password.equals("welcome"))
        {
            _succeeded = true;
            _name = "developer";
            _password = password.toCharArray();
            _authPrincipals = new SamplePrincipal[2];
            //Adding username as principal to the subject
            _authPrincipals[0] = new SamplePrincipal("developer");
            //Adding role developers to the subject
            _authPrincipals[1] = new SamplePrincipal("developers");
        }

        if(username.equals("manager") && password.equals("welcome"))
        {
            _succeeded = true;
            _name = "manager";
            _password = password.toCharArray();
```

```
        _authPrincipals = new SamplePrincipal[3];
        //Adding username as principal to the subject
        _authPrincipals[0] = new SamplePrincipal("manager");
        //Adding roles developers and managers to the subject
        _authPrincipals[1] = new SamplePrincipal("developers");
        _authPrincipals[2] = new SamplePrincipal("managers");
    }


    ((PasswordCallback)callbacks[1]).clearPassword();
    callbacks[0] = null;
    callbacks[1] = null;

    if (debug())
    {
        System.out.println("\t\t[SampleLoginModule] success : " + _succeeded);
    }

    if (!_succeeded)
        throw new LoginException
                        ("Authentication failed: Password does not match");

    return true;
}


/**
 * <p> This method is called if the LoginContext's
 * overall authentication succeeded
 * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
 * succeeded).
 * <p/>
 * <p> If this LoginModule's own authentication attempt
 * succeeded (checked by retrieving the private state saved by the
 * <code>login</code> method), then this method associates a
 * <code>Principal</code>
 * with the <code>Subject</code> located in the
 * <code>LoginModule</code>.  If this LoginModule's own
 * authentication attempted failed, then this method removes
 * any state that was originally saved.
 * <p/>
 * <p/>
 *
 * @return true if this LoginModule's own login and commit
 *         attempts succeeded, or false otherwise.
 * @throws LoginException if the commit fails.
 */
public boolean commit()
        throws LoginException {
    try {

        if (_succeeded == false) {
            return false;
        }

        if (_subject.isReadOnly()) {
            throw new LoginException("Subject is ReadOnly");
        }

        // add authenticated principals to the Subject
```

```
            if (getAuthPrincipals() != null) {
                for (int i = 0; i < getAuthPrincipals().length; i++) {
                    if(!_subject.getPrincipals().contains(getAuthPrincipals()[i]))
{
                        _subject.getPrincipals().add(getAuthPrincipals()[i]);
                    }
                }
            }

            // in any case, clean out state
            cleanup();
            if (debug()) {
                printSubject(_subject);
            }

            _commitSucceeded = true;
            return true;

        } catch (Throwable t) {
            if (debug()) {
                System.out.println(t.getMessage());
                t.printStackTrace();
            }
            throw new LoginException(t.toString());
        }
    }


    /**
     * <p> This method is called if the LoginContext's
     * overall authentication failed.
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * did not succeed).
     * <p/>
     * <p> If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * <code>login</code> and <code>commit</code> methods),
     * then this method cleans up any state that was originally saved.
     * <p/>
     * <p/>
     *
     * @return false if this LoginModule's own login and/or commit attempts
     *         failed, and true otherwise.
     * @throws LoginException if the abort fails.
     */
    public boolean abort() throws LoginException {
        if (debug()) {
            System.out.println
                   ("\t\t[SampleLoginModule] aborted authentication attempt.");
        }

        if (_succeeded == false) {
            cleanup();
            return false;
        } else if (_succeeded == true && _commitSucceeded == false) {
            // login succeeded but overall authentication failed
            _succeeded = false;
            cleanup();
        } else {
            // overall authentication succeeded and commit succeeded,
```

```
        // but someone else's commit failed
        logout();
    }
    return true;
}


protected void cleanup() {
    _name = null;
    if (_password != null) {
        for (int i = 0; i < _password.length; i++) {
            _password[i] = ' ';
        }
        _password = null;
    }
}


protected void cleanupAll() {
    cleanup();

    if (getAuthPrincipals() != null) {
        for (int i = 0; i < getAuthPrincipals().length; i++) {
            _subject.getPrincipals().remove(getAuthPrincipals()[i]);
        }
    }
}


/**
 * Logout the user.
 * <p/>
 * <p> This method removes the <code>Principal</code>
 * that was added by the <code>commit</code> method.
 * <p/>
 * <p/>
 *
 * @return true in all cases since this <code>LoginModule</code>
 *         should not be ignored.
 * @throws LoginException if the logout fails.
 */
public boolean logout() throws LoginException {
    _succeeded = false;
    _commitSucceeded = false;
    cleanupAll();
    return true;
}

// helper methods //

protected static void printConfiguration(SampleLoginModule slm) {
    if (slm == null) {
        return;
    }
    System.out.println("\t\t[SampleLoginModule] configuration options:");
    if (slm.debug()) {
        System.out.println("\t\t\tdebug = " + slm.debug());
    }
}
```

```
    protected static void printSet(Set s) {
        try {
            Iterator principalIterator = s.iterator();
            while (principalIterator.hasNext()) {
                Principal p = (Principal) principalIterator.next();
                System.out.println("\t\t\t" + p.toString());
            }
        } catch (Throwable t) {
        }
    }


    protected static void printSubject(Subject subject) {
        try {
            if (subject == null) {
                return;
            }
            Set s = subject.getPrincipals();
            if ((s != null) && (s.size() != 0)) {
                System.out.println
                      ("\t\t[SampleLoginModule] added the following Principals:");
                printSet(s);
            }

            s = subject.getPublicCredentials();
            if ((s != null) && (s.size() != 0)) {
                System.out.println
             ("\t\t[SampleLoginModule] added the following Public Credentials:");
                printSet(s);
            }
        } catch (Throwable t) {
        }
    }
}
```

The `Principal` that this `LoginModule` uses is in Example 8–3.

Example 8–3 SamplePrincipal example

```
package oracle.security.jazn.samples;

import java.security.Principal;

public class SamplePrincipal implements Principal {

    private String _name = null;


    public SamplePrincipal(String name) {
        _name = name;
    }


    public boolean equals(Object another) {
        return ((SamplePrincipal)another).getName().equals(_name);
    }


    public String getName() {
        return _name;
```

```
    }


    public int hashCode() {
        return _name.hashCode();
    }


    public String toString() {
        return "[SamplePrincipal] : " + _name;
    }
}
```

9

External LDAP Security Providers

This chapter discusses how to configure OC4J to use a non-Oracle ("third-party" or "external") LDAP server as the user repository. It is divided into the following sections:

- Overview of External LDAP Provider Configuration and Administration
- Configuring External LDAP Providers in Application Server Control
- External LDAP Provider Settings in system-jazn-data.xml
- Granting RMI Permission to an LDAP Principal
- Sample Configuration for Sun Java System Directory Server

The OC4J 10.1.3 implementation supports the following external LDAP providers:

- Active Directory (for Windows Server 2003)
- Sun Java System Directory Server (version 5.2)

Notes:

- Support for external LDAP providers requires JDK 1.4 or later.
- The concept of security realms is not supported when using external LDAP providers.

See Also:

- Chapter 8, "Login Modules"

Overview of External LDAP Provider Configuration and Administration

When you deploy an application using Application Server Control Console, you have the opportunity to specify an external (third-party) LDAP provider, as noted in "Specifying the Security Provider through Application Server Control" on page 5-13.

(This assumes that you have already completed the prerequisite of installing and configuring Sun Java System Directory Server, formerly iPlanet, or Active Directory.)

Specifying an external LDAP provider automatically results in the following setting in `orion-application.xml`:

```
<jazn provider="XML">
   <property name="custom.ldap.provider" value="true" />
</jazn>
```

Notes:

- Note the setting `provider="XML"` is used for external LDAP providers as well as for the file-based provider.
- Be aware that when you use an external LDAP provider, role comparisons for authorization are *not* case-sensitive unless you add the following property setting to the `<jazn>` element in `orion-application.xml`:

```
<property name="role.compare.ignorecase" value="false" />
```

Troubleshooting Tips: Note the following potential issues if you have trouble using an external LDAP provider:

- Be sure you are using the Distinguished Name (DN) of the LDAP user to connect to the LDAP server. This user must be an administrator with privileges to search users and groups.
- If you provide the correct user name and password for login, but still get an authentication failure for invalid credentials, ensure that the LDAP host and port are configured correctly. Using the `ldapbind` command to bind against the configured LDAP host and port will be a good way to check.

OC4J provides a login module, `LDAPLoginModule`, to use for authentication and authorization with an external LDAP provider. (Alternatively, you can provide a custom login module to use with any custom repository.) Configurable options for an external LDAP provider include the following:

- URL of the external LDAP provider
- LDAP principal DN to connect (user must have privileges to query role information for any user in the LDAP directory)
- Credential of the LDAP principal DN
- LDAP attribute that uniquely identifies a user
- User object classes, search bases, search scope
- Role object classes, search bases, search scope
- Enabling or disabling of connection pooling
- Enabling or disabling of login module caching

Option settings are reflected within a `<login-module>` element in `system-jazn-data.xml`, which configures `LDAPLoginModule`.

Note: Sample login module entries for Sun Java System Directory Server and Microsoft Active Directory are provided in the directory *ORACLE_HOME*`/j2ee/home/jazn/config`. A non-provider-specific login module entry is provided in the file `ldap_login_module.template` in the *ORACLE_HOME*`/j2ee/home/jazn/config` directory.

Configuring External LDAP Providers in Application Server Control

This section discusses the following topics for administering external LDAP providers using the Application Server Control Console:

- Specifying and Configuring an External LDAP Provider during Deployment
- Changing to an External LDAP Provider after Deployment

> **Note:** Procedures discussed throughout this section assume you are logged in to Application Server Control as a user with required administrative permissions (as `oc4jadmin`, for example).

Specifying and Configuring an External LDAP Provider during Deployment

When you plan to use an external LDAP provider and deploy an application through Application Server Control, you have the opportunity to configure the external LDAP provider when you specify it as the security provider.

From the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 5-11 for how to get to this page):

1. Go to the Select Security Provider task.
2. In the resulting Deployment Settings: Select Security Provider page, choose Third Party LDAP Server from the Security Provider dropdown list.
3. Under "Configuration of Oracle Security Provider for 3rd Party LDAP Server" (which appears after you choose Third Party LDAP Server), specify settings for the options documented in:
   - Table 9–1, " Application Server Control External LDAP Provider Options"
   - Table 9–2, " Application Server Control External LDAP Connection Pool Options" (if you enable connection pooling)
   - Table 9–3, " Application Server Control External LDAP User Options"
   - Table 9–4, " Application Server Control External LDAP Role and Member Options"

   Or, alternatively, choose **Set Values to Vendor Defaults**.
4. Choose **OK** to finish the security provider selection.
5. Back in the Deploy: Deployment Settings page, choose **Deploy** to complete the deployment, or choose another task, as desired. The list of tasks is noted in "Deploying an Application through Application Server Control" on page 5-11.

Table 9–1 Application Server Control External LDAP Provider Options

Option	Required? Or Settings / Default	Equivalent Option in Table 9–5 (see for description)
LDAP Location	Required	oracle.security.jaas.ldap.provider.url
LDAP Directory Vendor	Active Directory, Sun Directory Server, or Other (from dropdown menu)	oracle.security.jaas.ldap.provider.type
User DN	Required	oracle.security.jaas.ldap.provider.principal

Table 9–1 (Cont.) Application Server Control External LDAP Provider Options

Option	Required? Or Settings / Default	Equivalent Option in Table 9–5 (see for description)
User Password	Required	oracle.security.jaas.ldap.provider.credential
Enable Caching (checkbox)	Default: `true`	oracle.security.jaas.ldap.lm.cache_enabled
Enable Connection Pooling (checkbox)	Default: `true`	oracle.security.jaas.ldap.provider.connect.pool

Table 9–2 Application Server Control External LDAP Connection Pool Options

Option	Default	Description
Initial Size of Connection Pool	2	Number of connections initially created in the pool for each connection identity
Maximum Size of Connection Pool	25	Maximum number of connections that can be concurrently maintained in the pool for each connection identity
Preferred Size of Connection Pool	10	Preferred number of connections in the pool for each connection identity
Idle Connection Timeout (milliseconds)	300000 (5 minutes)	The amount of time that an idle connection can remain in the pool before being removed

Note: The above connection pooling properties correspond to the following:

```
com.sun.jndi.ldap.connect.pool.initsize
com.sun.jndi.ldap.connect.pool.maxsize
com.sun.jndi.ldap.connect.pool.prefsize
com.sun.jndi.ldap.connect.pool.timeout
```

As described at:

http://java.sun.com/products/jndi/tutorial/ldap/connect/config.html

Table 9–3 Application Server Control External LDAP User Options

Option	Required? Or Settings / Default	Equivalent Option in Table 9–6 (see for description)
User Search Base	Required	oracle.security.jaas.ldap.user.searchbase
User Search Scope	Subtree (default) or One Level (from dropdown menu) **Note**: Although the default in the dropdown menu is Subtree, the vendor default is One Level.	oracle.security.jaas.ldap.user.searchscope
LDAP User Name Attribute	Required	oracle.security.jaas.ldap.user.name.attribute
LDAP User Object Class	Required	oracle.security.jaas.ldap.user.object.class

Table 9–4 Application Server Control External LDAP Role and Member Options

Option	Required? Or Settings / Default	Equivalent Option in Table 9–7 (see for description)
Group Search Base	Required	oracle.security.jaas.ldap.role.searchbase
Group Search Scope	Subtree (default) or One Level (from dropdown menu) **Note**: Although the default in the dropdown menu is Subtree, the vendor default is One Level.	oracle.security.jaas.ldap.role.searchscope
LDAP Group Name Attribute	Required	oracle.security.jaas.ldap.role.name.attribute
LDAP Group Object Class	Required	oracle.security.jaas.ldap.role.object.class
LDAP Group Member Attribute	Required	oracle.security.jaas.ldap.member.attribute
Group Membership Scope Search	Direct (default) or Nested (from dropdown menu)	oracle.security.jaas.ldap.membership.searchscope

Changing to an External LDAP Provider after Deployment

You can select a security provider for your application at deployment time, as described above. You can also change to a different security provider after deployment. In particular, to change to an external LDAP provider:

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.
2. In the Security Provider page, choose "Change Security Provider".
3. In the Change Security Provider page, select Oracle Security Provider for 3rd Party LDAP Server from the Security Provider Type dropdown.
4. Under "Security Provider Attributes: Oracle Security Provider for 3rd Party LDAP Server" (which appears after you select 3rd Party LDAP Server in the dropdown), specify settings for the options documented in:
   - Table 9–1, " Application Server Control External LDAP Provider Options"
   - Table 9–2, " Application Server Control External LDAP Connection Pool Options" (if you enable connection pooling)
   - Table 9–3, " Application Server Control External LDAP User Options"
   - Table 9–4, " Application Server Control External LDAP Role and Member Options"

   Or, alternatively, choose **Set Values to Vendor Defaults**.
5. Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you can examine the settings.

External LDAP Provider Settings in system-jazn-data.xml

Configuration of an external LDAP provider is reflected in a `<login-module>` element in `system-jazn-data.xml` that configures the `LDAPLoginModule`, the login module used for external LDAP providers in OracleAS JAAS Provider. Any `<login-module>` elements are subelements of the `<login-modules>` element under `<jazn-loginconfig>`.

Each option in a `<login-module>` element is represented by the `<name>` subelement of an `<option>` element and corresponds to a configuration setting in the external LDAP provider.

You can specify settings of these options through Application Server Control, as documented in "Specifying and Configuring an External LDAP Provider during Deployment" on page 9-3, which also documents the correspondence between options listed in this section and what you see in the Application Server Control Console.

Supported options are listed in Table 9–5, Table 9–6 , and Table 9–7. Where applicable, the tables indicate default values that are used when you configure an external LDAP provider through Application Server Control and choose **Set Values to Vendor Defaults**. Except where noted otherwise, these options are required, either by specifying them directly or using vendor defaults.

Note: The `<jazn-loginconfig>` element can also appear in the `orion-application.xml` file, in which case it is copied from there into the `system-jazn-data.xml` file.

See Also:

- "Settings in system-jazn-data.xml for Sun Java System Directory Server" on page 9-9 for examples of some option settings

Table 9–5 External LDAP Provider Options

Option Name	Meaning
oracle.security.jaas.ldap.provider.url	The URL of the LDAP provider, in the format `ldap://host:port`, such as: `ldap://myhost.example.com:389`
oracle.security.jaas.ldap.provider.principal	The Distinguished Name (DN) of the LDAP user that is used to connect to the LDAP server. This user must be an administrator with privileges to search users and roles, and to invoke `ldapcompare` on a user password if the target directory supports that functionality.
oracle.security.jaas.ldap.provider.credential	The credential (generally a password) used to authenticate the LDAP user defined in: `oracle.security.jaas.ldap.provider.principal`
oracle.security.jaas.ldap.provider.type	(Optional) The product name of the LDAP provider. Supported values are `sun directory server`, `active directory`, and `other`. If you supply `sun directory server` or `active directory`, the login module is able to infer some LDAP properties and do some optimizations.
oracle.security.jaas.ldap.provider.connect.pool	(Optional) Boolean indicating whether connection pooling is enabled. A `true` setting (default) enables connection pooling; `false` disables it.
oracle.security.jaas.ldap.lm.cache_enabled	(Optional) Boolean indicating whether login module caching is enabled. A `true` setting (default) enables caching, `false` disables it.

Table 9–6 External LDAP User Options

Option Name	Meaning
oracle.security.jaas.ldap.user.name.attribute	The name of the LDAP attribute that uniquely identifies the name of the user. The default for Sun Java System Directory Server is `uid`; for Active Directory, it is `sAMAccountName`.
oracle.security.jaas.ldap.user.object.class	A list of one or more space-delimited LDAP schema object classes to represent a user. The default for either Sun Java System Directory Server or Active Directory is `inetOrgPerson`.
oracle.security.jaas.ldap.user.searchbase	A list of space-delimited distinguished names (DNs) in the LDAP directory that contains users. Here is a sample DN: `cn=users,dc=us,dc=abc,dc=com`
oracle.security.jaas.ldap.user.searchscope	Specifies how deep in the LDAP directory tree to search for users. Supported values are `subtree` or `onelevel` (default).

Table 9–7 External LDAP Role and Member Options

Option Name	Meaning
oracle.security.jaas.ldap.role.name.attribute	The name of the LDAP attribute that uniquely identifies the name of the role. For either Sun Java System Directory Server or Active Directory, the default is "`cn`".
oracle.security.jaas.ldap.role.object.class	A list of one or more space-delimited LDAP schema object classes that is used to represent a role. The default for Sun Java System Directory Server is `groupOfUniqueNames`; for Active Directory, it is `group`.
oracle.security.jaas.ldap.role.searchbase	A list of space-delimited distinguished names (DN) in the LDAP directory that contains a role. For example: `cn=groups,dc=us,dc=abc,dc=com`
oracle.security.jaas.ldap.role.searchscope	Specifies how deep in the LDAP directory tree to search for roles. Supported values are `subtree` or `onelevel` (default).
oracle.security.jaas.ldap.membership.searchscope	Specifies how deep in the LDAP directory tree to search for role membership. Supported values are `direct` (default) or `nested`. A `direct` setting means the runtime will only get the roles directly assigned to the role or user in question, as opposed to nested roles within roles.
oracle.security.jaas.ldap.member.attribute	The attribute of a static LDAP role object specifying the distinguished names (DNs) of the members of the role. The default for Sun Java System Directory Server is `uniqueMember`; for Active Directory, it is `member`.

Granting RMI Permission to an LDAP Principal

When using an external LDAP provider, it may be necessary to grant RMI "login" permission for an LDAP principal.

The following example uses the OracleAS JAAS Provider Admintool to accomplish this:

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes \
       com.evermind.server.rmi.RMIPermission login
```

This example would result in the following configuration in the `system-jazn-data.xml` file:

```
<jazn-policy>
```

```
   <grant>
      <grantee>
         <principals>
            <principal>
               <class>oracle.security.jazn.realm.LDAPPrincipal</class>
               <name>hobbes</name>
            </principal>
         </principals>
      </grantee>
      ...
      <permissions>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

Sample Configuration for Sun Java System Directory Server

This section provides the following sample configuration to use the Sun Java System Directory Server as an external LDAP provider:

- Sample LDIF Description
- Sample Entries in OC4J Configuration Files

The `orion-application.xml` and `system-jazn-data.xml` settings would be made automatically if you use Application Server Control Console as described earlier in this chapter.

> **Note:** A template file containing a sample login module entry for Sun Java System Directory Server is provided in the file `sample_login_module.sun` in the *ORACLE_HOME*`/j2ee/home/jazn/config` directory. (Similarly, a template file containing a sample login module entry for Active Directory is provided in the file `sample_login_module.ad` in the *ORACLE_HOME*`/j2ee/home/jazn/config` directory.)

Sample LDIF Description

Assume the following LDIF description is used for the Sun Java System Directory Server example:

Example 9–1 Sample LDIF Defining a User and Role

```
# An example user object entry
uid= jdoe,dc=us,dc=example,dc=com
uid= jdoe
givenName=John
sn=Doe
cn=John Doe
userPassword={SSHA}zD/44JbZY33osry4mzfLn0du7nBhIIAHKDG5Fg==
uidNumber=1
```

```
gidNumber=1
homeDirectory=c:\
objectClass=top
objectClass=person
objectClass=organizationalPerson
objectClass= inetOrgPerson
objectClass=posixAccount

# An example role object entry
cn=managers,ou=groups,dc=us,dc=example,dc=com
objectClass=top
objectClass= groupOfUniqueNames
cn=managers
uniqueMember=uid=jdoe,dc=us,dc=example,dc=com
```

Sample Entries in OC4J Configuration Files

This section shows OC4J configuration in the following files for an external LDAP provider:

- Settings in system-jazn-data.xml for Sun Java System Directory Server
- Settings in orion-application.xml for External LDAP Server

Settings in system-jazn-data.xml for Sun Java System Directory Server

Assume your Sun Java System Directory Server installation is described by the set of LDIF entries shown in Example 9–1. The corresponding `<jazn-loginconfig>` entries in the `system-jazn-data.xml` file are shown in the following example:

Example 9–2 JAAS Login Module Configuration Corresponding to Example 9–1

```
<jazn-data ... >
   ...
   <jazn-loginconfig>
      <application>
         <name>callerInfo</name>
         <login-modules>
            <login-module>
               <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
               <control-flag>required</control-flag>
               <options>
                  <option>
                     <name>oracle.security.jaas.ldap.user.name.attribute</name>
                     <value>uid</value>
                  </option>
                  <option>
                     <name>oracle.security.jaas.ldap.user.object.class</name>
                     <value>inetOrgPerson</value>
                  </option>
                  <option>
                     <name>oracle.security.jaas.ldap.user.searchbase</name>
                     <value>dc=us,dc=example,dc=com</value>
                  </option>
                  <option>
                     <name>oracle.security.jaas.ldap.role.name.attribute</name>
                     <value>cn</value>
                  </option>
                  <option>
                     <name>oracle.security.jaas.ldap.role.object.class</name>
                     <value>groupOfUniqueNames</value>
```

```
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.role.searchbase</name>
                   <value>ou=groups,dc=us,dc=example,dc=com</value>
                </option>
                <option>
                   <name>oracle.security.jaas.ldap.member.attribute</name>
                   <value> uniqueMember </value>
                </option>
             </options>
          </login-module>
       </login-modules>
    </application>
  </jazn-loginconfig>
  ...
</jazn-data>
```

Note: An option setting for an external LDAP location would look like the following:

```
...
<options>
   ...
   <option>
      <name>oracle.security.jaas.ldap.provider.url</name>
      <value>ldap://myhost.example.com:389</value>
   </option>
   ...
</options>
...
```

Settings in orion-application.xml for External LDAP Server

The following settings in `orion-application.xml` are used for any external LDAP provider:

```
<jazn provider="XML">
  <property name="custom.ldap.provider" value="true" />
</jazn>
```

You must restart OC4J to synchronize the login module information from `system-jazn.data.xml`.

10

COREid Access Security Provider

This chapter discusses Oracle COREid Access and Identity and describes how to use the COREid Access security provider for authentication, authorization, and single sign-on. The chapter is useful for those who have already deployed, or plan to deploy, the Oracle COREid Access and Identity system. It describes integration between the COREid 7.0.4 implementation and the OC4J 10.1.3 implementation, and how to secure applications deployed on OC4J through features of COREid. This includes detailed configuration steps for Web applications, and examples for Web applications, EJBs, and Web Service authentication schemes (including username token, X.509 token, and SAML token).

This chapter covers the following topics:

Notes:

- The emphasis in this chapter is on what you must do from an OC4J and OracleAS JAAS Provider perspective to enable the use of Oracle COREid Access and Identity with Oracle Application Server. Some prior familiarity with Oracle COREid Access and Identity is assumed, and there is no attempt to thoroughly document the use of COREid features or tools, such as Access Manager. In terms of what to do for necessary COREid setup, the emphasis is on required settings, and how they map to OracleAS JAAS Provider configuration, rather than on how to accomplish the settings. Please consult COREid documentation for details on COREid features and administration.
- Only the 7.0.4 implementation of Oracle COREid Access and Identity is supported with the 10.1.3 implementation of OC4J.

See Also:

- *Oracle COREid Access and Identity Installation Guide*
- *Oracle COREid Access and Identity Administration Guide*
- *Oracle COREid Access and Identity Developer Guide*

These and other COREid documents are available from:

http://www.oracle.com/technology/documentation/appserver1012.html

(Scroll down the page until you see the entry for COREid documentation.)

Getting Started with Oracle COREid Access and Identity

This section provides an overview of Oracle COREid Access and Identity and discusses prerequisites and architecture:

Overview of Oracle COREid Access and Identity

Oracle COREid Access and Identity is an enterprise-class authentication, authorization, and auditing solution that provides centralized security administration. This includes functionality for access control, single sign-on (separate from OracleAS Single Sign-On), personalization, and user profile management in heterogeneous application environments across a variety of application servers, legacy applications, and databases. COREid provides key features for creating, managing, and enforcing access policies. If you have different sets of users that require access to different data sets, while all require access to a common set of data, COREid can allow the right levels of access to each group so that everyone can access only the data that is appropriate for them.

In comparing Oracle COREid Access and Identity to other authentication, single sign-on, and authorization services, note the following differentiating features.

- You can centralize authentication and authorization for multiple OC4J instances through a single Oracle COREid Access and Identity instance, allowing centralized single sign-on and auditing functionality, as well as more robust authentication options.
- COREid offers superior identity administration through workflow, fine-grained attribute control, and delegation of administration.
- COREid supports access control based on dynamic groups, with members based on a given identity profile.
- COREid allows realtime access and identity integration, with runtime changes made through COREid being automatically populated into the Access Server cache to eliminate security loopholes.

In the OC4J 10.1.3 implementation, OracleAS JAAS Provider supports Oracle COREid Access and Identity integration through a custom login module and a special authentication method setting.

Oracle COREid Access and Identity includes the following components:

- WebGate, the policy enforcer, is a Web server plug-in access client (with an associated Apache mod for use on Oracle HTTP Server) that intercepts HTTP requests and forwards them to the Access Server for authentication and authorization. In comparison, an AccessGate is a custom access client, built with the COREid Access SDK, that processes Web and non-Web resource requests from users or applications. It intercepts user requests and forwards them to the Access Server for authentication and authorization. The terms WebGate and AccessGate can be used interchangeably in most situations.
- WebPass is a Web server plug-in that passes information between a Web server and a COREid server.
- COREid Identity Server processes all user identity, group, organization, and credential-management requests.
- Access Server, the policy decision-maker, receives requests, responds to the access client, and manages the login session. The Access Server receives requests from WebGate and queries the authentication, authorization, and auditing rules in Oracle Internet Directory. The Access Server also manages the login session by helping WebGate terminate sessions, set user session timeouts, reauthenticate when timeouts occur, and track session activity.
- Access Manager writes policy data to Oracle Internet Directory, and updates the Access Server with policy modifications. It includes an Access System Console that enables administrators to manage policies and the system configuration.

Note: In the 10.1.3 implementation, Application Server Control does not support configuration of Oracle COREid Access and Identity.

COREid Prerequisites

This section describes what you must have installed to use Oracle COREid Access and Identity. Oracle COREid Access and Identity components are version 7.0.4.

See Also:

- *Oracle COREid Access and Identity Installation Guide* for installation instructions

At a high level, prerequisites include installing Oracle COREid Access and Identity and Oracle Application Server, and configuring the Access SDK and your applications on OC4J.

Detailed requirements on the COREid side:

1. A suitable LDAP repository, such as Oracle Internet Directory (included in the Oracle Application Server 10.1.2 infrastructure).
2. A Web server, such as Oracle HTTP Server (included in the Oracle Application Server 10.1.3 middle-tier infrastructure).
3. The COREid Identity Server and Access Server. When you install Oracle COREid Access and Identity, you will be asked to specify the LDAP repository you are using, which must be accessible for communication with COREid Identity Server and Access Server during runtime.
4. COREid WebGate, WebPass, and Access Manager installed on the Web server. WebGate is the SSO interceptor and communicates with Access Server during

runtime. WebPass communicates with COREid Identity Server. Access Manager communicates with the LDAP repository. When you install WebGate and WebPass, you will be asked to specify the Access Server and COREid Identity Server you are using.

Detailed requirements on the OC4J side:

1. Oracle Application Server 10.1.3 middle-tier installation, with OC4J and Oracle HTTP Server, including the `mod_oc4j` Apache mod. Note that this is separate from the Web server you install on the COREid side, which may or may not be Oracle HTTP Server.

   Using the COREid Access security provider requires Oracle HTTP Server; you cannot use standalone OC4J.

2. COREid WebGate installed on this Oracle HTTP Server.

3. Additional OC4J instances as needed. Typically, when using COREid SSO, multiple OC4J instances are used in the topology, so the Oracle HTTP Server instance must be configured to route and maintain multiple OC4J instances.

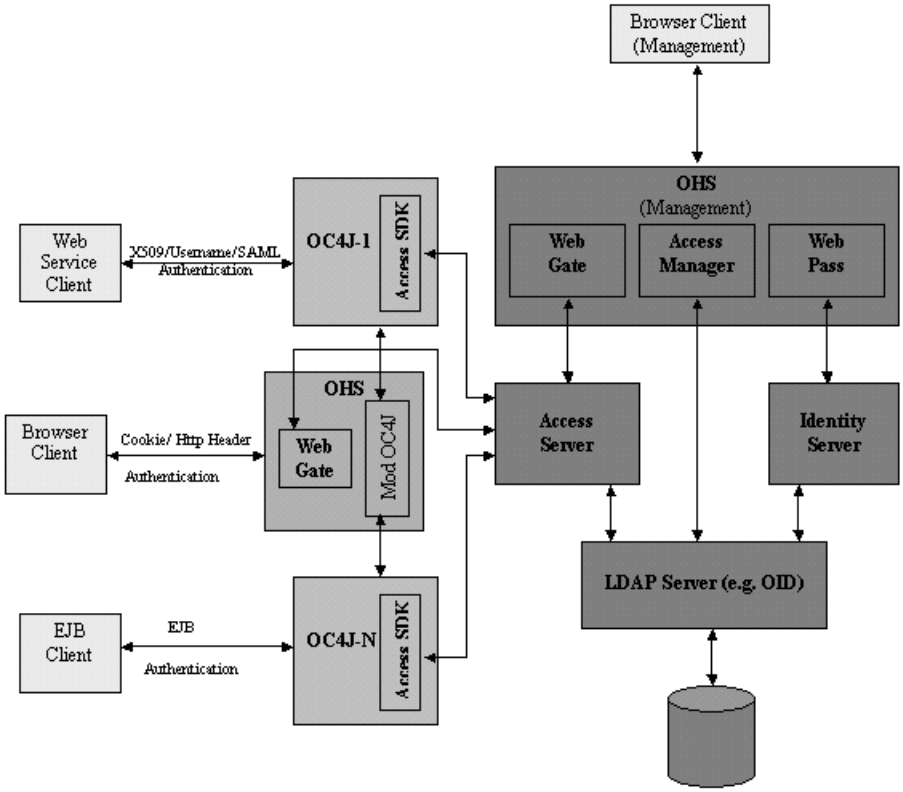
4. COREid Access SDK, one for each OC4J instance, on the same system as OC4J. The Access SDK is required by OC4J at runtime to communicate with Access Server.

The next section, "COREid Architecture", shows how the Oracle COREid Access and Identity components fit with key components of the Oracle Application Server middle-tier infrastructure.

COREid Architecture

Figure 10–1 shows the Oracle COREid Access and Identity architecture.

Figure 10–1 COREid Architecture



Top-Level Summary of Configuration Stages

There are three stages of configuration steps to use OC4J applications with the COREid Access security provider:

1. One-time configurations for the COREid installation. This includes setting up authentication schemes and configuring a COREid resource type. Refer to "Configuring COREid Access" on page 10-8.
2. Configurations for each OC4J instance. This includes configuring each OC4J instance with an Access SDK installation. Refer to "Configuring OC4J with the Access SDK" on page 10-14.
3. Configurations for your application. This includes `web.xml` settings, deployment-time settings, `orion-application.xml` settings (pre- or post-deployment), and JAAS login module settings. Refer to "Configuring the Application" on page 10-15.

Running the Access Manager

Several of the configuration steps documented later in this chapter involve running the COREid Access Manager. Run it with a URL such as the following, then log in:

`http://`*`host`*`:`*`port`*`/access/oblix`

This will put you at the Access System Console, used frequently in this chapter.

Oracle COREid Access and Identity Concepts

This section provides background on some COREid concepts that will be relevant later in this chapter:

- About COREid Resource Types
- About COREid Authentication
- About Using HTTP Header Variables for Authentication
- About the COREid Single Sign-On Cookie

About COREid Resource Types

In Oracle COREid Access and Identity, a resource type describes the kind of resource to be protected, including its associated operations. Operations associated with a resource are tied to its type. Before you can add resources to a policy domain, you must define their types and the operation or operations associated with them that you want to protect.

For example, by default Oracle COREid Access and Identity supports resource types that are named "HTTP" and "EJB". The HTTP resource type supports operations such as `CONNECT`, `DELETE`, `GET`, `POST`, `PUT`, and `TRACE`. The EJB resource type supports the operation `EXECUTE`, which executes a bean. For a custom resource type, you can specify a custom operation name.

To create a session to the Access Server, OC4J uses the Access SDK, and the SDK expects some resource type and resource operation to be specified. For this reason, when you configure the COREid login module, you must configure a custom COREid resource type, including the following:

- Desired name of the resource type (can be arbitrary)
- Desired name of the operation (can be arbitrary)

  You will specify just a single resource operation, but this will encompass whatever you want to execute against the protected resource.
- URL of the protected resource

See Also:

- *Oracle COREid Access and Identity Administration Guide* for information about COREid resource types

About COREid Authentication

In order to validate any user, COREid must be configured with an authentication scheme. An authentication scheme consists of plug-ins.

OC4J support for Oracle COREid Access uses the COREid `credential_mapping` plug-in, which maps user credentials to profiles, and, where applicable, the `validate_password` plug-in, which validates user passwords. You must configure these plug-ins as shown later in this chapter.

Additionally, OC4J supports two modes of integrating end-user authentication (identity assertion) with COREid Access:

- Use of the COREid SSO cookie, `ObSSOCookie`, discussed further in the next section, "About the COREid Single Sign-On Cookie"

- Use of a user name and password that are passed in HTTP headers, discussed further in "About Using HTTP Header Variables for Authentication" on page 10-7

See Also:

- *Oracle COREid Access and Identity Administration Guide* for information about COREid plug-ins

About the COREid Single Sign-On Cookie

Oracle COREid Access implements single-domain and multi-domain single sign-on through an encrypted session cookie called the `ObSSOCookie`. (This is one of two possible modes of end-user authentication, the other involving HTTP header variables as discussed in the next section.) WebGate sends this cookie to the user's browser upon successful authentication. This cookie can then act as an authentication mechanism for other protected resources that require the same or a lower level of authentication.

When a user requests access to a resource, the request flows from WebGate to the Access Server. Once the user is validated, the `ObSSOCookie` is set, and then passed to OC4J. With this single sign-on functionality, COREid uses the cookie for subsequent requests instead of prompting the user to supply authentication credentials.

OC4J uses the `ObSSOCookie` to connect to the COREid Access Server and retrieve user roles.

Note: `ObSSOCookie` is a session cookie by default, but can be made persistent.

See Also:

- *Oracle COREid Access and Identity Administration Guide* for information about the `ObSSOCookie`

About Using HTTP Header Variables for Authentication

COREid supports the use of HTTP header variables for authentication, where a user name and password are passed in HTTP headers to assert an end user. (This is one of two possible modes of end-user authentication, the other being the use of the COREid `ObSSOCookie` as discussed in the preceding section.)

To use this mode, you must configure the COREid login module with this user name and password (as discussed in "Configure the COREid JAAS Login Module" on page 10-16) for OC4J to use in accessing the COREid Access Server.

Consider the 4K size limit of the HTTP header when you use HTTP header variables and cookies to pass information to downstream applications. This HTTP header size limit includes all cookies, server variables, and environment variables—that is, all of the content of the HTTP header. There is no constraint on the number of individual elements an HTTP header can contain if the content does not exceed the 4K limit. Therefore, when assessing the amount of available space in the HTTP header, take into account the byte size of the data used by COREid and other applications. For example, if COREid and other applications combine to use 1K in the HTTP header, you would have 3K for your data.

Configuring COREid Access

This section discusses one-time configurations to your COREid Access installation:

1. Configure COREid Form-Based Authentication
2. Configure COREid Basic Authentication
3. Configure the Resource Type
4. Protect the Action URL

Configure COREid Form-Based Authentication

For single sign-on functionality, a form-based authentication scheme must be used in protecting your resources, due to limitations in the basic authentication scheme. (Some aspects of your configuration will have to use a no-password authentication, however, as discussed in "Configure COREid Basic Authentication" on page 10-10.)

The steps here are to create and protect a login page for form-based authentication in Oracle COREid Access and Identity, for use by WebGate in protecting your resource. You will later configure your application to be protected by this form-based authentication.

1. Create a Login Form
2. Define Form-Based Authentication in Access Manager
3. Configure the credential_mapping Plug-In for Form-Based Authentication
4. Configure the validate_password Plug-In for Form-Based Authentication

See Also:

- Discussion of how to configure form-based authentication in the *Oracle COREid Access and Identity Administration Guide* for additional information about the steps described here

Create a Login Form

Create a login page for form-based authentication. As will be pointed out, some of the parameters you set in this page correspond to settings you will make in Access Manager and the COREid plug-ins.

Put the login page under the *OHS_HOME*/*document_root* directory, typically *ORACLE_HOME*/Apache/Apache/htdocs, on the middle-tier system.

Here is a sample login page, `login.html`. Assume it is in the *ORACLE_HOME*/Apache/Apache/htdocs/login directory.

```
<html>
<head>
<title> COREid SSO Login Page</title>
<body bgcolor="white">
<h1 align="center">COREid SSO Provider Example : Sign in</h1>
<form method="POST" action="/coreid/access/test.html" >
  <table border="0" cellspacing="5">
    <tr>
      <th align="right">Username:</th>
      <td align="left"><input type="text" name="usernamevar"></td>
    </tr>
    <tr>
      <th align="right">Password:</th>
      <td align="left"><input type="password" name="passwordvar"></td>
```

```
    </tr>
    <tr>
      <td align="right"><input type="submit" value="Log In"></td>
      <td align="left"><input type="reset"></td>
    </tr>
  </table>
</form>
</body>
</html>
```

The action URL for the `POST` method can be arbitrary, but must match the action URL you specify when you configure authentication management in Access Manager in the next step.

The variable for the user name (`usernamevar` here) must match what you specify in the COREid `credential_mapping` plug-in. The variable for the password (`passwordvar` here) must match what you specify in the COREid `validate_password` plug-in.

Define Form-Based Authentication in Access Manager

This step uses the COREid Access Manager to define form-based authentication. Navigate as follows in Access Manager:

Access System Console > Access System Configuration > Authentication Management

List all authentication schemes, then choose to add a new scheme. In the **General** tab to define a new authentication scheme, makes entries such as the following:

```
Name:                   COREidSSOform
Description:            COREid SSO Form Based
Level:                  1
Challenge Method:       Form
Challenge Parameter:    form: /login/login.html
                        creds: usernamevar passwordvar
                        action: /coreid/access/test.html
                        passthrough: No

SSL Required:           No
Challenge Redirect
Enabled:                Yes
```

You can choose any name and description; here "COREidSSOform" and "COREid SSO Form Based" are just examples. The challenge parameter specifies `login/login.html` as the form because that is the path relative to the Oracle HTTP Server document root where we created the login page in the previous step. Leave "Challenge Redirect" blank.

Note that the entries for "creds" here must match the variables specified for user and password in your login page in the previous step, and these variables are used in the `credential_mapping` plug-in and `validate_password` plug-in, respectively, for form-based authentication.

Also note that the action URL (`/coreid/access/test.html` here) can be arbitrary, but must match the action URL for the `POST` method in your login page. Protect this URL with the basic (no password) authentication scheme described in "Configure COREid Basic Authentication" on page 10-10.

Configure the credential_mapping Plug-In for Form-Based Authentication

Next, you must configure the COREid `credential_mapping` plug-in for form-based authentication in Access Manager. This is to protect the login form.

Navigate to the appropriate page as follows:

Access System Console > Access System Configuration > Authentication Management

List all authentication schemes, then choose the form-based scheme, then go to the **Plugins** tab.

Modify the `credential_mapping` plug-in with entries such as the following:

```
obMappingBase="cn=users,dc=us,dc=oracle,dc=com",obMappingFilter="(&(&
(objectclass=inetorgperson)(uid=%usernamevar%))(|(!
(obuseraccountcontrol=*)) (obuseraccountcontrol=ACTIVATED)))"
```

The value entered for `uid` (`usernamevar` here) must match the variable you specified for the user name in the login form, and when defining form-based authentication in Access Manager, shown in previous steps.

This also corresponds to the value of the `coreid.name.attribute` option in the COREid login module configuration in OC4J.

See Also:

- *Oracle COREid Access and Identity Administration Guide* for more information about the `credential_mapping` plug-in

Configure the validate_password Plug-In for Form-Based Authentication

Now configure the COREid `validate_password` plug-in for form-based authentication in Access Manager. This is to protect the login form.

Navigate as for the `credential_mapping` plug-in in the previous step. Modify the `validate_password` plug-in with an entry such as the following:

```
obCredentialPassword="passwordvar"
```

The value entered for `obCredentialPassword` (`passwordvar` here) must match the password variable specified in the login page, and when defining form-based authentication in Access Manager, shown in previous steps.

This also corresponds to the value of the `coreid.password.attribute` option in the COREid login module configuration.

Configure COREid Basic Authentication

You must configure the COREid basic authentication scheme, which must not be password protected. (It will use only the `credential_mapping` plug-in, not the `validate_password` plug-in.) This scheme will be used to protect two resources:

- A URL associated with the resource type that you configure, as discussed in "Configure and Protect the URL of the Configured Resource Type" on page 10-12. The COREid login module will use this URL to communicate to the Access Server through the Access SDK.
- The action URL for the form page, noted in "Create a Login Form" on page 10-8 and "Define Form-Based Authentication in Access Manager" on page 10-9. This is so submitted form requests can be intercepted by WebGate in order to enforce rules for submitted credentials.

(Your application itself, however, must be protected by form-based authentication, described in "Configure COREid Form-Based Authentication" on page 10-8.)

These steps define basic authentication, without a password, to protect the resource.

1. Define Basic Authentication in Access Manager
2. Configure the credential_mapping Plug-In for Basic Authentication

Define Basic Authentication in Access Manager

This step uses the COREid Access Manager to configure basic authentication. Navigate as follows in Access Manager:

Access System Console > Access System Configuration > Authentication Management

List all authentication schemes, then choose to add a new scheme. In the **General** tab to define a new authentication scheme, makes entries such as the following:

```
Name:                    COREidSSONoPwd
Description:             Authentication without Password
Level:                   1
Challenge Method:        Basic
Challenge Parameter:     realm:NetPoint Basic Over LDAP
SSL Required:            No
Challenge Redirect
Enabled:                 Yes
```

You can choose any name and description; here "COREidSSONoPwd" and "Authentication without Password" are just examples. The challenge parameter entry indicated here is one of the choices available from a dropdown list. Leave "Challenge Redirect" blank.

Configure the credential_mapping Plug-In for Basic Authentication

Next, configure the COREid `credential_mapping` plug-in for basic authentication in Access Manager. This is to protect your resource, but without a password so WebGate can intercept results.

Navigate to the appropriate page as follows:

Access System Console > Access System Configuration > Authentication Management

List all authentication schemes, then choose the basic scheme, then go to the **Plugins** tab.

Modify the `credential_mapping` plug-in with entries such as the following:

```
obMappingBase="cn=users,dc=us,dc=oracle,dc=com",obMappingFilter="(&(&
(objectclass=inetorgperson)(uid=%usernamevar%))(|(!
(obuseraccountcontrol=*)) (obuseraccountcontrol=ACTIVATED)))"
```

These are the same entries as for the `credential_mapping` plug-in for form-based authentication. The value entered for `uid` (`usernamevar` here) must match the user name variable specified in the login form.

This also corresponds to the value of the `coreid.name.attribute` option in the COREid login module configuration.

See Also:

- *Oracle COREid Access and Identity Administration Guide* for more information about the `credential_mapping` plug-in

Configure the Resource Type

In Oracle COREid Access and Identity, a resource type describes the kind of resource to be protected, including its associated operations. Operations associated with a resource are tied to its type. You must configure an Oracle COREid Access and Identity resource type for your resource, and then protect your resource type, action URL, and application.

There are three parts to configuring the resource type for your resource, accomplished through Access Manager and described below:

1. Configure the Name and Operation of the Resource Type
2. Configure and Protect the URL of the Configured Resource Type
3. Configure the Return Action Attributes

The COREid login module will need information for the resource type, as will be noted. OC4J uses the resource type to retrieve user information based on the Oracle COREid Access and Identity `ObSSOCookie` or the user name, using APIs of the Access SDK.

Once these configuration steps are complete, the resource URL will be associated with the resource type and protected by the no-password basic authentication method you configured.

See Also:

- "About COREid Resource Types" on page 10-6

Configure the Name and Operation of the Resource Type

To configure the name and operation of the resource type in Access Manager, navigate as follows:

Access System Console > Access System Configuration > Common Information Configuration > Resource Type Definitions

On the page that lists all resource types, choose to add a new resource type.

Make entries such as the following to define a new resource type:

```
Resource Name:        myresourcetype
Display Name:         My Resource Type Display Name
Resource Matching:    Case Insensitive
Resource Operation:   myresourceoperation
```

You can choose any names for the resource type and resource operation, but you must use the same names for the `coreid.resource.type` and `coreid.resource.operation` option values in the COREid login module configuration.

Configure and Protect the URL of the Configured Resource Type

To configure and protect the URL of your configured resource type in Access Manager, navigate as follows:

Access Manager > Create Policy Domains

Choose the link for your policy domain. In the **Resources** tab, use entries such as the following:

```
Resource Type:      myresourcetype
Host Identifiers:   myhost
```

```
URL Prefix:          /myresourceurl
Description:         My Description
```

This configuration must be protected using a no-password scheme. Use the basic scheme that you configured in "Define Basic Authentication in Access Manager" on page 10-11.

Choose your resource type (`myresourcetype` in these examples) from the dropdown list, then choose the appropriate host name.

The URL prefix must start with a "`/`" and is the designated URL of the resource type. This must match the value of the `coreid.resource.name` option in the COREid login module configuration.

The description can be anything; "My Description" is just an example.

> **Note:** Do not confuse the URL specified here with the "action URL" specified when setting up authentication in earlier steps. The two are separate.

Configure the Return Action Attributes

After authentication, OC4J requires access to the user's roles in order to check for authorization. To enable this, you must set up a COREid "return action" that allows COREid to return the appropriate roles to OC4J for the user after successful authentication.

To set up the return action in COREid, navigate as follows:

Access Manager > My Policy Domains > Select myresourcetype > Authorization Rules tab > Choose role name > Actions tab

Under the Authorization Success section, add the following entries (continuing the preceding example using `myresourcetype`):

```
Return Type:      myresourcetype
Return Name:      myresourcetype
Return Attribute: ObMyGroups
```

`ObMyGroups` is an action attribute defined in Oracle COREid Access and Identity for use in returning all the roles of an authenticated user.

See Also:

- *Oracle COREid Access and Identity Administration Guide* for information about the `ObMyGroups` action attribute

Protect the Action URL

Using Access Manager, protect the action URL you specified in "Configure COREid Form-Based Authentication" on page 10-8. Use similar steps as for protecting the resource type URL, as described in "Configure and Protect the URL of the Configured Resource Type" on page 10-12.

- This configuration must be under a no-password authentication scheme. Use the basic authentication scheme that you configured in "Configure COREid Basic Authentication" on page 10-10.
- Use "HTTP" as the resource type.
- Specify the action URL (`/coreid/access/test.html` in the examples).

See Also: For information about how to protect resources:

- *Oracle COREid Access and Identity Administration Guide* (volume 2, chapter 3)

Configuring OC4J with the Access SDK

This section describes configuration steps for each OC4J instance on the middle tier.

As a prerequisite to this, you must install WebGate on the Oracle HTTP Server instance in the middle tier. This Oracle HTTP Server instance, in turn, can (and typically does) support multiple OC4J instances.

This section covers the following steps:

1. Create OC4J Instances as Needed
2. Configure the Access SDK to Each OC4J Instance
3. Configure the Access SDK Library Path for Each OC4J Instance

---

Note: Your middle-tier and OC4J installation can be on the same system as COREid, but would typically not be.

See Also:

- *Oracle COREid Access and Identity Installation Guide* for information about installing AccessGate/WebGate

Create OC4J Instances as Needed

Typically, when using COREid SSO, multiple OC4J instances are used in the topology, so the Oracle HTTP Server instance must be configured to route and maintain multiple OC4J instances:

1. Create new OC4J instances as desired, using the `createinstance` utility as described in the *Oracle Containers for J2EE Configuration and Administration Guide*.
2. Each OC4J instance should be tied to the Oracle HTTP Server instance. Each application deployed to an OC4J instance must be configured in the `mod_oc4j` configuration file, *`ORACLE_HOME`*`/Apache/Apache/conf/mod_oc4j.conf`, so that requests are properly routed to the OC4J instance. This should occur automatically when you create the OC4J instance.

Configure the Access SDK to Each OC4J Instance

You will need COREid Access SDK, one installation for each OC4J instance, on the same system as OC4J. The Access SDK is required by OC4J at runtime to communicate with Access Server. OC4J must be given the Access SDK location during startup (through the `java.library.path` property, as shown later in this chapter), so that it can initialize the SDK. Note this initialization occurs only if at least one application is using COREid Access as the security provider. Also note the following:

1. Create a separate Access SDK installation for each OC4J instance, on the same system as OC4J. You can have multiple Access SDK installations on the same system.
2. Configure each Access SDK to work with the appropriate Access Server. From the *`Access_SDK_Home`*`/access/oblix/tools/configureAccessGate`

directory, run the command `configureAccessGate`. This utility requires the Access Server ID, AccessGate ID, and other related parameters.

3. Copy the COREid file `jobaccess.jar` from the Access SDK to the OC4J path. You will find this file in the *`Access_SDK_Home`*`/AccessServerSDK/oblix/lib` directory. Create the directory *`ORACLE_HOME`*`/j2ee/home/lib/ext` (if it does not already exist) and copy the `jobaccess.jar` to that directory.

See Also:

- *Oracle COREid Access and Identity Developer Guide* for information about installing the Access SDK
- *Oracle COREid Access and Identity Administration Guide* for information about the `configureAccessGate` utility

Configure the Access SDK Library Path for Each OC4J Instance

You must configure the `java.library.path` property for each OC4J instance, in the *`ORACLE_HOME`*`/opmn/conf/opmn.xml` file, so that the OC4J instance has access to the Access SDK at runtime. Set the property so that it points to the SDK location.

For example, on a Windows system:

```
-Djava.library.path=C:\CoreID\AccessSDK\AccessServerSDK\oblix\lib
```

See Also:

- *Oracle Process Manager and Notification Server Administrator's Guide* for information about OPMN and the `opmn.xml` file

Configuring the Application

Instructions in this section are geared toward a Web application, consisting of the following steps:

1. Protect the Application URLs in web.xml
2. Settings for Application Deployment
3. Configure COREid SSO in orion-application.xml
4. Protect the Application URLs in COREid Access
5. Configure the COREid JAAS Login Module
6. Test the Application

Protect the Application URLs in web.xml

The first step in protecting your application is to protect appropriate URLs or URL prefixes through settings in the `web.xml` file, using standard J2EE features.

These are the same URLs that you will you protect through COREid configuration in "Protect the Application URLs in COREid Access" on page 10-16.

Settings for Application Deployment

In the Oracle Application Server 10.1.3 implementation, Application Server Control does not yet support COREid Access as a security provider. When you deploy your application using the Application Server Control Console, choose the file-based

provider. This will be overridden through the configuration steps documented in this chapter.

Configure COREid SSO in orion-application.xml

To use COREid Single Sign-On as the authentication method for Web applications, set the `auth-method` attribute to "`COREIDSSO`" in the `<jazn-web-app>` element in the OC4J `orion-application.xml` file. You can do this as either a pre-deployment step (packaged in the EAR file) or a post-deployment step.

Notes:

- You do not need an `<auth-method>` setting in the `web.xml` file. Any setting in `web.xml` would be overridden by the "`COREIDSSO`" setting in `orion-application.xml`.
- The `<jazn-web-app>` element is also supported in the `orion-web.xml` file. In the event of conflict, `orion-web.xml` takes precedence over `orion-application.xml` for the particular Web application in question.

Here is a sample entry in `orion-application.xml`, where `<jazn-web-app>` is a subelement of the `<jazn>` element:

```
<orion-application ... >
   ...
   <jazn provider="XML" >
      <jazn-web-app auth-method="COREIDSSO"/>
      ...
   </jazn>
   ...
</orion-application>
```

Protect the Application URLs in COREid Access

Use Access Manager to protect your application URLs or URL prefixes through form-based authentication. These will be the same URLs as in "Protect the Application URLs in web.xml" on page 10-15. Use the following navigation:

Access Manager > Create Policy Domains

Then choose the appropriate public policy domain. You should protect each URL or URL prefix you protected in `web.xml`, as follows:

1. Use "HTTP" as the resource type.
2. Specify the URL (for example, `/foo`).
3. The configuration must be under the form-based authentication scheme that you defined in "Configure COREid Form-Based Authentication" on page 10-8.

See Also: For information about how to protect resources:

- *Oracle COREid Access and Identity Administration Guide* (volume 2, chapter 3)

Configure the COREid JAAS Login Module

For a Web application, the OC4J implementation to support Oracle COREid Access and Identity requires the login module `CoreIDLoginModule`, supplied by Oracle.

The following template shows the general form of the configuration, in the `system-jazn-data.xml` file. Note the `<class>` and `<control-flag>` element settings. Table 10–1 describes the available options. Examples of specific scenarios and their configurations are shown later in this chapter.

> **Note:** As with other custom login modules, the setting `provider="XML"` is used with the COREid login module.

```
<application>
   <name>yourappname</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>option1name</name>
               <value>option1value</value>
            </option>
            <option>
               <name>option2name</name>
               <value>option2value</value>
            </option>

            ...

         </options>
      </login-module>
   </login-modules>
</application>
```

Table 10–1 COREid Login Module Options

Option Name	Required/Optional	Option Value
addAllRoles	Required	This flag should be set to `true` so the authenticated user will have permissions for all his/her roles. With a `false` setting, there are permissions only for top-level roles, not nested roles.
coreid.resoure.type	Required	Name of the resource type you defined through Access Manager. **See Also**: "About COREid Resource Types" on page 10-6 and "Configure the Name and Operation of the Resource Type" on page 10-12
coreid.resource.operation	Required	Name of the resource operation associated with the resource type specified in `coreid.resource.type`, as defined through Access Manager. **See Also**: "Configure the Name and Operation of the Resource Type" on page 10-12

Table 10–1 (Cont.) COREid Login Module Options

Option Name	Required/Optional	Option Value
coreid.resource.name	Required	The URL prefix associated with the resource type specified in `coreid.resource.type`, and protected using the no-password basic authentication scheme defined through Access Manager. **See Also**: "Configure and Protect the URL of the Configured Resource Type" on page 10-12
coreid.name.attribute	Required	Variable for the user name for authentication, as defined in the `credential_mapping` plug-in. **See Also**: "About COREid Authentication" on page 10-6 and "Configure the credential_mapping Plug-In for Form-Based Authentication" on page 10-10
coreid.password.attribute	Required (except when using X.509 token or SAML authentication)	Variable for the password for authentication, as defined in the `validate_password` plug-in. **See Also**: "Configure the validate_password Plug-In for Form-Based Authentication" on page 10-10
coreid.name.header	Optional	If you use HTTP header variables for authentication, this parameter is the user name that OC4J should use to authenticate against the COREid Access Server. **See Also**: "About Using HTTP Header Variables for Authentication" on page 10-7 and "Web Application Using HTTP Header Variables through COREid" on page 10-20
coreid.password.header	Optional	If you use HTTP header variables for authentication, this parameter is the password that OC4J should use with the user name specified in `coreid.name.header` to authenticate against the COREid Access Server.

Note: The values of `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name` are determined during one-time COREid configuration, as described in "Configure the Resource Type" on page 10-12, and are the same for any application using the same installation of Oracle COREid Access and Identity. Each application must configure these property values in its configuration for the COREid login module.

The following sample corresponds to the example that runs throughout "Configure COREid Form-Based Authentication" on page 10-8, "Configure COREid Basic Authentication" on page 10-10, and "Configure the Resource Type" on page 10-12:

```
<application>
   <name>foo</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>usernamevar</value>
            </option>
            <option>
               <name>coreid.password.attribute</name>
               <value>passwordvar</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

(This uses all supported options for the COREid login module except for `coreid.name.header` and `coreid.password.header`. Examples for these are shown later in this chapter.)

Test the Application

After you have deployed your Web application, restarted OC4J, and restarted Oracle HTTP Server, run the application. This example assumes Oracle HTTP Server listens on port 6666:

```
http://www.example.com:6666/foo
```

WebGate will intercept this request and will check the authentication scheme for this URL. The configuration shown earlier in this chapter will result in the user being prompted with the `login.html` login form from "Create a Login Form" on page 10-8. Then the following sequence will take place:

1. WebGate will capture the user name and password from the login form and communicate to Access Server.
2. Access Server will communicate to Oracle Internet Directory (or other repository that you use).

3. After the user is authenticated, the COREid SSO token will be returned to WebGate.
4. WebGate will set the `ObSSOCookie` and pass the cookie and other HTTP headers to `mod_oc4j`, which will route the request to the appropriate OC4J instance.
5. OC4J will take the cookie and validate it, or retrieve roles for the user associated with this cookie from Access Server using the Access SDK configured on OC4J.

COREid Examples for J2EE Applications

This section discusses the following COREid usages for Web applications and EJBs:

- Web Application Using HTTP Header Variables through COREid
- Web Application Using the COREid ObSSOCookie
- EJB Application Using COREid

See Also:

- "COREid Support and Examples for Web Services" on page 10-23

Web Application Using HTTP Header Variables through COREid

You can optionally configure a Web application to use HTTP header variables for authentication. The header variable for user name corresponds to the `coreid.name.header` option in the COREid login module configuration. The header variable for password corresponds to the `coreid.password.header` option.

You must execute the following steps to use these header variables:

1. Configure HTTP Header Variables in Access Manager
2. Configure HTTP Header Variables for the COREid Login Module
3. Secure the Web Application

See Also:

- "About Using HTTP Header Variables for Authentication" on page 10-7

Configure HTTP Header Variables in Access Manager

Use Access Manager to create an HTTP header variable for passing the user name, and, as appropriate, a header variable for passing the password. Whether to include the password depends on whether your COREid authentication scheme uses just the `credential_mapping` plug-in, or also uses the `validate_password` plug-in.

See Also: For information about using HTTP header variables:

- *Oracle COREid Access and Identity Administration Guide* (volume 2, chapters 4 and 5)

Configure HTTP Header Variables for the COREid Login Module

Include option settings for `coreid.name.header` and (as appropriate) `coreid.password.header` in the COREid login module configuration in `system-jazn-data.xml`. In the following example, password authentication is used, and the HTTP header variables you created in the previous step are `myhttpuservar` and `myhttppwdvar`:

```
<options>
   ...
   <option>
      <name>coreid.name.header</name>
      <value>myhttpuservar</value>
   </option>
   <option>
      <name>coreid.password.header</name>
      <value>myhttppwdvar</value>
   </option>
   ...
</options>
```

> **Note:** When using HTTP header variables, be aware that option settings for `coreid.name.attribute` and `coreid.password.attribute` are still required, in addition to settings for `coreid.name.header` and `coreid.password.header`.

Secure the Web Application

Define appropriate security constraints in your standard Web application configuration, and set `auth-method="COREIDSSO"` in `orion-application.xml` as shown in "Configure COREid SSO in orion-application.xml" on page 10-16.

Web Application Using the COREid ObSSOCookie

When no HTTP header variables are provided for a secure Web application, the COREid `ObSSOCookie` is used to retrieve authentication information. By default, this cookie contains the cookie in the HTTP header.

You must execute the following steps to use the cookie:

1. Configure User Name and Password for the COREid Login Module
2. Secure the Web Application

Configure User Name and Password for the COREid Login Module

Include option settings for `coreid.name.attribute` and (as appropriate) `coreid.password.attribute` in the COREid login module configuration in `system-jazn-data.xml`. In the following example, password authentication is used, and the user name and password variables you defined for the `credential_mapping` and `validate_password` plug-ins are `usernamevar` and `passwordvar`:

```
<options>
   ...
   <option>
      <name>coreid.name.attribute</name>
      <value>usernamevar</value>
   </option>
   <option>
      <name>coreid.password.attribute</name>
      <value>passwordvar</value>
   </option>
   ...
</options>
```

Secure the Web Application

Define appropriate security constraints in your standard Web application configuration, and set `auth-method="COREIDSSO"` in `orion-application.xml` as shown in "Configure COREid SSO in orion-application.xml" on page 10-16.

EJB Application Using COREid

For EJB authentication, OC4J gets the user name and password from the EJB context and passes them to the COREid login module. The same user name and password are used to authenticate against Oracle COREid Access and Identity.

The EJB scenario requires both the `credential_mapping` plug-in and the `validate_password` plug-in, discussed earlier in this chapter. The user name and password variables you define for the plug-ins must be reflected in option settings for the COREid login module, as discussed in "Configure COREid Form-Based Authentication" on page 10-8.

The client must send the user name and password for authenticating itself before it can access the EJB.

Configure the COREid login module, where COREid Access authentication variables are as follows:

- `myejbappname` is the name of the EJB application.
- `myejbusernamevar` is the variable name for the EJB user name, as you define in the `credential_mapping` plug-in.
- `myejbpwdvar` is the variable name for the EJB user password, as you define in the `validate_password` plug-in.

```
<application>
   <name>myejbappname</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>myejbusernamevar</value>
            </option>
            <option>
```

```
            <name>coreid.password.attribute</name>
            <value>myejbpwdvar</value>
          </option>
        </options>
      </login-module>
    </login-modules>
</application>
```

Note: In the current release there is no direct support for a scenario where COREid `ObSSOCookie` is sent instead of the user name and password for authentication.

See Also:

- "Configure the Resource Type" on page 10-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`

COREid Support and Examples for Web Services

Web services can use Oracle COREid Access and Identity for authenticating Web service clients. With respect to COREid, OC4J supports username token authentication, X.509 token authentication, and SAML token authentication, as follows:

- Username token: OC4J extracts the user name and password, and uses them to authenticate against COREid.
- X.509 token: OC4J uses the CN value of the X.509 entry to authenticate against COREid.
- SAML token: OC4J uses the subject name to authenticate against COREid.

The following usages are shown below:

- Web Service with Username Token Authentication for COREid
- Web Service with X.509 Token Authentication for COREid
- Web Service with SAML Token Authentication for COREid

Note: In the current release there is no direct support for a scenario where the COREid `ObSSOCookie` is sent instead of the user name and password for authentication.

See Also:

- "COREid Examples for J2EE Applications" on page 10-20
- *Oracle Application Server Web Services Security Guide* for general information about username token, X.509 token, and SAML token authentication

Web Service with Username Token Authentication for COREid

A username token client uses the user name and password for authentication. You must configure variables for the user name and password through the COREid

credential_mapping and validate_password plug-ins, with corresponding settings for the coreid.name.attribute and coreid.password.attribute options in the COREid login module configuration, as discussed in "Configure COREid Form-Based Authentication" on page 10-8.

Configure the COREid login module as follows, where:

- UsernameAppName is the name of the Web service application using username token authentication.
- UsernameNamevar is the variable name for the user name, as you define in the credential_mapping plug-in.
- UsernamePwdvar is the variable name for the user password, as you define in the validate_password plug-in.

```
<application>
   <name>UsernameAppName</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>UsernameNamevar</value>
            </option>
            <option>
               <name>coreid.password.attribute</name>
               <value>UsernamePwdvar</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

See Also:

- "Configure the Resource Type" on page 10-12 for information about coreid.resource.type, coreid.resource.operation, and coreid.resource.name

Web Service with X.509 Token Authentication for COREid

An X.509 client uses the CN value from the X.509 entry for authentication. You must configure a variable for the CN user name through the COREid `credential_mapping` plug-in, with a corresponding setting for the `coreid.name.attribute` option in the COREid login module configuration, as discussed in "Configure COREid Form-Based Authentication" on page 10-8.

You do not configure the COREid `validate_password` plug-in or set the login module `coreid.password.attribute` option when X.509 token authentication is used.

Configure the COREid login module as follows, where:

- `X509AppName` is the name of the Web service application using X.509 token authentication.
- `cn_name_var` is the variable name for the CN user name, as you define in the `credential_mapping` plug-in.

```
<application>
   <name>X509AppName</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>coreid.resource.type</name>
               <value>myresourcetype</value>
            </option>
            <option>
               <name>coreid.resource.operation</name>
               <value>myresourceoperation</value>
            </option>
            <option>
               <name>coreid.resource.name</name>
               <value>/myresourceurl</value>
            </option>
            <option>
               <name>coreid.name.attribute</name>
               <value>cn_name_var</value>
            </option>
         </options>
      </login-module>
   </login-modules>
</application>
```

See Also:

- "Configure the Resource Type" on page 10-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`

Web Service with SAML Token Authentication for COREid

For a SAML client, OC4J determines the subject name, and you must configure a variable name for SAML subject authentication through the COREid `credential_mapping` plug-in. This `credential_mapping` setting must be reflected in the setting of the `coreid.name.attribute` option in the COREid login module configuration, as discussed in "Configure COREid Form-Based Authentication" on page 10-8. OC4J passes the subject name and `credential_mapping` variable name to COREid for authentication.

You do not configure the COREid `validate_password` plug-in or set the login module `coreid.password.attribute` option when SAML authentication is used.

Configure the COREid login module as shown below, where:

- `SAMLAppName` is the name of the Web service application using SAML token authentication.
- `subject_name_var` is the variable for the subject name, as you define in the `credential_mapping` plug-in.

In the SAML scenario, there is also a SAML login module, `SAMLLoginModule`, that you must configure along with the COREid login module, as follows. This example uses `www.example.com` for the issuer name.

> **Important:** The `SAMLLoginModule` configuration must precede the `CoreIDLoginModule` configuration in `system-jazn-data.xml`.

```
<application>
   <name>SAMLAppName</name>
   <login-modules>

      <login-module>
         <class>
            oracle.security.jazn.login.module.saml.SAMLLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
               <name>issuer.name.1</name>
               <value>www.example.com</value>
            </option>
         </options>
      </login-module>

      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
         <options>
            <option>
               <name>addAllRoles</name>
               <value>true</value>
            </option>
            <option>
```

```
            <name>coreid.resource.type</name>
            <value>myresourcetype</value>
         </option>
         <option>
            <name>coreid.resource.operation</name>
            <value>myresourceoperation</value>
         </option>
         <option>
            <name>coreid.resource.name</name>
            <value>/myresourceurl</value>
         </option>
         <option>
            <name>coreid.name.attribute</name>
            <value>subject_name_var</value>
         </option>
      </options>
   </login-module>

</login-modules>
</application>
```

See Also:

- "Configure the Resource Type" on page 10-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`
- *Oracle Application Server Web Services Security Guide* for information about the `SAMLLoginModule`

Troubleshooting the Oracle COREid Access and Identity Setup

Table 10–2 provides some troubleshooting tips for your Oracle COREid Access and Identity setup and configuration.

Table 10–2 Oracle COREid Access and Identity Troubleshooting

Problem	Cause/Solution
The application is configured to use COREid SSO. When you try to access the application, Access Server crashes and restarts.	This will happen if you have configured the incorrect search base in Oracle Internet Directory, or the group name is not properly created.
When you try to access the COREid SSO application, it throws a Class Not Found exception.	Confirm you copied the COREid file `jobaccess.jar` from the Access SDK to the OC4J path, as described in "Configure the Access SDK to Each OC4J Instance" on page 10-14.
When you try to access the COREid SSO application, it gives an internal server error.	Confirm that the Access SDK installed on the OC4J server is configured to use the appropriate Access Server, as discussed in "Configure the Access SDK to Each OC4J Instance" on page 10-14. Also confirm that OC4J is running.
When you try to access the COREid SSO application, it does not appear in the login page.	Confirm you have enabled your authentication scheme with proper settings, using Access Manager, as discussed in "Configure COREid Form-Based Authentication" on page 10-8.

Table 10–2 (Cont.) Oracle COREid Access and Identity Troubleshooting

Problem	Cause/Solution
When you try to access the COREid SSO application, the login page keeps coming back.	Confirm that the form-based authentication scheme is enabled, and that the form variable names (for user and password) in your login page are the same as you configured in the COREid form-based authentication scheme, and that the credential mapping scheme and password validation scheme are configured for the form-based authentication scheme. Refer to "Configure COREid Form-Based Authentication" on page 10-8.
You have configured the application to use Oracle COREid Access and Identity, but you always get an "unauthorized" or "unauthenticated" error.	Confirm that the COREid login module is correctly configured for this application in `system-jazn-data.xml`. Refer to "Configure the COREid JAAS Login Module" on page 10-16.
You have configured the application to use Oracle COREid Access and Identity, but you get an internal server error.	Confirm that the LDAP server (Oracle Internet Directory, for example) that is configured with the COREid Identity Server is running and accessible.
You have configured the application to use COREid SSO, but when you attempt to access it, after you enter the user name and password, the application hangs.	Confirm that the action URL used in the form page is protected with an authentication scheme without password, such as the basic scheme. (Protecting the action URL with a password-protected authentication scheme results in an execution loop.) See "Create a Login Form" on page 10-8.

11

Integration with SSL and ORMIS

OC4J supports Secure Socket Layer (SSL) communication between Oracle HTTP Server and OC4J in an Oracle Application Server environment, using AJPS. This is the secure version of Apache JServ Protocol, the protocol that Oracle HTTP Server uses to communicate with OC4J. (Note, however, that the AJPS protocol used between Oracle HTTP Server and OC4J is not visible to the end user.)

OC4J also supports ORMI over SSL, or ORMIS. With this feature, OC4J now supports RMI communication over SSL between objects across OC4J server instances.

This chapter covers the following topics:

Notes:

- Secure communication between a client and Oracle HTTP Server is independent of secure communication between Oracle HTTP Server and OC4J.
- This chapter assumes some prior knowledge of security and SSL concepts.

See Also:

- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Administrator's Guide* for information about configuring additional Oracle Application Server components to take advantage of SSL
- *Oracle Containers for J2EE Services Guide* for general information about using ORMI in OC4J

Using Keys and Certificates with OC4J and Oracle HTTP Server

The steps below describe using keys and certificates for SSL communication in OC4J. 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 Oracle Application Server instance.

Note that a *keystore* stores 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 uses what is called a *wallet* for the same purpose.

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.

The Oracle Wallet Manager has functionality for Oracle wallets that is equivalent to the functionality of `keytool` for keystores.

See Also:

- For information about `keytool`:

  `http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html`

- For information on Oracle Wallet Manager, the *Oracle Application Server Administrator's Guide*

Here are the steps in using certificates between OC4J and Oracle HTTP Server:

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.

   Generate your own signature:

   a. Use `keytool` to "self-sign" the certificate. This is appropriate if your clients trust you as, in effect, your own certificate authority.

   Alternatively, 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 is matched with the associated certificate.

Note: Oracle Application Server includes Oracle Application Server Certificate Authority (OCA). OCA enables customers to create and issue certificates for themselves and their users, although these certificates would probably be unrecognized outside a customer's organization without prior arrangements.

See Also:

- *Oracle Application Server Certificate Authority Administrator's 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 Oracle Application Server, this document 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, Inc., for example:

```
http://www.verisign.com/

http://www.thawte.com/
```

For SSL communication between OC4J and Oracle HTTP Server, execute the preceding steps for Oracle HTTP Server, but use a wallet and Oracle Wallet Manager instead of a keystore and the `keytool` utility.

In addition to steps 1 and 2 above, execute the following steps as necessary:

1. **If the OC4J certificate is signed by an entity that Oracle HTTP Server does not yet trust**, obtain the certificate of the entity and import it into Oracle HTTP Server. The specifics depend on whether the OC4J certificate in question is self-signed, as follows.

   If OC4J has a self-signed certificate (essentially, Oracle HTTP Server does not yet trust OC4J):

   a. From OC4J, use `keytool` to export the OC4J certificate. This step places the certificate into a file that is accessible to Oracle HTTP Server.

   b. From Oracle HTTP Server, use Oracle Wallet Manager to import the OC4J certificate.

   Alternatively, if OC4J has a certificate that is signed by another entity (that Oracle HTTP Server does not yet trust):

   a. Obtain the certificate of the entity in any appropriate way, such as by exporting it from the entity. The exact steps vary widely, depending on the entity.

   b. From Oracle HTTP Server, use Oracle Wallet Manager to import the certificate of the entity.

2. **If the Oracle HTTP Server certificate is signed by an entity that OC4J does not yet trust, and OC4J is in a mode of operation that requires client authentication:**

   (This is discussed in "Requesting Client Authentication" on page 11-12.)

   a. Obtain the certificate of the entity in any appropriate way, such as by exporting it from the entity. The exact steps vary widely, depending on the entity.

   b. From OC4J, use `keytool` to import the certificate of the entity.

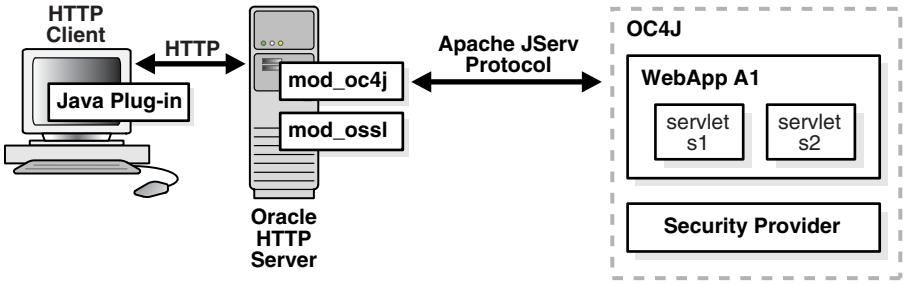
Note: During communications over SSL between Oracle HTTP Server and OC4J, all data on the communications channel between the two is encrypted. The following steps are executed:

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 process also occurs during establishment of the encrypted channel.
3. Any further communication after this initial exchange will be encrypted.

Integrating the Security Provider with SSL-Enabled Applications

SSL is an industry standard protocol for managing the security of message transmission on the Internet. Figure 11–1 shows an application running in an SSL-enabled J2EE environment.

Figure 11–1 Oracle Component Integration in SSL-Enabled J2EE Environments



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 OC4J. Oracle HTTP Server handles the request.
2. The `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 or client wallet certificate is configured, the HTTP client is prompted to accept the server certificate of Oracle HTTP Server and provide the client certificate.
4. OC4J security provider retrieves the SSL client certificate.
5. The security provider retrieves the SSL user from the certificate.
6. The final step or steps depend on the `jaas-mode` setting in the `<jazn>` element. Refer to "JAAS Authorization and JAAS Mode" on page 2-7 and "Tasks for JAAS Mode and Authorization" on page 5-6 for information about how JAAS mode works.

Using SSL with Standalone OC4J

This section describes how to use SSL in a standalone OC4J environment, without Oracle HTTP Server. Use the following steps:

1. Create a keystore.
   a. Change the 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
```

   In this tool:

   - 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:

```
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):
```

   Always press RETURN for the key password. In the OC4J 10.1.3 implementation, the keystore password must be the same as the key entry password.

   The mykeystore file is created in the current directory. The default alias of the key is mykey.

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

See Also:

- For detailed information about the keytool utility:

  http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html

2. If you do not have a `secure-web-site.xml` file, copy the `default-web-site.xml` to *`ORACLE_HOME`*`/j2ee/home/config/secure-web-site.xml` (by convention).

   (Also remember to change the `display-name` setting appropriately.)

3. Update `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" >

      ...

   </web-site>
   ```

   b. Add the following under the `<web-site>` element to define the keystore and password.

   ```
   <ssl-config keystore="your_keystore" keystore-password="your_password" />
   ```

   Where *`your_keystore`* is the path to the keystore—either absolute, or relative to *`ORACLE_HOME`*`/j2ee/home/config` (where the Web site XML file is located)—and *`your_password`* is the keystore password. For example:

   ```
   <!-- Enable SSL -->
   <ssl-config keystore="../../keystore" keystore-password="123456"/>
   ```

   Note: You can hide the password through password indirection, as described in "Creating an Indirect Password" on page 5-2.

   c. Change the `<web-site>` port setting to some available port. For example, `port="4443"`. (To use the default of 443, you have to be a super user.)

   d. Also see "Optional Steps in secure-web-site.xml" below.

   e. Save the changes to `secure-web-site.xml`.

4. Ensure that `server.xml` points to the `secure-web-site.xml` file.

   a. As necessary, uncomment or add the following line in `server.xml`:

   ```
   <web-site path="./secure-web-site.xml" />
   ```

   b. Save the changes to `server.xml`.

5. Stop and restart 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` pointer in the `server.xml` configuration file:

```
<web-site path="./secure-web-site.xml" /> - comment this to remove SSL
<default-site path="./default-web-site.xml" /> - comment this to remove non-SSL
```

(These Web sites must use different ports.)

Optional Steps in secure-web-site.xml

In addition to the steps outlined above for configuring `secure-web-site.xml`, the following may be appropriate as well:

1. 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.

   See Also:

   - "Requesting Client Authentication" on page 11-12 for related information

2. 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.

   Note: Set `shared="true"` in the `<default-web-app>` element if you intend to use HTTPS tunneling.

   See Also:

   - *Oracle Containers for J2EE Configuration and Administration Guide* for more information about shared applications

3. Optionally, set the cookie domain if `shared="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 nondefault 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 nondefault 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" port="4443" 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.

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 HTTP and SSL.

Using SSL with OC4J in Oracle Application Server

This section describes how to use SSL with OC4J in an Oracle Application Server environment using Oracle HTTP Server and managed by Oracle Process Manager and Notification Server (OPMN). This involves the following:

1. Configure OC4J with SSL (mostly as documented earlier for standalone OC4J)
2. Use Oracle HTTP Server with SSL (available by default)
3. Configure AJP over SSL (`mod_oc4j` settings)
4. Configure OPMN to Enable HTTPS and Use SSL

This discussion concludes with sample configuration files.

Configure OC4J with SSL

Configuring OC4J with SSL in an Oracle Application Server environment is largely the same as for standalone OC4J, as covered above in "Using SSL with Standalone OC4J" on page 11-5. Refer there for details; only differences are highlighted here.

1. Create a keystore.
2. Copy `default-web-site.xml` to `secure-web-site.xml` (by convention).
3. Update `secure-web-site.xml` to set `secure="true"` and configure an `<ssl-config>` element. Use `protocol="ajp13"` (instead of "`http`"). Typically, the choice of port defers to OPMN (as indicated by the `port="0"` setting, which is added automatically).
4. Ensure that `server.xml` points to `secure-web-site.xml`.
5. Stop and restart OC4J to initialize `secure-web-site.xml`.

Here is an example:

```
<web-site display-name="OC4J Web Site" protocol="ajp13" port="0" 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" />
</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; `ajp13` with `secure="true"` indicates AJPS protocol.

> **Note:** It is possible to enter a real port here, rather than port 0, if you configure OPMN to not override the setting in this Web site XML file.

Use Oracle HTTP Server with SSL

In the Oracle Application Server 10.1.3 implementation, SSL is enabled by default for Oracle HTTP Server. No special steps are required.

See Also:

- *Oracle HTTP Server Administrator's Guide*
- *Oracle Application Server Administrator's Guide* for additional information about SSL configuration in Oracle Application Server

Configure AJP over SSL

Configuring OC4J to use AJPS involves the following steps:

1. Use Oracle Wallet Manager to create an auto-login wallet (otherwise known as an SSO wallet) to use with Oracle HTTP Server.
2. In Oracle HTTP Server, verify proper SSL settings in the `mod_oc4j.conf` file for secure communication. SSL must be enabled, and you must specify a path to the wallet you created in step 1. (It is not necessary to specify a wallet password here.)

   ```
   Oc4jEnableSSL on
   Oc4jSSLWalletFile wallet_path
   ```

The *wallet_path* value is a directory path to the wallet file, without a file name. (The wallet file name is already known.)

See Also:

- *Oracle HTTP Server Administrator's Guide* for information about mod_oc4j.conf

3. Use the keytool utility to export a certificate from your keystore. (It is assumed you already have a keystore in OC4J from the step of configuring OC4J with SSL, described earlier.)

```
% keytool -export -file cert_file_name -keystore keystore_file_name
```

Where *cert_file_name* is the desired file name for the certificate that is produced, and *keystore_file_name* is the name of the keystore you already created.

You will be prompted for the keystore password, and will receive a message confirming the certificate file name if the command is successful.

4. Use Oracle Wallet Manager to import the generated certificate into your wallet.

See Also:

- For steps 1 and 4, *Oracle Application Server Administrator's Guide* for details about managing wallets and certificates

Configure OPMN to Enable HTTPS and Use SSL

In an Oracle Application Server environment, configuration steps are also required for the OPMN. Update the file *ORACLE_HOME*/opmn/conf/opmn.xml as follows:

1. Under component ID "OC4J", configure the security parameters (wallet information):

```
<ias-component id="OC4J">
   ...
   <category id="security-parameters">
      <data id="wallet-file" value="file:walletfile"/>
      <data id="wallet-password" value="pwd"/>
   </category>
   ...
</ias-component>
```

2. Also under component ID "OC4J", specify AJPS protocol for the Web site:

```
<ias-component id="OC4J">
   ...
   <port id="secure-web-site" range="12501-12600" protocol="ajps"/>
   ...
</ias-component>
```

3. Under component ID "HTTP_Server", confirm SSL is enabled with the default "ssl-enabled" setting. (A setting of "ssl-disabled" would disable it.)

```
<ias-component id="HTTP_Server">
   ...
   <data id="start-mode" value="ssl-enabled"/>
   ...
</ias-component>
```

See Also:

- *Oracle Process Manager and Notification Server Administrator's Guide* for details about OPMN and `opmn.xml`

Sample Configuration Files for SSL

This section presents samples relating to the configuration discussed in the preceding sections.

Sample <web-site> Element

This shows a sample `<web-site>` element from the `secure-web-site.xml` file:

```
<web-site port="0" protocol="ajp13" secure="true">
   <default-web-app application="default" name="defaultWebApp" root="/j2ee" />
   <web-app application="default" name="dms" root="/dmsoc4j" />
   ...
   <ssl-config
        keystore="C:\demotest\j2eetest\tsrc\shiphome\sslfiles\KEYSTORE\keystore"
        keystore-password="welcome1"/>
</web-site>
```

Sample mod_oc4j.conf File

This shows a sample `mod_oc4j.conf` file:

```
<IfModule mod_oc4j.c>

    Oc4jEnableSSL on
    Oc4jSSLWalletFile C:\demotest\j2eetest\tsrc\shiphome\sslfiles\ssl.wlt\default
    Oc4jSSLWalletPassword welcome1

    <Location /oc4j-service>
        SetHandler oc4j-service-handler
        Order deny,allow
        Deny from all
        Allow from localhost ani-pc.us.oracle.com ani-pc
    </Location>

</IfModule>
```

Sample opmn.xml File

This shows sample `opmn.xml` configuration for component IDs "`OC4J`" and "`HTTP_Server`".

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-Xrs -server
               -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
               -Djava.awt.headless=true"/>
         </category>
         <category id="security-parameters">
            <data id="wallet-file" value=
              "file:C:/demotest/j2eetest/tsrc/shiphome/sslfiles/ssl.wlt/default"/>
            <data id="wallet-password" value="welcome"/>
         </category>
         <category id="stop-parameters">
            <data id="java-options" value=
```

```
                "-Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
                 -Djava.awt.headless=true"/>
            </category>
         </module-data>
         <start timeout="600" retry="2"/>
         <stop timeout="120"/>
         <restart timeout="720" retry="2"/>
         <port id="secure-web-site" range="12501-12600" protocol="ajps"/>
         <port id="rmi" range="3201-3300"/>
         <port id="jms" range="3701-3800"/>
         <process-set id="default_island" numprocs="1"/>
      </process-type>
   </ias-component>

   <ias-component id="HTTP_Server">
      <process-type id="HTTP_Server" module-id="OHS">
         <module-data>
            <category id="start-parameters">
               <data id="start-mode" value="ssl-enabled"/>
            </category>
         </module-data>
         <process-set id="HTTP_Server" numprocs="1"/>
      </process-type>
   </ias-component>
```

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 Oracle Application Server 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 "Using SSL with Standalone OC4J" on page 11-5 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 "Using SSL with Standalone OC4J" on page 11-5. 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.

> **Note:** By contrast, to provide a certificate in a standalone OC4J environment, you would set the certificate in the client browser security area if the client is a browser, or programmatically present the client certificate and the certificate chain when initiating the HTTPS connection for a Java client.

During secure communication between the client and OC4J, the following functionality is executed:

- 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.

Resolving Common SSL Problems

This section discusses some common SSL errors and their causes and remedies, followed by a brief discussion of general SSL debugging.

Common SSL Errors and Solutions

The following errors may occur when using SSL certificates:

Keytool Error: java.security.cert.CertificateException: Unsupported encoding

Cause: There is trailing white space, which the `keytool` utility does not allow.

Action: Delete all trailing white space. If the error still occurs, add a newline in your certificate reply file.

Keytool Error: KeyPairGenerator not available

Cause: You are probably using the `keytool` utility from an older JDK.

Action: Use the `keytool` utility 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` utility cannot locate the root CA certificates in your keystore, and therefore cannot build the certificate chain from your server key to the trusted root certificate authority.

Action: Execute the following command:

```
% 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` utility, then execute these commands:

```
% keytool -keystore keystore -genkey -keyalg RSA -alias serverkey
% keytool -keystore keystore -certreq -file my.host.com.csr
```

Get the certificate from the Certificate Signing Request (CSR), then execute the following command:

```
% keytool -keystore keystore -import -file my.host.com.cer -alias serverkey
```

No available certificate corresponds to the SSL cipher suites that are enabled

Cause: Something is wrong with your certificate.

Action: Determine and rectify the problem.

General SSL Debugging

While you are developing in OC4J standalone, you can display verbose debug information from the Java Secure Socket Extension (JSSE) implementation. To get a list of options, start OC4J as follows:

```
% java -Djavax.net.debug=help -jar oc4j.jar
```

Start it as follows to enable full verbosity:

```
% java -Djavax.net.debug=all -jar oc4j.jar
```

This will display the browser request header, server HTTP header, server HTTP body, content length (before and after encryption), and SSL version.

Enabling ORMIS for OC4J

ORMI over SSL (ORMIS) is disabled by default in OC4J, because it is recommended that client and server keystores or Oracle Wallets be created before ORMIS is used.

This section describes the configuration to enable ORMIS with OC4J in a standalone environment, or in a clustered environment in Oracle Application Server. Once these steps are complete, the "`ormis:`" protocol can be used wherever the "`ormi:`" protocol was used previously.

In all, the following topics are discussed:

- Configuring ORMIS for Standalone OC4J
- Configuring ORMIS for OC4J in an Oracle Application Server Environment
- Configuring ORMIS Access Restrictions
- Configuring Clients to Use ORMIS

Configuring ORMIS for Standalone OC4J

ORMIS configuration, and related RMI configuration, requires updates to the `server.xml` file and `rmi.xml` file on each OC4J instance. This section covers the following topics:

- Configure server.xml for the RMI Configuration File Location
- Configure rmi.xml for ORMIS

- Disabling ORMI with ORMIS Enabled

Configure server.xml for the RMI Configuration File Location

To enable ORMIS in an OC4J instance, the first step is to ensure that `server.xml`, the OC4J server configuration file, has an `<rmi-config>` element that specifies the path to `rmi.xml`, the OC4J RMI configuration file.

Specify the path to `rmi.xml` as follows:

```
<rmi-config path="rmi_path" />
```

Because both the `server.xml` file and the `rmi.xml` file are typically in the *ORACLE_HOME*`/j2ee/home/config` directory, the typical value for *rmi_path* is `"./rmi.xml"`.

See Also:

- *Oracle Containers for J2EE Configuration and Administration Guide* for additional information about `server.xml`

Configure rmi.xml for ORMIS

To use ORMIS, take the following steps to define the SSL configuration in `rmi.xml` on each OC4J instance:

1. Use the `ssl-port` attribute in the `<rmi-server>` element to specify the SSL listener port. For example:

   ```
   <rmi-server ... port="23791" ssl-port="23943">
      ...
   </rmi-server>
   ```

   (This also sets the ORMI listener port to 23791.)

   Note: The default RMI port is 23791; the default ORMIS port is 23943.

2. Add an `<ssl-config>` subelement under the `<rmi-server>` element. This is for keystore configuration, as desired, and results in an ORMIS listener (in addition to the non-secure ORMI listener) when OC4J is restarted. There are two techniques, described below. One is to specify a keystore and password; the other is to use an anonymous cipher suite.

   See Also:

   - *Oracle Containers for J2EE Services Guide* for additional information about `rmi.xml`

Using a Keystore and Password The following example sets the SSL port to 23943 and configures OC4J to use Oracle Wallet-based certificates (as well as specifying an RMI log file):

```
<rmi-server xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
               "http://xmlns.oracle.com/oracleas/schema/rmi-server-10_0.xsd"
            port="23791" ssl-port="23943">
   <ssl-config keystore="/wallets/wallet-server-a/ewallet.p12"
               keystore-password="serverkey-a" />
   ...
```

```
   <log>
      <file path="../log/rmi.log" />
   </log>
</rmi-server>
```

The value of the `keystore` attribute specifies the keystore location (absolute path, or path relative to *ORACLE_HOME*`/j2ee/home/config`, where the Web site XML file is located) and file name.

To use a Java keystore instead of an Oracle Wallet, configure the `<ssl-config>` element as follows:

```
<ssl-config keystore="/keystores/keystore_a.jks" keystore-password="serverkey-a"/>
```

When using keystores and passwords, the server keystore must contain the signed certificate of any client that is authorized to connect to OC4J through ORMIS, or contain the root CA-issued certificate of the client.

Using an Anonymous Cipher Suite Alternatively, you can enable ORMIS using anonymous cipher suites. To accomplish this, omit the `keystore` and `keystore-password` attributes from the `<ssl-config>` element:

```
<ssl-config/>
```

In this mode, any ORMIS client can connect to the server without certification checks being performed.

> **Important:** Use this mode judiciously, given that it allows SSL communication without regard for a client's transport-level authenticity.

Disabling ORMI with ORMIS Enabled

In standalone OC4J, ORMI can be disabled while ORMIS is enabled. To do this, set the ORMI port to -1:

```
<rmi-server ... port="-1" ssl-port="23943">
   <ssl-config keystore="keystore" keystore-password="password" />
   ...
</rmi-server>
```

With this configuration, the non-secure ORMI listener will be disabled when OC4J is restarted.

> **Note:** This is not supported for an OPMN-managed OC4J instance.

Configuring ORMIS for OC4J in an Oracle Application Server Environment

To enable ORMIS in a clustered Oracle Application Server environment managed by OPMN, do the following:

1. Generally complete the steps documented for standalone OC4J above, in "Configure server.xml for the RMI Configuration File Location" on page 11-15 and "Configure rmi.xml for ORMIS" on page 11-15. The exception is to *not* set `ssl-port` in the `<rmi-server>` element in `rmi.xml`. This is not required in an OPMN-managed environment; in fact, the OPMN-managed RMIS port will override the `ssl-port` attribute in `rmi.xml`.

2. For each Oracle Application Server instance that belongs to the cluster, update the `opmn.xml` file to add a `<port>` element with the `rmis` port range shown below:

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-server
                -Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
                -Djava.awt.headless=true
                -Dhttp.webdir.enable=false"/>
         </category>
         <category id="stop-parameters">
            <data id="java-options" value=
               "-Djava.security.policy=$ORACLE_HOME/j2ee/home/config/java2.policy
                -Djava.awt.headless=true -Dhttp.webdir.enable=false"/>
         </category>
      </module-data>
      <start timeout="600" retry="2"/>
      <stop timeout="120"/>
      <restart timeout="720" retry="2"/>
      <port id="default-web-site" range="12501-12600" protocol="ajp"/>
      <port id="rmi" range="12401-12500"/>
      <port id="rmis" range="12701-12800"/>
      <port id="jms" range="12601-12700"/>
      <process-set id="default_group" numprocs="1"/>
   </process-type>
   ...
</ias-component>
```

See Also:

- *Oracle Application Server Administrator's Guide* for general information about OPMN and the `opmn.xml` file

Configuring ORMIS Access Restrictions

ORMIS (like ORMI) supports the ability to restrict incoming IP access by defining access control list (ACL) masks, through settings in the `<access-mask>` element and its `<host-access>` and `<ip-access>` subelements in `rmi.xml`.

Access controls can be either exclusive or inclusive:

- In the exclusive mode, access is denied to all IP addresses or hosts except those specifically included. Use `mode="deny"` in `<access-mask>`, then specify which particular hosts or IP addresses to allow by using `mode="allow"` in a `<host-access>` subelement, `<ip-access>` subelement, or both.
- In the inclusive mode, access is available to all IP addresses or hosts except those specifically excluded. Use `mode="allow"` in `<access-mask>`, then specify which particular hosts or IP addresses to deny by using `mode="deny"` in a `<host-access>` subelement, `<ip-access>` subelement, or both.

The following example configures an exclusive mode, allowing access to only `localhost` and `192.168.1.0`. (`255.255.255.0` is the applicable subnet mask.)

```
<rmi-server xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:noNamespaceSchemaLocation=
                "http://xmlns.oracle.com/oracleas/schema/rmi-server-10_0.xsd"
            port="23791" ssl-port="23943">

   <ssl-config keystore="../wallets/wallet-server-a/ewallet.p12"
```

```
                keystore-password="serverkey-a" />

   <access-mask default="deny">
      <host-access domain="localhost" mode="allow"/>
      <ip-access ip="192.168.1.0" netmask="255.255.255.0" mode="allow"/>
   </access-mask>

   ...

</rmi-server>
```

See Also:

- *Oracle Containers for J2EE Servlet Developer's Guide* for additional information about the `<access-mask>` element, which is supported with the same functionality in `orion-web.xml`

Configuring Clients to Use ORMIS

This section discusses the following client-side configurations for ORMIS:

- Specify the Appropriate Java Naming Provider URL
- Specify the Keystore and Password

Specify the Appropriate Java Naming Provider URL

For an application in a standalone OC4J environment, specify the `ormis` protocol in the setting of the `java.naming.provider.url` environment property, which defines the URI of the system and application:

```
java.naming.provider.url=ormis://hostname/appname
```

For an application in an Oracle Application Server (OPMN-managed) environment, specify the `opmn:ormis` protocol:

```
java.naming.provider.url=opmn:ormis://hostname/appname
```

Note: It is not necessary to include a port number in the URL. The protocol determines what port will be used.

Specify the Keystore and Password

To call an EJB over ORMIS, you must also specify the following on the client side, as applicable:

- Path to client keystore (absolute path is recommended)

  This is the location of the client-side keystore, where server certificates have been imported.

- Keystore password

There are two choices for where to specify these settings, in order of precedence:

- As JSSE properties:

  ```
  -Djava.net.ssl.keystore=keystore_path
  -Djava.net.ssl.keyStorePassword=keystore_password
  ```

- As properties in `ejb_sec.properties` (ignored if JSSE property settings are used):

```
oc4j.keyStoreLoc=keystore_path
oc4j.keyStorePass=keystore_password
```

Note: To use `ejb_sec.properties`, place it in the current directory, from which the client Java VM was launched.

12

Oracle HTTPS for Client Connections

This chapter describes the OC4J implementation of HTTPS that provides Secure Sockets Layer functionality to client HTTP connections. This includes using Oracle HTTPS with standard Java Secure Socket Extension (JSSE) features. The following topics are included:

- Oracle HTTPS and Clients
- Overview of Oracle HTTPS Features
- Specifying Default System Properties
- Oracle HTTPS Example
- Using HTTPClient with JSSE

Note: 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 chapter covers only secure communication between OC4J and the client.

This chapter assumes that you have already obtained keys and certificates. For general information about configuring OC4J to use the Secure Sockets Layer, see Chapter 11, "Integration with SSL and ORMIS".

Standalone OC4J supports SSL communication directly between a client and OC4J, using HTTPS. This is discussed in "Using SSL with Standalone OC4J" on page 11-5.

See Also:

- "Requesting Client Authentication" on page 11-12

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 `HTTPClient` package or the Sun Microsystems `java.net` package can easily use Oracle HTTPS to secure client interactions with a server.

Oracle HTTPS has functionality based on 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`.

Important: The Oracle implementation of `HTTPClient` has diverged from the original open source version upon which it was based. The Oracle version should be considered as a distinct product. Even though there are still many similarities, the two are not necessarily compatible with each other.

Note: Oracle `HTTPClient` supports two different SSL implementations: JSSE and OracleSSL. This documentation discusses the two implementations separately.

See Also:

- Documentation for JSSE and the `java.net` package:

  `http://java.sun.com/products/jsse/index.jsp`

  `http://java.sun.com/j2se/1.4.2/docs/api/`

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 12-7 have been added to this class.

See Also:

- `HTTPClient` Javadoc: *Oracle Application Server HTTPClient API Reference*

OracleSSLCredential Class

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
- Accessing Information for Established SSL Connections
- Security-Aware Applications Support
- Support for java.net.URL Framework

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 the OracleSSL and JSSE SSL implementations, each of which supports a number of cipher suites.

The rest of this section discusses the following topics:

- Choosing a Cipher Suite
- SSL Cipher Suites Supported by OracleSSL
- SSL Cipher Suites Supported by JSSE

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 12–1. Note that with NULL encryption, SSL is used only for authentication and data-integrity purposes.

Table 12–1 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 12–2. Note that with NULL encryption, SSL is used only for authentication and data integrity purposes.

Table 12–2 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

Table 12–2 (Cont.) Cipher Suites Supported by JSSE

Cipher Suite	Authentication	Encryption	Hash Function (Digest)
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

Accessing Information for Established SSL Connections

Users can access information regarding established SSL connections using the `getSSLSession()` method in the `HTTPConnection` class of the Oracle `HTTPClient` package. 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 require 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

Support for java.net.URL Framework

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 supported only 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, 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:

- The next section, "Specifying Default System Properties", for information about configuring your client to use JSSE.
- Javadoc for the `java.net.URL` class, at:

  `http://java.sun.com/j2se/1.4.2/docs/api/`
- *Oracle Application Server Administrator's Guide* for information about wallets and Oracle Wallet Manager

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:

- Property javax.net.ssl.KeyStore
- Property javax.net.ssl.KeyStorePassword
- Property Oracle.ssl.defaultCipherSuites

The following sections describe how to set these properties.

Property 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.

Property 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.

Property Oracle.ssl.defaultCipherSuites

For OracleSSL, 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 12–1, " Cipher Suites Supported by OracleSSL" on page 12-4

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.servlet.request.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 example uses a wallet created by Oracle Wallet Manager to set up credential information.

1. First, create the credentials and load the wallet:

   ```
   mycredential = new OracleSSLCredential();
   mycredential.setWallet(wallet_path, password);
   ```

2. After the credentials are created, pass them to your `HTTPConnection` instance (here called `httpsConnection`) through its `setSSLCredential()` method. This method takes the `OracleSSLCredential` instance, created in the first step, as input:

   ```
   httpsConnection.setSSLCredential(mycredential);
   ```

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, as follows:

```
httpsConnection.connect();
```

After the connection is established, the connection information, in this case the server certificate chain, is obtained as follows:

```
peerCerts = (httpsConnection.getSSLSession()).getPeerCertificateChain();
```

Finally the server certificate common name is obtained as follows:

```
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 as follows:

```
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 through 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 as follows:

```
HTTPResponse rsp = httpsConnection.Get("/");
```

Using HTTPClient with JSSE

Oracle Application Server supports HTTPS client connections using the Java Secure Socket Extension (JSSE). A client can configure `HTTPClient` to use JSSE as the underlying SSL provider as follows:

1. Create a truststore using the keytool.
2. Set the truststore property. A client wishing to use JSSE must specify the client truststore location through the `javax.net.ssl.trustStore` property. Unlike OracleSSL, the client is not required to set the `javax.net.ssl.keyStore` property.
3. Obtain the JSSE SSL socket factory (`javax.net.ssl.SSLSocketFactory` instance) by calling the static `SSLSocketFactory.getDefault()` method.
4. Create an `HTTPClient` connection (`HTTPConnection` instance).
5. Configure the `HTTPClient` connection to use the JSSE implementation of SSL. `HTTPClient` can be configured to use JSSE in either of the following ways:
   - (**For each connection**) The client calls the following method on the `HTTPConnection` instance, specifying the JSSE SSL socket factory retrieved by the `getDefault()` method in step 3:

     ```
     void setSSLSocketFactory(SSLSocketFactory factory)
     ```

     In this case, the SSL socket factory is set for only this connection instance. Example 12–1 below demonstrates this technique.
   - (**Entire VM**) The client calls the following static method on the `HTTPConnection` class:

     ```
     void HttpConnection.setDefaultSSLSocketFactory(SSLSocketFactory factory)
     ```

     In this case, the SSL socket factory is set for all connection instances in the Java VM, until the method is called again with a different setting. This method must be called before instantiating any `HTTPConnection` instances that are to be affected.
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.

Example 12–1 Using JSSE with HTTPClient

```
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");

}
```

Notes:

- If no SSL socket factory is specified, JSSE would be used anyway, by default, if OracleSSL classes are not found in the application classpath. If no SSL socket factory is specified and OracleSSL classes *are* found in the classpath, then OracleSSL is used by default.
- The JSSE SSL implementation is not thread-safe; if you must use SSL in a threaded application, use OracleSSL.
- The JSSE implementation of SSL has some subtle differences from the Oracle 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.

See Also:

- For complete information on JSSE:

  http://java.sun.com/products/jsse/

- For details on using the keytool:

  http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html

13

Web Application Security Configuration

This chapter discusses security issues affecting Web applications, covering the following topics:

- Specifying the Authentication Method (auth-method)
- Web Application Security Role Configuration

See Also:

- *Oracle Containers for J2EE Servlet Developer's Guide* for general information about Web applications
- "Synchronization of OracleAS JAAS Provider User Context with Servlet Sessions" on page 6-17 for information relevant when using Oracle Identity Management as the security provider with OracleAS Single Sign-On
- "Introduction to JAAS Mode" on page 2-8 and "Use OracleAS JAAS Provider JAAS Mode" on page 5-8 for information about JAAS mode, which can be used with Web applications
- The following Web site for OC4J "how-to" examples:

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

Specifying the Authentication Method (auth-method)

This section discusses configuration settings to specify the authentication method for Web applications. The following topics are covered:

- Specifying auth-method in web.xml
- Configuring OC4J for OracleAS Single Sign-On
- Using Client-Cert Authentication
- Using Form-Based Authentication
- Using Digest Authentication with Oracle Internet Directory

See Also:

- "Authentication in the OC4J Environment" on page 2-5 for an overview of supported authentication methods

Specifying auth-method in web.xml

To specify a standard authentication method at the Web application level, use the `<login-config>` element in `web.xml`. For example:

```
<web-app ... >
   ...
   <login-config>
      <auth-method>BASIC</auth-method>
      ...
   </login-config>
   ...
</web-app>
```

Table 13–1 shows the supported `<auth-method>` settings in `web.xml`.

Table 13–1 Values for auth-method in web.xml

Setting	Meaning
`BASIC`	The application uses basic authentication.
`DIGEST`	The application uses digest authentication (not supported for an external LDAP provider or custom provider).
`FORM`	The application uses custom form-based authentication (not supported for a custom provider).
`CLIENT-CERT`	The application requires the client to supply its own HTTPS certificate for use with SSL.

Note: For either the file-based provider or Oracle Identity Management, we recommend digest authentication as a more secure solution than basic authentication.

Be aware of the following:

- To use OracleAS Single Sign-On as the authentication method, configure the OC4J `<jazn-web-app>` element as described in the next section, "Configuring OC4J for OracleAS Single Sign-On".
- To use COREid Single Sign-On, as the authentication method, configure the OC4J `<jazn-web-app>` element as described in "Configure COREid SSO in orion-application.xml" on page 10-16.
- Other than for OracleAS Single Sign-On or COREid Single Sign-On, all authentication method configuration should be in the `web.xml` file, not in any OC4J-specific file (which differs in some cases from proprietary functionality in earlier releases, although that functionality is still supported for backward compatibility).
- When you use `DIGEST` with Oracle Identity Management as your security provider, you must take preparatory steps as described in "Using Digest Authentication with Oracle Internet Directory" on page 13-3.
- When you use `FORM`, you can optionally set an OC4J flag for appropriate client-side redirects, as described in "Using Form-Based Authentication" on page 13-4. (This section also discusses standard configuration for form-based authentication.)

- To use CLIENT-CERT, you must also configure the OracleAS JAAS Provider property x509cert.mapping.attribute, as described in "Using Client-Cert Authentication" on page 13-5.

Configuring OC4J for OracleAS Single Sign-On

To use OracleAS Single Sign-On for authentication, set the auth-method attribute to "SSO" in the <jazn-web-app> element (a subelement of the <jazn> element) in the OC4J orion-application.xml file.

Here is a sample entry:

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

Notes:

- You do not need an <auth-method> setting in the web.xml file. Any setting in web.xml would be overridden by the "SSO" setting in orion-application.xml.
- The auth-method="SSO" setting is automatically written to the orion-application.xml file when you specify Oracle Identity Management with single sign-on when deploying an application through Application Server Control.
- The <jazn-web-app> element is also supported in the orion-web.xml file. In the event of conflict, orion-web.xml takes precedence over orion-application.xml for the particular Web application in question.

Using Digest Authentication with Oracle Internet Directory

Before using digest authentication with Oracle Identity Management as your security provider, you must complete the following preparatory steps:

1. Using Oracle Directory Manager, update the Oracle Internet Directory password policy for your realm:
   a. Launch Oracle Directory Manager with the oidadmin command.
   b. In the Oracle Directory Manager "System Objects" window, under "Oracle Internet Directory Servers", look for the appropriate server (if there are more than one).
   c. For the appropriate server, under "Password Policy Management", select "Password Policy" for the appropriate realm that you have configured for the security provider. If your realm is "us", for example, this would be "Password Policy for Realm dc=us,dc=oracle,dc=com".
   d. In the Oracle Directory Manager "Password Policy for Realm..." window, enable Userpassword Reversible Encryption.

2. Create users and assign roles in Oracle Internet Directory. Do *not* do this until you have completed step 1. You can administer users and roles through Delegated Administration Services (DAS).
3. In the OracleAS JAAS Provider configuration, ensure that SSL has not been disabled for LDAP. Under the `<jazn>` element in the bootstrap `jazn.xml` file, be sure that the `ldap.protocol` property does *not* have a setting of "`no-ssl`". (SSL is enabled by default.)

See Also:

- "Overview of Oracle Identity Management and Oracle Internet Directory Tools" on page 3-4
- "Configuring LDAP User and SSL Properties" on page 6-18

Using Form-Based Authentication

This section discusses standard and OC4J-specific aspects of form-based authentication.

Setting Standard Configuration for Form-Based Authentication

A setting of `FORM` requires additional configuration within the `<login-config>` element in `web.xml` to specify the URLs for the login page and error page. There is nothing OC4J-specific about this functionality. Here is an example:

```
<login-config>
   <auth-method>FORM</auth-method>
   ...
   <form-login-config>
      <form-login-page>mylogin.jsp</form-login-page>
      <form-error-page>myerror.jsp</form-error-page>
   </form-login-config>
</login-config>
```

Setting the OC4J Flag for Client-Side Redirects

If you set the `oc4j.formauth.redirect` property to `true` when you start OC4J, then OC4J will execute an appropriate client-side redirect after a user has entered their credentials for form-based authentication, affecting the request URI that is listed in the browser. The property is set as follows:

```
-Doc4j.formauth.redirect=true
```

The default setting is `false`.

With a `true` setting, if a user enters a user name and password with sufficient credentials to pass the form-based authentication, the content of the protected resource will be displayed, and the request URI listed in the browser is the same as the URI that triggered the form-based authentication. (If the form-based authentication does not succeed, the client will instead be redirected to the error page specified in the configuration, described in the preceding section, "Setting Standard Configuration for Form-Based Authentication".)

Without a `true` setting, an OC4J-specific `j_security_check` request URI is listed in the browser after any form-based authentication.

As an example, assume the following resource is protected by form-based authentication:

```
http://myhost:8888/testapp/SecureServlet
```

If `oc4j.formauth.redirect=true` and form-based authentication succeeds, then the `SecureServlet` URI shown above will be listed in the browser when the content of the protected resource is displayed after form-based authentication. Without the `true` flag setting, though, the request URI listed in the browser would be the following:

```
http://myhost:8888/testapp/j_security_check
```

Using Client-Cert Authentication

This section describes how to configure OC4J to authenticate a client through HTTPS, and describes the OC4J logic flow for this authentication method.

Configuring OC4J for Client-Cert Authentication

To use client authentication through HTTPS:

1. Set `<auth-method>` to `CLIENT-CERT` in `web.xml`, as described in "Specifying auth-method in web.xml" on page 13-2.
2. Set the OC4J `x509cert.mapping.attribute` property in the `<jazn>` element of the application `orion-application.xml` file, as necessary.
3. If you use a default realm other than `jazn.com` (the default realm specified in `jazn.xml`), specify that through the `default-realm` attribute in the `<jazn>` element.

For the file-based provider, the default `x509cert.mapping.attribute` value is "`CN`". For Oracle Identity Management (LDAP-based provider) or an external LDAP provider, the default value is "`DN`". Here is an example that explicitly sets it to "`CN`" for the file-based provider, and also specifies a default realm:

```
<orion-application ... >
   ...
   <jazn provider="XML" ... default-realm="myrealm" ... >
      <property name="x509cert.mapping.attribute" value="CN"/>
      ...
   </jazn>
   ...
</orion-application>
```

See Also:

- "Using SSL with Standalone OC4J" on page 11-5, including information about the `needs-client-auth` flag
- "Using SSL with OC4J in Oracle Application Server" on page 11-8

Client-Cert Execution Flow in OC4J

Here is how `CLIENT-CERT` authentication works in OC4J:

1. A user tries to access a protected resource.
2. OracleAS JAAS Provider retrieves the distinguished name of the certificate user from the certificate.
3. According to the value of `x509cert.mapping.attribute`, OracleAS JAAS Provider retrieves the appropriate value from the distinguished name. For example, if the attribute setting is "`CN`", OracleAS JAAS Provider retrieves the common name from the distinguished name.

4. OracleAS JAAS Provider searches for the certificate user in the relevant user repository (such as `jazn-data.xml` for the file-based provider, or Oracle Internet Directory for the LDAP-based provider). The setting of `x509cert.mapping.attribute` determines what is searched for. If the attribute setting is "`CN`", for example, the common name is what is searched for in the user repository.

   Note, however, that the exact behavior differs between the file-based provider and the LDAP-based or an external LDAP provider. If `johndoe` is the common name, for example:

   - For the file-based provider, OracleAS JAAS Provider looks for "`johndoe`" in the repository.
   - For the LDAP-based provider or an external LDAP provider, OracleAS JAAS Provider looks for "`cn=johndoe`" in the repository.

5. OracleAS JAAS Provider loads the certificate principal and its granted roles, and populates a `Subject` instance with this information.
6. Authorization is performed against the subject.

Web Application Security Role Configuration

This section describes role types and how they are mapped together:

- J2EE Security Roles
- Mapping of Application Roles to J2EE Roles
- Definition of JAAS Roles and Users
- OC4J Mapping of J2EE Roles to JAAS Roles

J2EE Security Roles

The J2EE development environment includes a feature for portable security roles defined in the `web.xml` file for servlets and JavaServer Pages. Security roles define a set of resource access permissions for an application. Associating a principal (in this case, a JAAS user) 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. A role is tied to capabilities and constraints through additional standard descriptor elements, such as under the `<security-constraint>` element in `web.xml`:

```
 <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>
```

Mapping of Application Roles to J2EE Roles

So that you do not have to update application code to change role definitions, J2EE provides a way in the `web.xml` file to map from roles defined in your application to roles defined in `web.xml`. This is accomplished through the `<security-role-ref>` element:

```
<security-role-ref>
   <role-name>DEV</role-name>
   <role-link>sr_developer</role-link>
</security-role-ref>
```

The `<role-name>` element indicates a role in the application code. The `<role-link>` element specifies that this application role (`DEV` in the example) should be linked to a J2EE role (`sr_developer`) described by a `<security-role>` element.

In this example, if a servlet running as a user belonging to `sr_developer` invokes the `HttpServletRequest` method `isUserInRole("DEV")`, the method will return `true`. (Whenever the container finds no `<security-role-ref>` element matching a particular security role, the container checks the `<role-name>` value against the entire list of security roles for the application.)

Definition of JAAS Roles and Users

For the file-based provider, JAAS users and roles are defined in the `system-jazn-data.xml` file.

For example, the following configures the `developers` role:

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

OC4J Mapping of J2EE Roles to JAAS Roles

OC4J enables you to map portable J2EE security roles defined in the `web.xml` file to JAAS roles. You can accomplish this through Application Server Control during deployment, as described in "Specifying Security Role Mapping through Application Server Control" on page 5-14. Mappings are reflected in `<security-role-mapping>` settings in the `orion-application.xml` file (for a J2EE application) or `orion-web.xml` file (for a single Web application).

For example, the following maps the J2EE role `sr_developer` to the JAAS role `developers`.

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

This association permits the `developers` role to access resources that are accessible for the `sr_developer` role.

Consider a user `john`, for example, who is a member of the `developers` role. Because this role is mapped to the J2EE role `sr_developer`, `john` has access to the application resources available to the `sr_developer` role.

Note: A `<group>` subelement under a `<security-role-mapping>` element in `orion-application.xml` corresponds to a role in the OracleAS JAAS Provider.

14

EJB Security Configuration

This chapter discusses security issues affecting EJBs, covering the following topics:

- EJB JNDI Security Properties
- Configuring EJB Security
- Permitting EJB RMI Client Access
- Enabling and Configuring Subject Propagation for ORMI

Note that beginning with the OC4J 10.1.3 implementation, the EJB container supports the OracleAS JAAS Provider.

See Also:

- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for general information about EJBs
- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* also for information about EJB 3.0 security annotations
- *Oracle Containers for J2EE Services Guide* for information about ORMI
- "Enabling ORMIS for OC4J" on page 11-14 (in this document) for information about ORMIS
- "Introduction to JAAS Mode" on page 2-8 and "Use OracleAS JAAS Provider JAAS Mode" on page 5-8 for information about JAAS mode, which can be used with EJB applications
- The following Web site for OC4J "how-to" examples:

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

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 client 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. standalone 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

JavaBeans running within the container pass their credentials within the `InitialContext` instance, which is created to look up the remote EJBs.

For example, to pass JNDI security properties within the `Hashtable` environment:

```
Hashtable env = new Hashtable();
env.put("java.naming.provider.url", "ormi://myhost/ejbsamples");
env.put("java.naming.factory.initial",
        "oracle.j2ee.naming.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);
```

> **Note:** `ApplicationClientInitialContextFactory` is in the file `oc4jclient.jar`.

Configuring EJB 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 the Browser
- Authenticating and Authorizing EJB Applications
- Specifying Credentials in EJB Clients
- Configuring Anonymous EJB Lookup

Granting Permissions in the 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

You can define security constraints and J2EE security roles in the EJB deployment descriptor to protect your EJB methods (or in the `orion-application.xml` descriptor for the overall J2EE application). These J2EE security roles must be mapped

to deployment users and roles in the OracleAS JAAS Provider. This security role mapping can be accomplished through Application Server Control during deployment, as described in "Specifying Security Role Mapping through Application Server Control" on page 5-14.

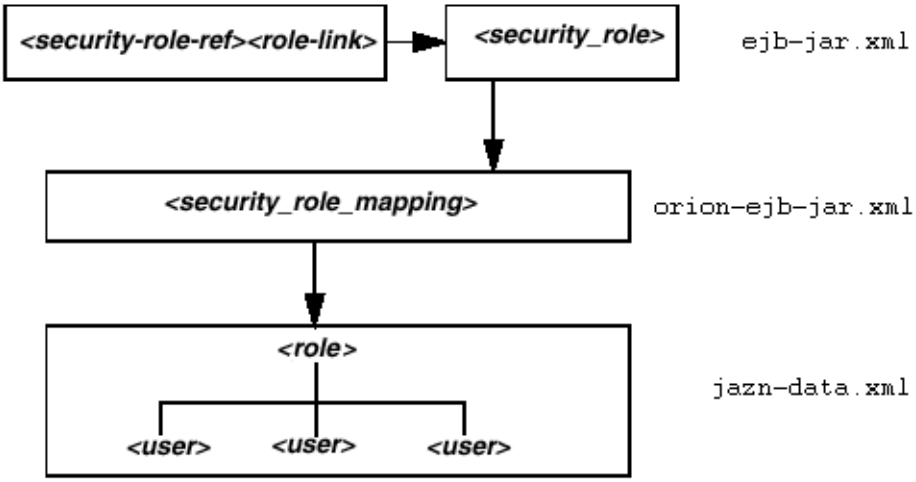
For authentication and authorization, this section focuses on XML configuration within the EJB deployment descriptors. EJB authorization is specified within the EJB and OC4J-specific deployment descriptors. You can manage the authorization piece of your security within the deployment descriptors, as follows:

- The EJB deployment descriptor describes access rules using logical roles.
- The OC4J-specific deployment descriptor maps the logical roles to deployment users and roles, made available to OC4J through the OracleAS JAAS Provider.

Users and roles are identities known by the container. Roles are the logical identities each application uses to indicate access rights to its different objects. The user name / password pairs can be digital certificates and, in the case of SSL, private key pairs.

Thus, the definition and mapping of roles is demonstrated in Figure 14–1.

Figure 14–1 End-to-End Security Role Configuration



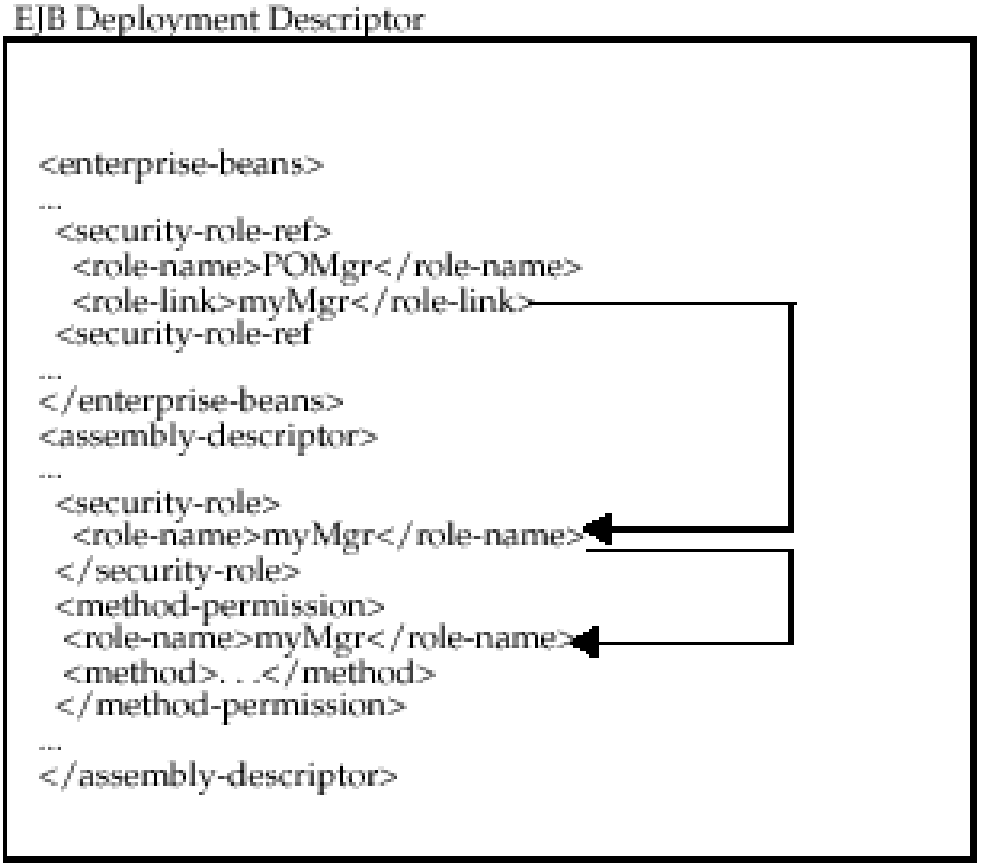
Defining users and roles are discussed in the following sections:

- Specifying Logical Roles in the EJB Deployment Descriptor
- Specifying Unchecked Security for EJB Methods
- Specifying the Run-As Security Identity
- Mapping Logical Roles to Users and Roles
- Specifying a Default Role Mapping for Undefined Methods

Specifying Logical Roles in the EJB Deployment Descriptor

As shown in Figure 14–2, you can use a logical name for a role within your bean implementation, and map this logical name to the correct security 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 Roles" on page 14-7 for more information.

Figure 14–2 Security Role References



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 in a `<security-role-ref>` element. For example, to define a role used within the purchase order example, you may have checked within the bean 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. For the bean to not require awareness 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 specify the logical security role, `POMgr`, in the `<role-name>` subelement of a `<security-role-ref>` 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>` setting 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 required only when you use 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 <ejb-name> element, a subelement of the <session> or <entity> 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> setting 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 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> subelement of <method-permission> 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 input parameters of the method. If the method has no input arguments, the `<method-params>` element contains no subelements.

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 defining a `<role-name>` element, 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 Run-As 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 identity as the security identity.

Specify the "run-as" security identity in the `<security-identity>` element, which is contained in the `<enterprise-beans>` section. The following XML demonstrates that `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 Roles

As noted earlier, you can define security constraints and J2EE security roles in the EJB deployment descriptor to protect your EJB methods. These J2EE security roles must be mapped to deployment users and roles in the OracleAS JAAS Provider. This security role mapping can be accomplished through Application Server Control during deployment, as described in "Specifying Security Role Mapping through Application Server Control" on page 5-14.

Mappings are reflected in `<security-role-mapping>` settings in Oracle-specific descriptors, as shown in the examples that follow.

See Also:

- *Oracle Containers for J2EE Developer's Guide* for information about the `orion-application.xml` file
- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for information about the `orion-ejb-jar.xml` file

Example 14–1 Mapping Logical Role to Actual Role

While we recommend that you use Application Server Control for role mapping, this example provides reference information for the resulting configuration in `orion-ejb-jar.xml` when you map the logical role `POMGR` to the `managers` role. Any user that can log in as part of this role is considered to have the `POMGR` role, and so can execute the methods of `PurchaseOrderBean`.

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

```
</security-role-mapping>
```

For mapping to a specific user:

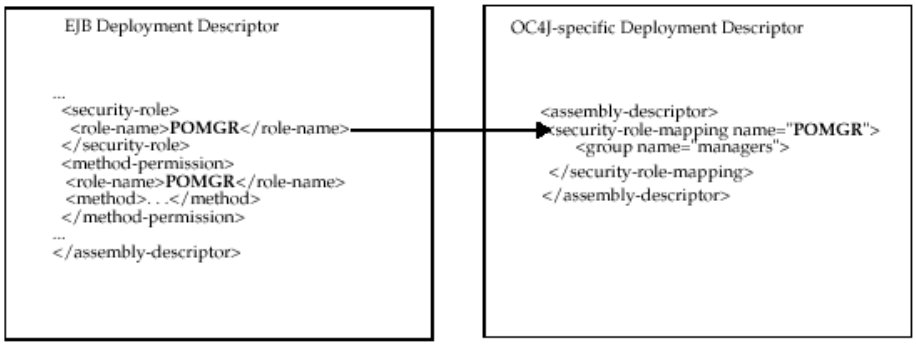
```
<security-role-mapping name="POMGR">
 <user name="guest" />
</security-role-mapping>
```

For mapping to a specific user within a specific role:

```
<security-role-mapping name="POMGR">
 <group name="managers" />
 <user name="guest" />
</security-role-mapping>
```

As shown in Figure 14–3, 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 14–3 Security Role Mapping



Notice that the `<role-name>` setting in the EJB deployment descriptor is the same as for 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 unsecured methods:

```
<default-method-access>
   <security-role-mapping name="&lt;default-ejb-caller-role&gt;"
                          impliesAll="true" />
</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 deployment user or role through the `<user>` and `<group>`

elements. The following example shows how all methods not associated with a method permission are mapped to the "others" role.

```
<default-method-access>
   <security-role-mapping name="default-role" impliesAll="false" />
      <group name="others" />
   </security-role-mapping>
</default-method-access>
```

Specifying Credentials in EJB Clients

When you access EJBs in a remote container, you must pass valid credentials to this container.

- Standalone 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 user name (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 `java.naming.factory.initial` setting indicates that you will use the Oracle JNDI implementation:

```
java.naming.security.principal=POMGR
java.naming.security.credentials=welcome
java.naming.factory.initial=
                    oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://myhost/ejbsamples
```

In your application program, authenticate and access the remote EJBs as shown in the following example:

```
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",
        "oracle.j2ee.naming.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");
```

Configuring Anonymous EJB Lookup

Anonymous EJB lookup is a mode you may consider, presumably only during early development or very special circumstances. In this mode, you do not specify the principal and credential when creating the `InitialContext`, and therefore do not

have to specify a principal or credential to remotely access EJBs. Your `jndi.properties` file would look like this:

```
java.naming.factory.initial=
   oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://localhost:23791/ejb30slsb
java.naming.security.principal=
java.naming.security.credentials=
```

> **Important:** Oracle strongly discourages this practice, except in special circumstances, as it leaves EJBs completely unsecured.

You can enable this mode as follows:

1. Confirm that the `anonymous` user is configured in `system-jazn-data.xml`, and that this user is activated, as described in "Predefined OC4J Accounts" on page 3-11.
2. Also in `system-jazn-data.xml`, under the appropriate realm, assign the `anonymous` user to a role that has been granted RMI permissions, as described in the next section, "Permitting EJB RMI Client Access". For example, assuming the `users` role is granted RMI permissions:

```
<jazn-data>
   ...
   <jazn-realm>
      <realm>
         <name>myrealm</name>
         ...
         <roles>

            <role>
               <name>users</name>
               <members>
                  <member>
                     <type>user</type>
                     <name>anonymous</name>
                  </member>
               </members>
            </role>
            ...
         </roles>
         ...
      </realm>
      ...
   </jazn-realm>
   ...
</jazn-data>
```

3. Give the role (`users` in this example) appropriate namespace access so that it can execute read and write operations on the EJB. Use configuration such as the following in the `orion-application.xml` file for the application:

```
<orion-application>
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="&lt;jndi-user-role&gt;">
               <group name="administrators" />
```

```
            <group name="users" />
          </security-role-mapping>
        </namespace-resource>
      </read-access>
      <write-access>
        <namespace-resource root="">
          <security-role-mapping name="&lt;jndi-user-role&gt;">
            <group name="administrators" />
            <group name="users" />
          </security-role-mapping>
        </namespace-resource>
      </write-access>
    </namespace-access>
    ...
</orion-application>
```

With this configuration, you can access remote EJBs without specifying principals or credentials.

Permitting EJB RMI Client Access

To enable fat client access to EJBs using RMI, you must grant RMI permission "login" to the appropriate role. You can accomplish this in one of the following ways:

- For the file-based provider, through Application Server Control by selecting the role and checking the "Grant RMI Permission" checkbox. (Also refer to "Create a Role" on page 7-7 or "Edit a Role" on page 7-8.)
- Or through the OracleAS JAAS Provider Admintool:

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

Restart OC4J for changes to take effect.

For an application to access EJBs, you must create the end user `JDOE_ENDUSER` in the `system-jazn-data.xml` file and grant it RMI permission "login".

See Also:

- Appendix C, "OracleAS JAAS Provider Admintool Reference"

Enabling and Configuring Subject Propagation for ORMI

This section discusses subject propagation in OC4J, and documents how to enable it with ORMI. (It is always used with IIOP, in accordance with the CSIv2 specification.) The following topics are covered:

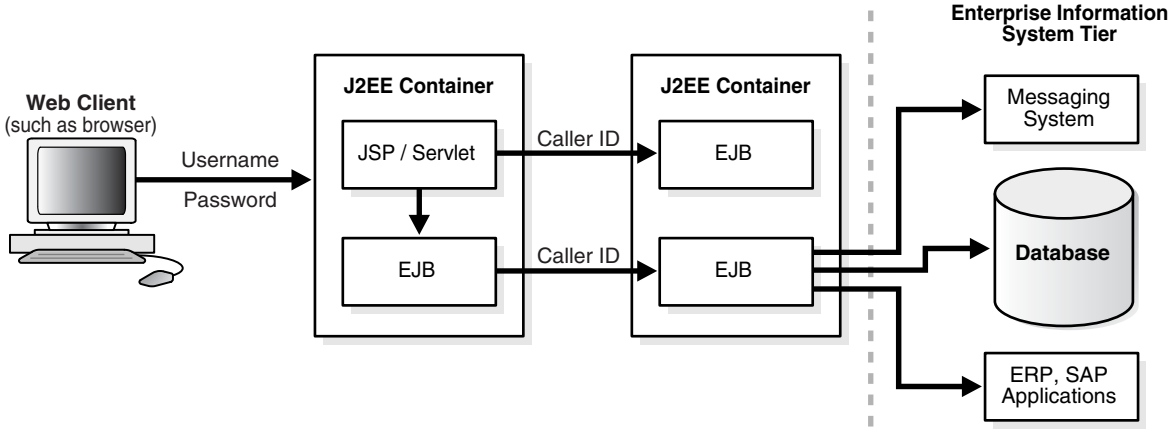
- Overview of Subject Propagation in OC4J
- Enabling Subject Propagation for ORMI
- Sharing Principal Classes for Subject Propagation
- Removing and Configuring Subject Propagation Restrictions

Note: Subject propagation is a powerful feature that should be used only in environments where the server is secure from untrusted client access. It is therefore advised, in order to ensure proper integrity of client requests, that appropriate safeguards be established before this feature is used in production environments. For example, consider using application or network firewalls, RMI access restrictions (through the `<access-mask>` element in `rmi.xml`, as documented in "Configuring ORMIS Access Restrictions" on page 11-17), or RMI subject-propagation restrictions (through the `<subject-propagation-mask>` element in `rmi.xml`, as documented in "Removing and Configuring Subject Propagation Restrictions" on page 14-14).

Overview of Subject Propagation in OC4J

OC4J supports subject propagation, as summarized in Figure 14–4. Through this feature, a Web client can establish its identity to a servlet, and the servlet can then use that identity to communicate with other EJBs and servlets, where the identity is the appropriate JAAS subject (`javax.security.auth.Subject` instance).

Figure 14–4 Subject Propagation



After the client's current subject is obtained, through a `Subject.getSubject()` call, subject propagation works as follows:

1. The subject is serialized over to the RMI server.
2. The RMI server deserializes the subject and uses it to set up the server-side JAAS context.

Subjects may be propagated through a series of EJB invocations, for example. The EJB incorporates the client identity if the EJB is configured to use the client's identity. The EJB cannot be configured to use run-as mode with a specific role.

The authenticated subject (from supplied JNDI credentials) is then merged with the propagated subject to form a combined subject.

Enabling Subject Propagation for ORMI

In OC4J, you can use subject propagation with ORMI if you specifically enable it on the client and server. You can accomplish this with the following system property setting at each end:

```
-Dsubject.propagation=true
```

In the current release, this setting controls subject propagation at a global OC4J level.

Be aware that for subject propagation to work properly, JAAS mode must be enabled for the Web application where the subject is being propagated from, and for the EJB where the subject is being propagated to. So for each, there must be a setting of `jaas-mode="doAs"` or `jaas-mode="doAsPrivileged"` in the `orion-application.xml` file.

See Also:

- "Introduction to JAAS Mode" on page 2-8

Sharing Principal Classes for Subject Propagation

While `java.security.Subject` is a class provided with the JDK, `java.security.Principal` is an interface that you can implement as desired. For subject propagation to work properly with ORMI, you must ensure that the remote client, application, and OC4J all have access to any `Principal` class definitions.

You can accomplish this by putting them in a library that is loaded as an OC4J shared library. There are two main steps to this (considering functionality of the Application Server Control Console in particular):

1. Load the library as an OC4J shared library. From the Application Server Control Console **Administration** tab for the OC4J instance, use the Shared Libraries task.

   This results in configuration such as the following in the OC4J `server.xml` file:

   ```
   <application-server ... >
      ...
      <shared-library name="mylib.jar" version="1.0" library-compatible="true">
         <code-source path="../mypath" />
      </shared-library>
      ...
   </application-server>
   ```

2. Import the library into your application. In deploying an application through Application Server Control, when you reach the Deploy: Deployment Settings page (as discussed in "Deploying an Application through Application Server Control" on page 5-11), you have the opportunity to import shared libraries.

   This results in configuration such as the following in your application `orion-application.xml` file:

   ```
   <orion-application ... >
      ...
      <imported-shared-libraries>
         <import-shared-library name="mylib.jar" />
         ...
      </imported-shared-libraries>
      ...
   </orion-application>
   ```

Note: This is the preferred way to use shared libraries in OC4J; however, the `<library>` element and *ORACLE_HOME*`/j2ee/home/applib` directory are still supported.

See Also:

- *Oracle Containers for J2EE Developer's Guide* for more information about OC4J class loading and shared libraries

Removing and Configuring Subject Propagation Restrictions

By default, access to subject propagation is denied to all ORMI clients. You can configure desired access through settings in the `<subject-propagation-mask>` element and its `<host-access>` and `<ip-access>` subelements in `rmi.xml`.

Subject propagation access can be either exclusive or inclusive:

- In the exclusive mode, access is denied to all IP addresses or hosts except those specifically included. Use `mode="deny"` in `<subject-propagation-mask>`, then specify which particular hosts or IP addresses to allow by using `mode="allow"` in a `<host-access>` subelement, `<ip-access>` subelement, or both.
- In the inclusive mode, access is available to all IP addresses or hosts except those specifically excluded. Use `mode="allow"` in `<subject-propagation-mask>`, then specify which particular hosts or IP addresses to deny by using `mode="deny"` in a `<host-access>` subelement, `<ip-access>` subelement, or both.

The following example configures an exclusive mode, allowing subject propagation for only `localhost` and `192.168.1.0`. (`255.255.255.0` is the applicable subnet mask.)

```
<rmi-server ... >
   ...
   <subject-propagation-mask default="deny">
      <host-access domain="localhost" mode="allow"/>
      <ip-access ip="192.168.1.0" netmask="255.255.255.0" mode="allow"/>
   </subject-propagation-mask>
   ...
</rmi-server>
```

The default setting is as follows:

```
<subject-propagation-mask default="deny"/>
```

15

Common Secure Interoperability Protocol

OC4J supports the Common Secure Interoperability Version 2 protocol (CSIv2). CSIv2 specifies different conformance levels; OC4J complies with the EJB specification, which requires conformance level 0.

This chapter covers the following topics:

- EJB Server Security Properties in internal-settings.xml
- EJB Client Security Properties in ejb_sec.properties
- Introduction to CSIv2 Security Properties
- CSIv2 Security Properties in internal-settings.xml
- CSIv2 Security Properties in ejb_sec.properties
- CSIv2 Security Properties in orion-ejb-jar.xml

> **Note:** If your application uses JAAS, you must configure the OracleAS JAAS Provider to use CSIv2. See information about the `supportCSIv2` option in "Configuring RealmLoginModule" on page 8-2.

EJB Server Security Properties in internal-settings.xml

Specify server security properties in `internal-settings.xml`.

> **Note:** You cannot update `internal-settings.xml` with the Application Server Control.

This file specifies certain properties as values within `<sep-property>` entities. Table 15–1 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 15–1 EJB Server Security Properties

Property	Meaning
`port`	IIOP port number (defaults to `5555`).
`ssl`	A `true` setting indicates IIOP/SSL is supported.

Table 15–1 (Cont.) EJB Server Security Properties

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 and path of the keystore (used only if `ssl` is `true`). An absolute path is recommended.
`keystore-password`	Keystore password (used only if `ssl` is `true`).
`trusted-clients`	Comma-delimited 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 example, `*.example.com`), or `*` (where "`*`" alone means that all clients are trusted). The default is to trust no clients.
`truststore`	Name and path of the truststore. An absolute path is recommended. 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 (used only if `ssl` is `true`).

- If OC4J is started by the Oracle Process Manager and Notification Server (OPMN) in an Oracle Application Server environment, then ports specified in `internal-settings.xml` are overridden. Note that IIOP SSL ports may fail to start if the keystore or truststore location or password is missing or incorrect. In such a case, you are advised to look at the appropriate OPMN log file to see the exact nature of the failure.
- If OPMN is configured to disable IIOP for a particular OC4J instance, then, even though IIOP may be enabled through `internal-settings.xml` (as pointed to by `server.xml`), IIOP is not enabled.
- Keystore and truststore settings are supported in `internal-settings.xml` for both standalone OC4J and a full Oracle Application Server environment. In Oracle Application Server, there are no OPMN options to set these values, so they must be configured manually.

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="true" />
    <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 here 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*)>
```

See Also:

- "CSIv2 Security Properties in internal-settings.xml" on page 15-4

EJB Client Security Properties in ejb_sec.properties

Any client, whether running inside a server or not, has EJB security properties. Table 15–2 lists the EJB client security properties controlled by the `ejb_sec.properties` file. By default, OC4J searches for this file in the current directory when running as a client, or in *`ORACLE_HOME`*`/j2ee/home/config` when running in the server. You can specify the location of this file explicitly with the system property setting `-Dejb_sec_properties_location=`*`pathname`*.

Table 15–2 EJB Client Security Properties

Property	Meaning
`# oc4j.iiop.keyStoreLoc`	The path and name of the keystore. An absolute path is recommended.
`# oc4j.iiop.keyStorePass`	The password for the keystore.
`# oc4j.iiop.trustStoreLoc`	The path name and name of the truststore. An absolute path is recommended.
`# 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`

Table 15–2 (Cont.) EJB Client Security Properties

Property	Meaning
`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. This has no effect if `client.sendpassword` is set to `false`. The list is comma-delimited. Each entry in the list can be an IP address, a host name, a host name pattern (for example, `*.example.com`), or `*` (where "`*`" alone means that all servers are trusted.

Notes:

- The `ejb_sec.properties` file is used for client-side operations, where `internal-settings.xml` would not be present.
- The properties marked with # can be set either in `ejb_sec.properties` or as system properties. The settings in `ejb_sec.properties` override settings specified as system properties.

See Also:

- "CSIv2 Security Properties in ejb_sec.properties" on page 15-5

Introduction to CSIv2 Security Properties

CSIv2 is an Object Management Group (OMG) standard for a secure interoperable wire protocol that supports authorization and identity delegation. Configure CSIv2 properties in three different locations:

- `internal-settings.xml` (server side)
- `ejb_sec.properties` (client side)
- `orion-ejb-jar.xml`

These configuration files are discussed in "CSIv2 Security Properties in internal-settings.xml" (the next section), "CSIv2 Security Properties in ejb_sec.properties" on page 15-5, and "CSIv2 Security Properties in orion-ejb-jar.xml" on page 15-6.

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 the syntax, see "EJB Server Security Properties in internal-settings.xml" on page 15-1.

To use the CSIv2 protocol with OC4J, 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 `ssl-port`, then no CSIv2 component tag is created, even if you configure an `<ior-security-config>` entity in `orion-ejb-jar.xml`.

When IIOP/SSL is enabled on the server, OC4J listens on two different sockets—one for server authentication alone and one for server and client authentication. Specify the server authentication port number within the `<sep-property>` element. OC4J adds 1 to this for the server and client authentication port number.

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

This section discusses usage of the `ejb_sec.properties` file for CSIv2 security settings. See "EJB Client Security Properties in ejb_sec.properties" on page 15-3 for general information about this file and its syntax.

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 DN from the client's certificate and then looks it up in the corresponding security provider; it does not perform password authentication.

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 15–2, " EJB Client Security Properties" on page 15-3 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 15-1 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 specify a list of trusted client (or intermediate) hosts that are allowed to insert identity assertions.

OC4J supports both kinds of trust. Configure trust using the bean `<ior-security-config>` element in `orion-ejb-jar.xml`. See the next section, "CSIv2 Security Properties in orion-ejb-jar.xml", for details.

CSIv2 Security Properties in orion-ejb-jar.xml

This section discusses the CSIv2 security properties for an EJB. Configure the CSIv2 security policies of each individual bean in its `orion-ejb-jar.xml` file. The CSIv2 security properties are specified within `<ior-security-config>` elements. Each element contains a `<transport-config>` subelement, an `<as-context>` subelement, and a `<sas-context>` subelement.

The DTD for the `<ior-security-config>` element is as follows:

```
<!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) >
```

The rest of this section covers the following elements:

- The <transport-config> element
- The <as-context> element
- The <sas-context> element

The <transport-config> element

This element specifies the transport security level. Each subelement under `<transport-config>` must be set to `supported`, `required`, or `none`. The setting `none` means that the bean neither supports nor uses that feature; `supported` means that the bean permits the client to use the feature; `required` means that the bean insists that the client use the feature. The subelements 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? This element may be set to either supported or none; it cannot be set to required.
- <establish-trust-in-client>: Does the client authenticate itself to the server?

Notes:

- If you set <establish-trust-in-client> to required, this overrides setting <auth-method> to username_password under <as-context>. If you do this, you must also set the <required> element 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. Its subelements are:

- <auth-method>: Must be set to either username_password or none. If it is set to username_password, beans use user names and passwords to authenticate the caller.
- <realm>: Must be set to default in the current implementation.
- <required>: If this is 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 subelement, <caller-propagation>, which can be set to supported, required, or none, as follows:

- If it is set to supported, the bean accepts delegated identities from intermediate servers.
- If it is set to required, the bean requires all other beans to transmit delegated identities.
- If it is set to none, the bean does not support identity delegation.

Example: <ior-security-config>

The following example uses the <ior-security-config> element and its subelements:

```
<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>
```

16

Security Support for Resource Adapters

This chapter discusses security considerations and how to configure security and authentication when using resource adapters for an enterprise information system (EIS) connection. The following topics are covered:

Overview of Security and Authentication Setup for EIS Connections

To ensure secure interactions between a J2EE application and an EIS, the J2EE Connector Architecture allows application components to associate a security context with connections established to the EIS. To accomplish this, the J2EE Connector Architecture security contract can work in conjunction with standard JAAS. The following sections provide an overview:

Summary of J2EE Connector Architecture Security Contract

The J2EE Connector Architecture security contract, between an application server and a resource adapter, extends the connection management contract with functionality relating to secure connections. The security contract supports standard JAAS interfaces, allowing it to be independent of any particular security framework or mechanism. In particular, the security contract includes features for the following:

- Propagating a security context, or subject, directly from a J2EE component to a resource adapter (for component-managed sign-on)
- Propagating a security context, or subject, from an application server to a resource adapter (for container-managed sign-on)

The security contract supports two particular authentication mechanisms:

- The commonly used basic password mechanism relies on a user name / password pair, contained together in a password credential object. The application server passes this object to the resource adapter for authentication.

- The Kerberos version 5 mechanism ("Kerbv5" for short) is an authentication protocol distributed by the Massachusetts Institute of Technology. This mechanism uses a "generic credential" object that encapsulates credential information such as a Kerberos ticket. The application server passes this object to the resource adapter for verification.

Security contract functionality includes use of the following key interfaces:

- `javax.security.auth.Subject`
- `java.security.Principal`
- `javax.security.auth.spi.LoginModule`
- `javax.resource.spi.security.PasswordCredential`

  This J2EE Connector Architecture class represents a user name / password pair for basic password authentication.

- `org.ietf.jgss.GSSCredential` (in J2SE version 1.4)

  This interface represents a generic credential object for Kerberos version 5 authentication. (This replaces the J2EE Connector Architecture `javax.resource.spi.security.GenericCredential` interface, which is deprecated.)

Note: Reauthentication may be supported in the `ra.xml` file of a resource adapter, through a value of `true` in the `<reauthentication-support>` element. In this case, it is possible for a managed connection to be reused even for a connection request with a security context that differs from the security context with which the managed connection was initially created.

Summary of Component-Managed Versus Container-Managed Sign-On

Sign-on from a J2EE application to an EIS can be managed either by the application component or by the J2EE container (OC4J). Component-managed sign-on must be set up programmatically and does not involve OC4J-specific configuration. Container-managed sign-on can be set up either declaratively, through OC4J-specific configuration without any programming requirements, or programmatically, involving a combination of OC4J-specific configuration and programming requirements. Programmatic container-managed sign-on can use either a principal mapping class or a JAAS login module.

The following list summarizes the options and the type of setup required for component-managed and container-managed sign-on. Bullets at each level represent choices.

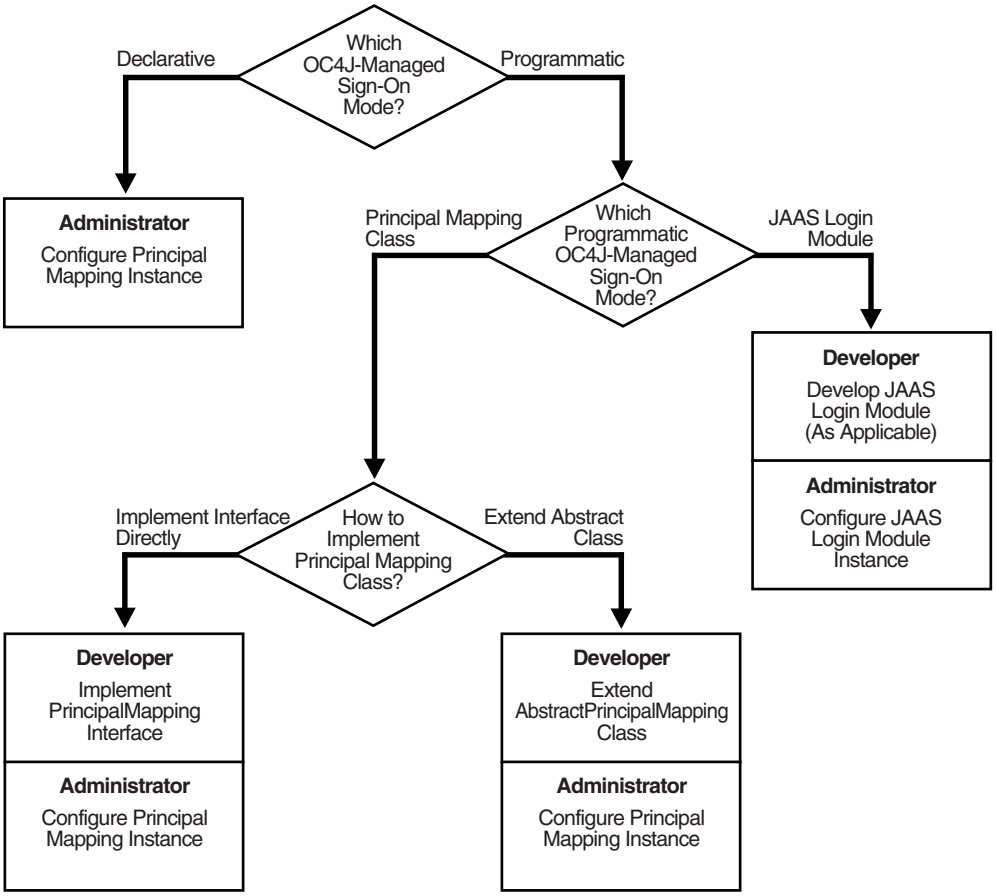
- Component-managed sign-on: Requires `web.xml` or `ejb-jar.xml` `<res-auth>` setting of `Application`; programmatic setup for sign-on; no OC4J-specific configuration.
- Container-managed sign-on: Requires `web.xml` or `ejb-jar.xml` `<res-auth>` setting of `Container`; setup for sign-on may be declarative or programmatic; use OC4J-specific configuration, as follows, for each of the container-managed sign-on modes:
  - None: Implies either component-managed sign-on or no security; specify by disabling security for container-managed sign-on through Application Server Control (as described in "Using Declarative Container-Managed Sign-On" on

page 16-8); reflected as `use="none"` in `<security-config>` element of `oc4j-ra.xml`.

- Declarative: OC4J configuration through principal mapping entries; specify by enabling security for container-managed sign-on through Application Server Control (as described in "Using Declarative Container-Managed Sign-On" on page 16-8); reflected as `use="principal-mapping-entries"` with appropriate subelements in `<security-config>` element of `oc4j-ra.xml`.
- Programmatic, using either a principal mapping class or a JAAS login module:
  * Principal mapping class: Implement `PrincipalMapping` interface directly or extend `AbstractPrincipalMapping` class (both in package `oracle.j2ee.connector`); configure directly through `oc4j-ra.xml` (no Application Server Control support) with `use="principal-mapping-interface"` and appropriate subelements in `<security-config>` element.
  * JAAS login module: Use a JAAS login module; configure directly through `oc4j-ra.xml` (no Application Server Control support) with `use="jaas-module"` and appropriate subelements in `<security-config>` element.

Choices for container-managed sign-on in OC4J are also illustrated in Figure 16–1.

Figure 16–1 Flow Chart of Choices for OC4J Container-Managed Sign-On



Summary of Security-Related Resource Adapter Configuration Elements

This section discusses the following key resource adapter configuration elements for security:

- The oc4j-ra.xml File <security-config> Element
- The oc4j-connectors.xml File <security-permission> Element

For additional information about the files and elements discussed here, refer to the *Oracle Containers for J2EE Resource Adapter Administrator's Guide*.

The oc4j-ra.xml File <security-config> Element

The `oc4j-ra.xml` descriptor provides OC4J-specific deployment information (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. OC4J 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 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:

- Specify mapping subelements explicitly (in the `<principal-mapping-entries>` subelement).
- Specify 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 16-8. This section discusses only the syntax for the `<security-config>` element.

A `<security-config>` element contains one of the following:

- A `<principal-mapping-entries>` element, specifying user names and passwords explicitly
- A `<principal-mapping-interface>` element, specifying the name of the mapping class
- A `<jaas-module>` element, specifying the JAAS module to be used for authentication

The oc4j-connectors.xml File <security-permission> Element

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 OC4J instance, as well as the resource adapters that are embedded within applications. 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>` element contains a `<security-permission-spec>` setting that conforms to the Java 2 Security policy file syntax.

OC4J 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.

```
<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>
```

Understanding Component-Managed Sign-On

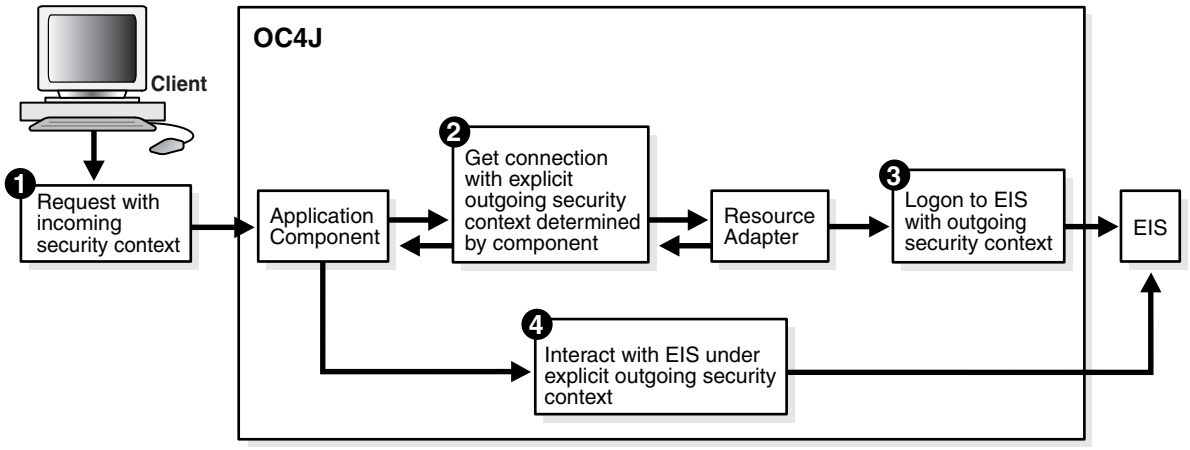
When deploying an application that is to manage its EIS sign-on, use a `<res-auth>` setting of `Application` in the appropriate descriptor file (`web.xml` for a Web component or `ejb-jar.xml` for an EJB component). The application component is then responsible for providing explicit security information for the sign-on. Here is an example:

```
<resource-ref>
   <res-ref-name>...</res-ref-name>
   <res-type>...</res-type>
   <res-auth>Application</res-auth>
   <res-sharing-scope>...</res-sharing-scope>
</resource-ref>
```

No OC4J-specific configuration is required for component-managed sign-on.

Figure 16–2 shows the steps in component-managed sign-on, with the text that follows providing further detail.

Figure 16–2 Component-Managed Sign-On



1. The client makes a request, which is associated with an incoming security context for the initiating principal.
2. As part of servicing the request, the application component maps the incoming security context to an outgoing security context for the resource principal, or hard-codes an outgoing security context, then uses the outgoing security context to request a connection to the EIS.
3. As part of the connection acquisition, the resource adapter signs on to the EIS using the outgoing security context provided by the application component.
4. Once the connection is acquired, the application component can interact with the EIS under the established outgoing security context.

The following example is an excerpt from an application that performs component-managed sign-on:

```
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");
// Assume a custom class ConnectionSpecImpl, used to store sign-on credentials.
com.myeis.ConnectionSpecImpl connSpec = ...
connSpec.setUserName("EISuser");
connSpec.setPassword("EISpassword");
// Pass sign-on credentials through getConnection() method call.
javax.resource.cci.Connection cx = cxf.getConnection(connSpec);
```

Understanding Container-Managed Sign-On

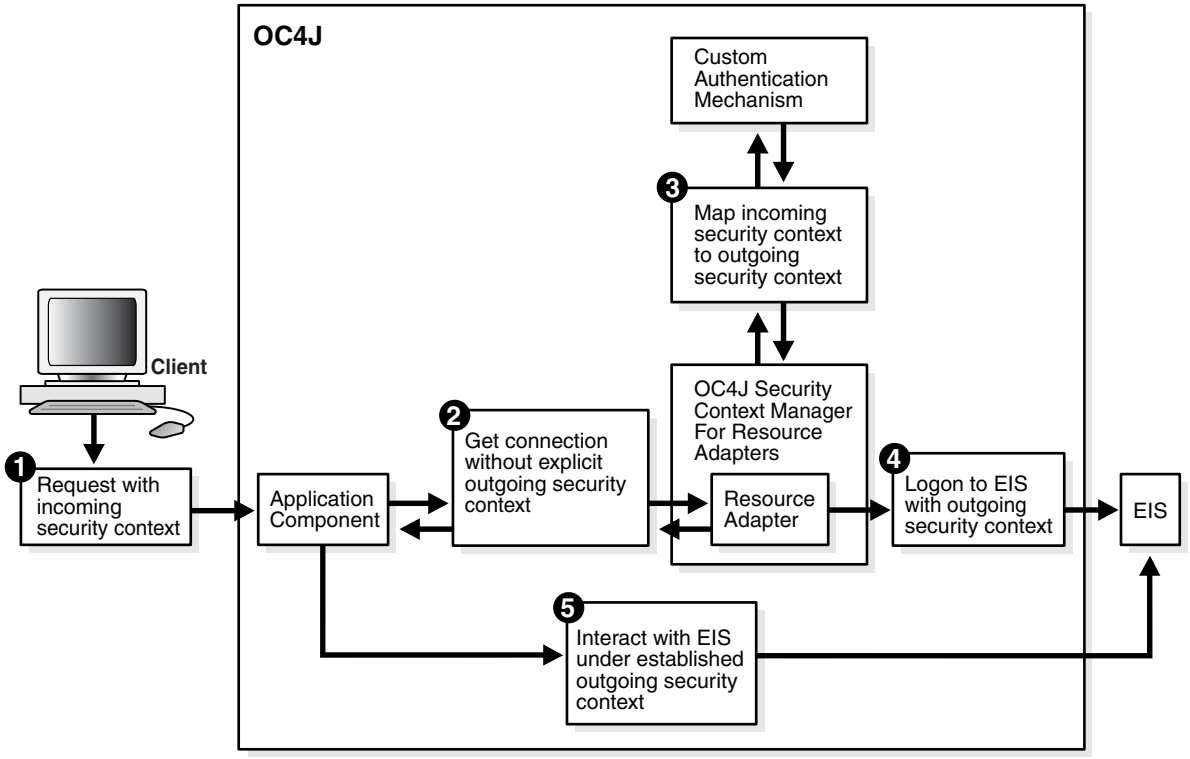
When deploying an application that is to depend on OC4J to manage EIS sign-on, use a `<res-auth>` setting of `Container` in the appropriate descriptor file (`web.xml` for a Web component or `ejb-jar.xml` for an EJB component). OC4J is then responsible for providing security information for the sign-on. Here is an example:

```
<resource-ref>
   <res-ref-name>...</res-ref-name>
   <res-type>...</res-type>
   <res-auth>Container</res-auth>
   <res-sharing-scope>...</res-sharing-scope>
</resource-ref>
```

For declarative container-managed sign-on, OC4J uses configuration information that you specify through Application Server Control, as described in "Using Declarative Container-Managed Sign-On" on page 16-8. For programmatic container-managed sign-on, through either a principal mapping class or a JAAS login module, OC4J uses configuration information that you specify directly through the `oc4j-ra.xml` file. When an application tries to obtain a connection, OC4J uses the applicable mechanism to determine the outgoing security context and to perform authentication.

Figure 16–3 illustrates the steps in container-managed sign-on. These steps are detailed following the diagram.

Figure 16–3 Container-Managed Sign-On



1. The client makes a request, which is associated with an incoming security context for the initiating principal.
2. As part of servicing the request, the application component requests a connection to the EIS.
3. As part of the connection acquisition, the container (the OC4J security context manager shown in Figure 16–3) maps the incoming security context to the outgoing security context for the resource principal. This is based on principal mapping entry elements, a principal mapping class, or a JAAS login module.
4. The resource adapter logs in to the EIS using the outgoing security context provided by OC4J.
5. Once the connection is acquired, the application component can interact with the EIS under the established outgoing security context.

The following example is an excerpt from an application that depends on container-managed sign-on:

```
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();
```

Authentication in Container-Managed Sign-On

When using container-managed sign-on, OC4J 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 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 OC4J 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.

OC4J supports all these methods through JAAS pluggable authentication, user-created authentication classes, or appropriate settings in the `oc4j-ra.xml` file.

Using Declarative Container-Managed Sign-On

This section describes how to set up authentication through OC4J-specific configuration of principal mapping entries. We refer to this as "declarative container-managed sign-on" (as opposed to "programmatic container-managed sign-on"). You can configure this through Application Server Control.

Specify a default resource user and a set of principal mapping entries. Each principal mapping entry specifies an initiating principal and a corresponding resource principal. If the actual initiating principal (OC4J user) during program execution matches one of the initiating principals you specified, then the corresponding resource principal is used for sign-on to the EIS. If the actual initiating principal does not match any you specified, then the default resource user is used for sign-on to the EIS, assuming one is provided or defined. If no default resource user is specified, then a `null` subject will be passed to the EIS. In this case, the EIS has the option of signing on with its own default.

Use the following steps in the Application Server Control Console:

1. From the **Connection Factories** tab of the appropriate Resource Adapter page, choose the connection factory you want to edit.
2. In the Edit Connection Factory page, go to the **Security** tab.

3. Choose to enable security for container-managed sign-on.
4. Specify declarative principal mappings. This is to specify the default resource user.
   a. Specify the default resource user name.
   b. Specify a password for the default resource user. You can choose to do this either indirectly or by typing the desired password in clear text. For an indirect password, specify a key (which might just be the user name, for example). OC4J uses the key to do a lookup in the security provider (such as through the `system-jazn-data.xml` file).
5. Specify initiating user mappings. Specify a mapping for each initiating principal that you want to map to a resource principal. You can edit an existing row or change an existing mapping, or add another row to specify a new mapping. For each mapping:
   a. Specify the initiating user, which is the user name of an initiating principal.
   b. Specify the resource user, which is the user name for a corresponding resource principal.
   c. Specify the resource password, which is a password for the mapped resource principal. As with the default principal mapping, you can choose to do this either indirectly or by typing the password directly.

Note: To get to the Resource Adapter page for a standalone resource adapter:

1. From the OC4J Home page, select the **Applications** tab.
2. View "Standalone Resource Adapters".
3. Select the resource adapter of interest.

To get to the Resource Adapter page for a resource adapter deployed with an application:

1. From the OC4J Home page, select the **Applications** tab.
2. View "Applications".
3. Select the desired application.
4. From the resulting Application Home page, under "Modules", select the resource adapter of interest.

Table 16–1 summarizes how these settings correspond to XML entities in the `oc4j-ra.xml` file. An example follows the table.

Table 16–1 Properties for Declarative Container-Managed Sign-On

Application Server Control Property	Corresponding XML Entity	Description
Enable security for container-managed sign-on	<security-config> element `use` attribute	Being enabled corresponds to `use="principal-mapping-entries"` (assuming declarative container-managed sign-on). Being disabled corresponds to `use="none"`.
Default Resource User	<res-user> subelement of <default-mapping>	User name for the default resource principal.

Table 16–1 (Cont.) Properties for Declarative Container-Managed Sign-On

Application Server Control Property	Corresponding XML Entity	Description
Indirect Password or Password (for Declarative Principal Mappings)	<res-password> subelement of <default-mapping>	Password for the default resource principal, specified either indirectly or directly.
Initiating User	<initiating-user> subelement of <principal-mapping-entry>	User name for an initiating principal that you want to map to a resource principal. This may be a simple user name, or a realm name followed by a slash and the user name.
Resource User	<res-user> subelement of <principal-mapping-entry>	User name for a resource principal that you want to map to an initiating principal. Each initiating-user/resource-user pair uses a separate <principal-mapping-entry> element.
Resource Password	<res-password> subelement of <principal-mapping-entry>	Password for the resource principal, specified either indirectly or directly.

```
<oc4j-connector-factories ... >
   <connector-factory ... >
      ...
      <security-config use="principal-mapping-entries">
         <principal-mapping-entries>
            <default-mapping>
               <res-user>scott</res-user>
               <res-password>->tiger</res-password>
            </default-mapping>
            <principal-mapping-entry>
               <initiating-user>servletuser1</initiating-user>
               <res-user>jmsuser1</res-user>
               <res-password>->jmsuser1</res-password>
            </principal-mapping-entry>
            <principal-mapping-entry>
               <initiating-user>servletuser2</initiating-user>
               <res-user>jmsuser2</res-user>
               <res-password>->jmsuser2</res-password>
            </principal-mapping-entry>
         </principal-mapping-entries>
      </security-config>
   </connector-factory>
   ...
</oc4j-connector-factories>
```

Note: At this release, the initiating user's name can be specified in the `<initiating-user>` element either as a simple name (`scott`) or as a realm name / user name pair separated by a slash, as in `myRealm/scott`. The user name must be a valid OracleAS JAAS Provider user.

In either case, you must specify a OracleAS JAAS Provider default realm. If you supply a simple user name, that name must be a member of the default realm.

Using Programmatic Container-Managed Sign-On

OC4J can manage programmatic authentication, either through an OC4J-specific mechanism that uses a principal mapping class, or through a pluggable JAAS mechanism that uses a JAAS login module. The following sections discuss these mechanisms plus additional features:

- Using a Principal Mapping Class
- Using a JAAS Login Module for an EIS Connection

Using a Principal Mapping Class

One option in OC4J for programmatic container-managed sign-on is to use an Oracle feature that implements principal mapping. The application must include a principal mapping class, which is a class that implements the `oracle.j2ee.connector.PrincipalMapping` interface. A developer can accomplish this by implementing the interface directly, or by extending the `oracle.j2ee.connector.AbstractPrincipalMapping` class, supplied by Oracle for convenience. You must configure a principal mapping class through the `oc4j-ra.xml` file. The following sections describe aspects of using a principal mapping class:

- Understanding the PrincipalMapping Interface APIs
- Extending the AbstractPrincipalMapping Class
- Configuring a Principal Mapping Class

Understanding the PrincipalMapping Interface APIs

Table 16–2 describes how OC4J uses methods of the `PrincipalMapping` interface.

Table 16–2 Method Descriptions for PrincipalMapping Interface

Method Signature	Use by OC4J
void init (java.util.Properties prop)	OC4J calls `init()` to initialize the settings for the `PrincipalMapping` instance, passing in property values specified under the `<principal-mapping-interface>` element in `oc4j-ra.xml`. (See "Configuring a Principal Mapping Class" on page 16-14.) The implementation class can use the properties to set either a default user name and password, information for LDAP connection, or a default mapping.

Table 16–2 (Cont.) Method Descriptions for PrincipalMapping Interface

Method Signature	Use by OC4J
void setManagedConnectionFactory (ManagedConnectionFactory mcf)	OC4J calls `setManagedConnectionFactory()` to provide the `PrincipalMapping` instance with a `ManagedConnectionFactory` instance (for connections to the EIS), which is used in creating a `PasswordCredential` instance.
void setAuthenticationMechanisms (java.util.Map authMechanisms)	OC4J calls `setAuthenticationMechanisms()` to pass the authentication mechanisms supported by the resource adapter to the `PrincipalMapping` instance. The key in the map that is passed is a string containing the supported mechanism type, such as `"BasicPassword"` or `"Kerbv5"`. The value corresponding to the key is a string containing the fully qualified name of the corresponding credentials interface, as declared in a `<credential-interface>` element in `ra.xml`, such as for the `PasswordCredential` interface. The map can contain multiple entries if the resource adapter supports multiple authentication mechanisms.
Subject mapping (Subject initiatingSubject)	OC4J calls `mapping()` to instruct the `PrincipalMapping` instance to perform the principal mapping. A `Subject` instance for the OC4J user (initiating principal) is passed in, and this method returns a `Subject` instance for the resource principal, for use by the resource adapter for sign-on to the EIS. (The implementation may return `null` if the proper resource principal cannot be determined.)

Extending the AbstractPrincipalMapping Class

As a convenience, OC4J provides the abstract class `AbstractPrincipalMapping`, which implements the `PrincipalMapping` interface. This class provides default implementations of the `setManagedConnectionFactory()` and `setAuthenticationMechanism()` methods, as well as utility methods to accomplish the following:

- Retrieve the managed connection factory used for connections to the EIS.
- Retrieve the authentication mechanisms supported by the resource adapter.
- Determine whether the resource adapter supports the basic password authentication mechanism.
- Determine whether the resource adapter supports the Kerberos version 5 authentication mechanism.
- Extract a `Principal` instance from a `Subject` instance.

When extending the `AbstractPrincipalMapping` class, developers need only implement the `init()` and `mapping()` methods.

The methods exposed by the `AbstractPrincipalMapping` class are summarized in Table 16–3.

Table 16–3 Method Descriptions for AbstractPrincipalMapping Class

Method Signature	Description
abstract void init (java.util.Properties prop)	The subclass must implement the `init()` method. See Table 16–2 for a description.
void setManagedConnectionFactory (ManagedConnectionFactory mcf)	The subclass need not implement the `setManagedConnectionFactory()` method. See Table 16–2 for a description.
void setAuthenticationMechanisms (java.util.Map authMechanisms)	The subclass need not implement the `setAuthenticationMechanisms()` method. See Table 16–2 for a description. Note that the subclass can use the `isBasicPasswordSupported()` and `isKerbv5Supported()` methods (described later in this table) to determine which authentication mechanism is supported by the resource adapter. The subclass can also use the `getAuthenticationMechanisms()` method to retrieve the authentication mechanisms.
abstract Subject mapping (Subject initiatingSubject)	The subclass must implement the `mapping()` method. See Table 16–2 for a description.
ManagedConnectionFactory getManagedConnectionFactory ()	The `getManagedConnectionFactory()` utility method returns the `ManagedConnectionFactory` instance (for connections to the EIS), which might be required to create a `PasswordCredential` instance.
java.util.Map getAuthenticationMechanisms ()	The `getAuthenticationMechanisms()` utility method returns a map of all authentication mechanisms supported by the resource adapter. See `setManagedConnectionFactory()` in Table 16–2 for a description of the map.
boolean isBasicPasswordSupported ()	The `isBasicPasswordSupported()` utility method determines whether the basic password authentication mechanism is supported by the resource adapter.
boolean isKerbv5Supported ()	The `isKerbv5Supported()` utility method determines whether the Kerbv5 authentication mechanism is supported by the resource adapter.
Principal getPrincipal (Subject)	The `getPrincipal()` utility method extracts the `Principal` instance from the OC4J user `Subject` instance passed from OC4J. **Note:** In cases where there are multiple principals in a subject (which is not typical), this method would retrieve the first principal. (There is also a `getPrincipals()` method, and the "first" principal is the first element of the collection of principals that this method would return.)

Example 16–1 extends the `AbstractPrincipalMapping` class to provide a principal mapping from the OC4J user to the EIS default user and password. This assumes a default user and password are specified under the `<principal-mapping-interface>` element in `oc4j-ra.xml`, as shown in "Configuring a Principal Mapping Class" on page 16-14.

Example 16–1 Extending AbstractPrincipalMapping

```
package com.example.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 is for 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 incoming Subject
    // (security context), corresponding to the OC4J user.
    // This code is included here only as an example.
    // The principal obtained is not actually used in this example.
    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, as in "Option A" in the
    // J2EE Connector Architecture specification.
    PasswordCredential cred =
      new PasswordCredential(m_defaultUser, resPasswordArray);
    cred.setManagedConnectionFactory(getManagedConnectionFactory());
    initiatingSubject.getPrivateCredentials().add(cred);
    return initiatingSubject;
  }
}
```

Configuring a Principal Mapping Class

To use a principal mapping class, you must update `oc4j-ra.xml` to include a `<principal-mapping-interface>` element for the class. This is a subelement of the `<security-config>` element and must include the following:

- An `<impl-class>` subelement to specify the fully qualified name of the principal mapping class.
- Property settings appropriate to the principal mapping class implementation. For the class shown in the preceding section, there would be a `<property>` subelement with `name="user"` and a `value` setting to specify the default user name for EIS sign-on, and a `<property>` subelement with `name="password"` and a `value` setting to specify the password for the default user, as shown in the following example.

```
<oc4j-connector-factories>
   <connector-factory name="..." location="...">
      ...
      <security-config use="principal-mapping-interface">
         <principal-mapping-interface>
```

```
            <impl-class>com.example.app.MyMapping</impl-class>
            <property name="user" value="scott" />
            <property name="password" value="tiger" />
          </principal-mapping-interface>
        </security-config>
      ...
    </connector-factory>
</oc4j-connector-factories>
```

> **Note:** You can use password indirection to hide the password. For a full discussion of password indirection, see "Guidelines for Password Management" on page 5-1.

Using a JAAS Login Module for an EIS Connection

Alternatively, you can manage sign-on to an EIS programmatically through JAAS.

See Also:

- Chapter 8, "Login Modules"

OC4J 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 vendor, or the resource adapter vendor. Login modules implement the `javax.security.auth.spi.LoginModule` interface.

OC4J provides initiating user subjects to login modules by passing an instance of the `javax.security.auth.Subject` class containing any public certificates and an instance of `oracle.j2ee.connector.InitiatingPrincipal` representing the OC4J user. OC4J can pass a null subject if there is no authenticated user (that is, an anonymous user). The login method of the JAAS login module 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` instance 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` instance is passed, the JAAS login module is responsible for creating a new `Subject` instance containing the resource principal and the appropriate credential.

The InitiatingPrincipal and InitiatingGroup Classes

The `oracle.j2ee.connector.InitiatingPrincipal` class represents OC4J users to a JAAS login module. OC4J creates instances of `InitiatingPrincipal` and

incorporates them into the subject that is passed to the `initialize()` method of a login module. The `InitiatingPrincipal` class implements the `java.security.Principal` interface and adds the method `getGroups()`.

The `oracle.j2ee.connector.InitiatingGroup` class also implements the `Principal` interface, but represents OC4J roles. OC4J creates an `InitiatingPrincipal` instance and incorporates it into the subject that is passed either to the `initialize()` method of a login module, or to the `mapping()` method of a principal mapping class. The `InitiatingPrincipal` class also has a `getGroups()` method.

The `getGroups()` method returns a set (`java.util.Set` instance) of `InitiatingGroup` objects, representing the OC4J roles or OracleAS JAAS Provider roles for this OC4J user. The role membership is defined in an OC4J-specific descriptor file, typically `system-jazn-data.xml`.

Login modules can use `getGroups()` to provide mappings between OC4J roles and EIS users. The `Principal` interface methods support mappings between OC4J users and EIS users. Login modules are not required to refer to the `InitiatingPrincipal` and `InitiatingGroup` classes if they do not provide mappings between OC4J roles 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 a JAAS login module 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/myservice" />
  <security-config>
    <jaas-module>
      <jaas-application-name>JAASModuleDemo</jaas-application-name>
    </jaas-module>
  </security-config>
</connector-factory>
```

With JAAS, you must specify which login module to use for a particular application, and in what order to invoke the login modules. JAAS uses values specified in `<jaas-application-name>` elements to look up login modules.

A

Tips and Troubleshooting for OC4J Security

This appendix discusses best practices for the OC4J security, as well as issues to be aware of and related hints:

- Best Practices for OC4J Security
- OC4J Security Issues and Hints
- Logging

Best Practices for OC4J Security

This section describes best practices in the following areas:

- HTTPS Best Practices
- Overall Security Best Practices
- JAAS Best Practices

HTTPS Best Practices

Oracle HTTP Server 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: Oracle HTTP Server 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: Oracle HTTP Server 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: Oracle HTTP Server 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 example, 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.

  These appliances provide much better solutions than adding mathematics or cryptography cards to UNIX, Windows, or Linux systems.

- *Ensure that sequential HTTPS transfers are requested through the same Web server.* Most CPU time in initiating SSL sessions is spent in the key exchange logic, where the bulk encryption key is exchanged. If the accesses are routed to the same Web server, caching the bulk encryption will significantly reduce CPU overhead on subsequent accesses.

- *Keep secure pages on separate servers from pages not requiring security.* 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 unsecured 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 Best Practices

The following general security practices are recommended.

- *When assigning privileges to modules, use the lowest levels that are adequate to perform the module functions*. 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.* Oracle HTTP Server caches a client's SSL session information by default. With session caching, only the first connection to the server incurs high latency.

  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 the `httpd.conf` file to meet your application needs.

JAAS Best Practices

The following JAAS practices are recommended:

- *Migrate your user management from principals.xml to the OracleAS JAAS Provider.* In earlier releases of Oracle Application Server, the J2EE application server component stored all user information in a file called `principals.xml` (including storing passwords in cleartext). The OracleAS JAAS Provider uses a similar security model as a default, without storing passwords in cleartext. The OracleAS JAAS Provider also offers tight integration with Oracle Application Server infrastructure (including OracleAS Single Sign-On and Oracle Internet Directory) out of the box.

- *Avoid writing custom UserManager classes.* The OC4J container continues to supply several methods and levels of extending security providers. Although you can still implement the `UserManager` interface (deprecated in the 10.1.3 release), leveraging the rich functionality provided by the OracleAS JAAS Provider, OracleAS Single Sign-On, and Oracle Internet Directory gives you more time to

focus on business logic instead of infrastructure code. Both OracleAS Single Sign-On and Oracle Internet Directory provide APIs to integrate with external authentication servers and directories, respectively. If you require custom functionality, you can use a custom login module instead of a custom `UserManager` implementation.

- *Use Oracle Internet Directory as the central repository for the OracleAS JAAS Provider in production environments.* Although the OracleAS JAAS Provider supports a file-based repository, it should be configured to use Oracle Identity Management, which uses Oracle Internet Directory as its repository, for most production environments. Oracle Internet Directory provides standard LDAP features for modeling administrative meta data and is built on the Oracle database platform, inheriting all the database properties of scalability, reliability, manageability, and performance.
- *Use OracleAS Single Sign-On as the authentication mechanism with the OracleAS JAAS Provider.* Various authentication options are available; however, we strongly recommend leveraging the OracleAS Single Sign-On server whenever possible because:
  - It is the default mechanism for most Oracle Application Server components, such as Portal, Forms, Reports, and Wireless.
  - It is easy to set up in a declarative fashion and does not require any custom programming.
  - It provides seamless PKI integration.
- *Use the OracleAS JAAS Provider 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 enables the deployer to use environment-specific security models for that application.
- *Use the fine-grained access control offered by the OracleAS JAAS Provider and the Java permission model.* Unlike the J2EE authorization model as it exists today, the OracleAS 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 access control.
- *Take advantage of the authorization features of the OracleAS 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
  - 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.

OC4J Security Issues and Hints

Be aware of the following issues and how to handle them:

- Failure to Specify OracleAS JAAS Provider as the JAAS Provider
- Realm Issues

File jazn.xml Not Found

Without a valid `jazn.xml` file, the OracleAS JAAS Provider cannot begin running. If no `jazn.xml` file is found, the following error message is generated.

```
"JAZN has not been properly configured"
```

See Also:

- "The jazn.xml File" on page 3-9

Issues for Custom Login Modules

When implementing a custom login module, you should be aware of the following issues:

- Subject-Based Authorization
- J2EE Security Integration

Subject-Based Authorization

When an application uses a custom login module, the subject (and the principals it contains) are used as the sole basis for authorization, including the evaluation of J2EE security constraints. To ensure that all relevant principals are taken into consideration during authorization, the login module should add the relevant principals (including any roles that the authenticated user belongs to) to the subject during the `commit` phase of the authentication process.

J2EE Security Integration

The custom login module framework supports the J2EE declarative security model. That is, the J2EE security constraints declared in application deployment descriptors, such as `web.xml` and `ejb-jar.xml`, are enforced using subject-based authorization.

We encourage J2EE developers to take advantage of the J2EE security model whenever possible, rather than writing their own security implementation; this ensures forward compatibility with future releases.

Issues for Oracle Identity Management

Important issues when troubleshooting the Oracle Identity Management LDAP-based provider include:

- Checking Configuration (JAZN-LDAP)
- Using ldapsearch to Retrieve Realm Names from Oracle Internet Directory
- Avoiding OC4J Restart for Oracle Internet Directory Changes to Take Effect

Checking Configuration (JAZN-LDAP)

To verify that your usage of Oracle Identity Management has been configured properly, do the following:

1. Use Application Server Control to verify that OC4J is associated with an Oracle Internet Directory instance and that the security provider is specified as Oracle Identity Management.

a. Go to the Security Provider page, as described in "Navigating to the Security Provider Page for Your Application" on page 5-17.

b. In the Security Provider page, confirm that the security provider type is listed as Oracle Identity Management Security Provider, and that the host and port listed for Oracle Internet Directory under the security provider attributes are correct.

2. Issue the Admintool `-listrealms` command to verify that data can be retrieved from Oracle Internet Directory.

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

3. If the Admintool responds with the message "`Communication Error`", then it is likely that Oracle Internet Directory is down.

4. If the Admintool responds with the message "`Invalid Credentials`", then the LDAP users and credentials are incorrectly configured.

Note: The Oracle Internet Directory location and port are reflected in the bootstrap `jazn.xml` file.

Using ldapsearch to Retrieve Realm Names from Oracle Internet Directory

In case the OracleAS JAAS Provider Admintool is unavailable, you can use LDAP search commands to retrieve a realm name from Oracle Internet Directory, as follows. (If the Admintool is available, use its `-listrealms` command instead, as shown in the preceding section.)

1. Start with a command such as the following, specifying the port, host, user DN, and password. This will return values for `orclSubscriberNicknameAttribute` and `orclSubscriberSearchbase`.

```
% ldapsearch -p port -h host -D dn_of_user -w password \
        -b "cn=common, cn=products,cn=oraclecontext" -s base "objectclass=*" \
        orclSubscriberNicknameAttribute orclSubscriberSearchbase
```

2. Next, use the values of `orclSubscriberNicknameAttribute` and `orclSubscriberSearchbase` to get the realm name:

```
% ldapsearch -p port -h host -D dn_of_user -w password \
           -b "orclSubscriberSearchbase" \
           -s sub "orclSubscriberNicknameAttribute=*" \
           orclSubscriberNicknameAttribute
```

This will return the Oracle Internet Directory realm, which is useful if you use multiple identity management realms in Oracle Internet Directory and would like to configure a specific nondefault realm for J2EE applications.

See Also:

- *Oracle Identity Management Guide to Delegated Administration*

Avoiding OC4J Restart for Oracle Internet Directory Changes to Take Effect

When doing administration to Oracle Internet Directory, such as adding grantees, permissions, or groups, you should disable LDAP caching. If caching is left enabled, your changes will not take effect until you stop and restart OC4J. See "Configuring LDAP Caching Properties" on page 6-20 for how to disable caching.

Failure to Specify OracleAS JAAS Provider as the JAAS Provider

If you receive an exception and stack trace similar to:

```
Exception in thread "main" java.lang.SecurityException: Unable to locate a login
configuration
at com.sun.security.auth.login.ConfigFile.<init>(ConfigFile.java:97)
at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
at sun.reflect.NativeConstructorAccessorImpl.newInstance
```

You have probably failed to specify the OracleAS JAAS Provider as the JAAS provider.

See Also:

- "Specifying an Alternate JAAS Policy Provider" on page 4-1

Realm Issues

This section discusses the following troubleshooting issues related to the use of realms:

- Realm Names Omitted from User Names
- Specifying Default Realm to Solve Authentication Failure

Realm Names Omitted from User Names

The OC4J property `jaas.username.simple` determines whether realm names are prefixed in user names for returned principals from key methods such as `getUserPrincipal()` or `getRemoteUser()` for servlets, or `getCallerPrincipal()` for EJBs. With the default "`true`" setting, realm names are *not* prefixed.

If you configure and use custom realms, you must explicitly set this property to "`false`" to ensure that OracleAS JAAS Provider authentication and authorization work properly. See "Omitting the Realm Name When Retrieving an Authenticated Principal" on page 5-6 for details.

Specifying Default Realm to Solve Authentication Failure

If authentication fails but your configuration seems correct, confirm whether you need to specify your default realm. You must configure a default realm (in the `<jazn>` element of the `orion-application.xml` file) if you use a default realm other than what is specified in the instance-level `jazn.xml` file.

This can apply to either the file-based provider or LDAP-based provider.

Logging

This section discusses logging features to aid in debugging:

- Using Oracle Diagnostic Logging with OracleAS JAAS Provider
- Using Standard JDK Logging with the OracleAS JAAS Provider Admintool

See Also:

- *Oracle Containers for J2EE Developer's Guide* for an introduction and overview of standard Java logging features
- *Oracle Containers for J2EE Configuration and Administration Guide* for information for about logging configuration and logging levels
- Javadoc for the java.util.logging package:

  http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html

Using Oracle Diagnostic Logging with OracleAS JAAS Provider

OC4J and OracleAS JAAS Provider support the Oracle Diagnostic Logging framework, or ODL, which provides plug-in components that complement the standard Java logging framework to automatically integrate log data with Oracle log analysis tools.

As with OC4J in general, change the logging level in *ORACLE_HOME*/j2ee/home/config/j2ee-logging.xml from the default NOTIFICATION:1 to some appropriate error or debug level. Two levels often used with OracleAS JAAS Provider are FINE and FINER, which correspond to TRACE:1 and TRACE:16, respectively.

OracleAS JAAS Provider logging entries are in *ORACLE_HOME*/j2ee/*instance_name*/logs/oc4j/log.xml, where relevant entries are the ones with a COMPONENT_ID of j2ee and a MODULE_ID of security, as in the following sample message:

```
<MESSAGE>
 <HEADER>
   <TSTZ_ORIGINATING>2005-12-14T11:41:08.974-08:00</TSTZ_ORIGINATING>
   <COMPONENT_ID>j2ee</COMPONENT_ID>
   <MSG_TYPE TYPE="TRACE"></MSG_TYPE>
   <MSG_LEVEL>16</MSG_LEVEL>
   <HOST_ID>www.example.com</HOST_ID>
   <HOST_NWADDR>555.55.5.555</HOST_NWADDR>
   <MODULE_ID>security</MODULE_ID>
   <THREAD_ID>10</THREAD_ID>
   <USER_ID>nmuralid</USER_ID>
 </HEADER>
 <CORRELATION_DATA>
   <EXEC_CONTEXT_ID>
      <UNIQUE_ID>555.55.5.555:30508:1134589268971:0</UNIQUE_ID><SEQ>0</SEQ>
   </EXEC_CONTEXT_ID>
 </CORRELATION_DATA>
 <PAYLOAD>
   <MSG_TEXT>location=system-jazn-data.xml</MSG_TEXT>
 </PAYLOAD>
</MESSAGE>
```

Alternatively, if you want only OracleAS JAAS Provider messages logged in the first place, you can add configuration to j2ee-logging.xml to set the logger name to oracle.j2ee.security, as in the following example:

```
<logger name="oracle.j2ee.security" level="NOTIFICATION:32"
        useParentHandlers="false">
   <handler name="oc4j-handler"/>
   <handler name="console-handler"/>
</logger>
```

Using Standard JDK Logging with the OracleAS JAAS Provider Admintool

OracleAS JAAS Provider supports standard JDK logging. To run logging with the OracleAS JAAS Provider Admintool, change the logging level from `INFO` to `FINE`, `FINER`, or `FINEST`. You can accomplish this either by editing the *`JAVA_HOME`*`/jre/lib/logging.properties` file, or by providing an updated copy of the file on the Admintool command line. The following command executes the Admintool and provides a properties file to set an appropriate logging level. Messages will be logged according to the configured log handler.

```
% java -jar jazn.jar -Djava.util.logging.config.file=modified_logging_properties
```

Note: The `jazn.debug.log.enable` flag, used in previous releases, is no longer supported.

B

OracleAS JAAS Provider Samples

This appendix shows various versions of a sample servlet, first using standard J2EE security APIs, then adding code to manage policy by granting permissions to a user, and finally adding code to check permissions of a user (JAAS mode and JAAS authorization):

- Security Configuration for Sample Servlet
- Sample Servlet: Invoking J2EE Security APIs
- Sample Servlet: Granting Permissions
- Sample Servlet: Checking Permissions

See Also:

- The following Web site for OC4J "how-to" examples:

  `http://www.oracle.com/technology/tech/java/oc4j/1013/how_to/index.html`

Security Configuration for Sample Servlet

The various versions of the sample servlet in this appendix use the file-based provider and depend on the following configurations:

- In `system-jazn-data.xml`, a user `developer` belonging to a role `developers`
- In `web.xml`, a role `sr_developer` and a security constraint for the servlet
- In `orion-application.xml`, a role mapping between `developers` and `sr_developer`

These configurations are shown in the subsections that follow.

Configuration in system-jazn-data.xml

The `system-jazn-data.xml` file defines the `developer` user and the `developers` role to which the user belongs, in the `jazn.com` realm.

The recommended way to define users and roles for the file-based provider is through Application Server Control. You can also use the OracleAS JAAS Provider Admintool.

```
<jazn-data>
   ...
   <jazn-realm>
      <realm>
         <name>jazn.com</name>
```

```
            <users>
               ...
               <user>
                  <name>developer</name>
                  <display-name>developer</display-name>
                  <credentials>{903}CafGQDjOlPMyMiwJEwUfyjhGLAbQkzhR</credentials>
               </user>
               ...
            </users>

            <roles>
               ...
               <role>
                  <name>developers</name>
                  <display-name>Developer Role</display-name>
                  <members>
                     <member>
                        <type>user</type>
                        <name>developer</name>
                     </member>
                  </members>
               </role>
               ...
            </roles>
         </realm>
      </jazn-realm>
      ...
</jazn-data>
```

Configuration in web.xml

The `web.xml` file sets up the security constraint and defines the role `sr_developer`. There is also a setting for the authentication method. (Note that it is possible to override the authentication method in `web.xml` with settings in the `<jazn-web-app>` element in `orion-application.xml`.)

```
<web-app>
   ...
   <security-role>
        <role-name>sr_developer</role-name>
   </security-role>
   ...
   <security-constraint>
      <web-resource-collection>
         <web-resource-name>CallerInfoA</web-resource-name>
         <url-pattern>/callerInfoA</url-pattern>
      </web-resource-collection>
      <!-- authorization -->
      <auth-constraint>
         <role-name>sr_developer</role-name>
      </auth-constraint>
   </security-constraint>
   ...
   <!-- authentication -->
   <login-config>
      <auth-method>BASIC</auth-method>
   </login-config>
   ...
</web-app>
```

Configuration in orion-application.xml

The `orion-application.xml` file specifies the file-based provider, and maps the security role `sr_developer` to the role `developers` that is defined in the identity store (in this case, `system-jazn-data.xml`).

Specify the security provider and security role mappings through Application Server Control.

```
<orion-application>
   ...
   <security-role-mapping name="sr_developer">
      <group name="developers" />
   </security-role-mapping>
   ...
   <!-- use JAZN-XML by default -->
   <jazn provider="XML" />
   ...
</orion-application>
```

Sample Servlet: Invoking J2EE Security APIs

This first version of the servlet uses standard J2EE security APIs to get a user, determine if the user is in a role, and get a user principal.

```
import java.io.IOException;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;


public class CallerInfo extends HttpServlet {

    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();

        response.setContentType("text/html");
        out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
        out.println("Time stamp: " + new Date().toString());
        out.println
           ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
        out.println("request.isUserInRole('ar_developer') = " +
                    request.isUserInRole("sr_developer") + "<br>");
        out.println
            ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");
        out.println("</BODY>");
        out.println("</HTML>");
    }
```

Sample Servlet: Granting Permissions

This version of the servlet adds code to grant permissions to a user. Alternatively, you could use the OracleAS JAAS Provider Admintool to grant permissions.

```
import java.io.*;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;
import oracle.security.jazn.oc4j.*;
import oracle.security.jazn.spi.Grantee;
import oracle.security.jazn.Policy.*;
import javax.security.auth.*;
import java.security.*;

public class CallerInfo extends HttpServlet {

    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        ServletOutputStream out = response.getOutputStream();

        response.setContentType("text/html");
        out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
        out.println("Time stamp: " + new Date().toString());
        out.println
            ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
        out.println("request.isUserInRole('ar_developer') = " +
                    request.isUserInRole("ar_developer") + "<br>");
        out.println
            ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

   //Grant Permissions to a user developer

   //get JAZNConfiguration related info
   JAZNConfig jc = JAZNConfig.getJAZNConfig();

   //create a Grantee for "developer"
   RealmManager realmmgr = jc.getRealmManager();
   Realm realm = realmMgr.getRealm("jazn.com");
   UserManager userMgr = realm.getUserManager();
   final RealmUser user = userMgr.getUser("developer");

   //grant scott file permission
   JAZNPolicy policy = jc.getPolicy();
   if ( policy != null) {
      Grantee gtee = new Grantee( (Principal) user);
      java.io.FilePermission fileperm = new java.io.FilePermission
                                        ("foo.txt","read");
```

```
      policy.grant( gtee, fileperm);
   }

out.println("</BODY>");
   out.println("</HTML>");
}
```

Sample Servlet: Checking Permissions

This version of the servlet adds configuration and code for JAAS mode and JAAS authorization, to check permissions.

JAAS mode controls whether a J2EE application is executed in a `Subject.doAs()` block or a `Subject.doAsPrivileged()` block. Once this mode is set, the authenticated subject is associated with the appropriate access control context. After this, authorization checks may be incorporated into applications using standard JAAS and J2SE APIs.

See Also:

- "Introduction to JAAS Mode" on page 2-8

JAAS Mode Configuration in orion-application.xml

This example expands the previously shown `orion-application.xml` configuration to also set the JAAS mode to "`doasprivileged`". With this setting, OC4J will execute the servlet inside a `Subject.doAsPrivileged()` block.

```
<orion-application>
   ...
   <security-role-mapping name="sr_developer">
      <group name="developers" />
   </security-role-mapping>
   ...
   <!-- use JAZN-XML by default -->
   <jazn provider="XML" jaas-mode="doasprivileged" />
   ...
</orion-application>
```

Servlet Code for Authorization

Here is the servlet code, checking whether the user has permission to read `foo.txt`.

```
import java.io.*;
import java.util.Date;
import java.util.Properties;
import javax.naming.*;
import javax.servlet.*;
import javax.servlet.http.*;

import oracle.security.jazn.*;
import oracle.security.jazn.realm.*;
import oracle.security.jazn.oc4j.*;
import oracle.security.jazn.spi.Grantee;
import oracle.security.jazn.Policy.*;

import javax.security.auth.*;
import java.security.*;

public class CallerInfo extends HttpServlet {
```

```
    public CallerInfo() {
        super();
    }

    public void init(ServletConfig config)
            throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
       ServletOutputStream out = response.getOutputStream();

       response.setContentType("text/html");
       out.println("<HTML><BODY bgcolor=\"#FFFFFF\">");
       out.println("Time stamp: " + new Date().toString());
       out.println
           ("request.getRemoteUser = " + request.getRemoteUser() + "<br>");
       out.println("request.isUserInRole('ar_developer') = " +
                   request.isUserInRole("ar_developer") + "<br>");
       out.println
           ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

      //create Permission
      FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
      {
         //get current AccessControlContext
         AccessControlContext acc = AccessController.getContext();

         javax.security.auth.Policy currPolicy =
                                    javax.security.auth.Policy.getPolicy();

         // Query policy now
         out.println("Policy permissions for this subject are " +
                     currPolicy.getPermissions(Subject.getSubject(acc),null));

         //Check Permissions
         out.println("Policy.impiles permission: "+ perm +" ? " +
         currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));
      }
      out.println("</BODY>");
      out.println("</HTML>");
   }
}
```

C

OracleAS JAAS Provider Admintool Reference

This chapter provides reference information for the OracleAS JAAS Provider Admintool. It is divided into the following sections:

- Authentication to Run the Admintool
- Summary of Admintool Command-Line Syntax and Options
- Admintool Shell
- Admintool Administrative Functions

See Also:

- "Overview of the OracleAS JAAS Provider Admintool" on page 3-3

Authentication to Run the Admintool

Run the Admintool by executing the OracleAS JAAS Provider `jazn.jar` file using the `java -jar` option.

When you run the Admintool, you must authenticate yourself, optionally using the `-user` and `-password` command-line options. You can authenticate yourself in one of two ways:

- The recommended way is to *not* supply `-user` and `-password` settings on the command line; Admintool will then prompt you for a user name and password:

  ```
  % java -jar jazn.jar ...
  AbstractLoginModule username: username
  AbstractLoginModule password: password
  ...
  ```

  In this mode, any options you specify are executed only after you have been prompted for and have supplied the user name and password. For example:

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

  When an example such as this is presented in this appendix, what is left unsaid is that you will be prompted for the user name and password before the command is executed (in this example, before the realms are listed).

- Alternatively, you can use the `-user` and `-password` options on the command line:

  ```
  % java -jar jazn.jar -user username -password password...
  ```

This is generally undesirable, because specifying passwords on command lines creates security vulnerabilities.

In this mode, a command such as the following would immediately list the realms:

```
% java -jar jazn.jar -user myname -password mypassword -listrealms
```

Note: If you specify the `-user` and `-password` options on the command line, they must be positioned before all other command-line options.

In either of these modes, once the options you specify on the command line have been executed, you are returned to your system prompt. To execute any further Admintool commands, you will have to rerun the tool and be authenticated again.

To run multiple commands without reauthenticating, you can use the Admintool shell mode, where you can repeatedly run commands from the Admintool prompt until you exit the shell, as described in "Admintool Shell" on page C-4.

Summary of Admintool Command-Line Syntax and Options

The Admintool provides a number of command options for administrative functions. The general syntax is as follows:

```
% java -jar jazn.jar [-user username -password password] [option1 option2 ... ]
```

Important:

- If you use the `-user` and `-password` options (which is not recommended, as discussed in the preceding section), you must specify them before all other options on the command line.
- Restart OC4J for Admintool changes to take effect.

This section lists all the Admintool command options, with cross-references for further information. You can also list all the options and their syntax with the `-help` option:

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

Command line options are summarized below:

- Administrative option

  `-activateadmin`

  See Also:

  - "Administrative Operations" on page C-10

- Authentication options

  `-user username -password password`

  See Also:

  - "Authentication to Run the Admintool" on page C-1

- Login module options

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

See Also:

- "Adding and Removing Login Modules" on page C-8
- "Listing Login Modules" on page C-12

- Migration option

```
-convert filename realm
```

See Also:

- "Converting from the principals.xml File to JAAS" on page C-14

- Password management options (file-based provider only)

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

See Also:

- "Checking Passwords (File-Based Provider)" on page C-10
- "Setting Passwords (File-Based Provider)" on page C-14

- Policy options

```
-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_class [permission_params]
-revokeperm {realm {-user user|-role role} | principal_class principal_params}
            permission_class [permission_params]
```

See Also:

- "Granting and Revoking Permissions" on page C-11
- "Listing Permissions" on page C-12

- Realm options

```
-addrealm realm admin {adminpwd adminrole | adminrole
  userbase rolebase realmtype }
-addrole realm role
-adduser realm username password
-grantrole role realm {user | -role to_role}
-listrealms realm
-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 Also:

- "Adding and Removing Realms" on page C-9
- "Adding and Removing Roles (File-Based Provider)" on page C-9
- "Adding and Removing Users (File-Based Provider)" on page C-9
- "Granting and Revoking Roles" on page C-12
- "Listing Realms" on page C-13
- "Listing Roles" on page C-13
- "Listing Users" on page C-13

- Shell option

  `-shell`

  See Also:

  - The next section, "Admintool Shell"

Admintool Shell

The Admintool shell provides interactive administration of JAAS principals and policies through a UNIX-like interface. The `-shell` option starts the shell. For example (entering `oc4jadmin` user and password when prompted):

```
% java -jar jazn.jar -shell
AbstractLoginModule username: oc4jadmin
AbstractLoginModule password: password
JAZN>
```

The shell responds with the `JAZN>` prompt. To leave the interface shell, use the `exit` shell command. To see a list of shell commands, use the `help` command. For information about a particular shell command, the shell supports the `man` command:

```
JAZN> man command
```

Note: Multiple-word arguments must be enclosed in quotation marks. For example:

```
% java -jar jazn.jar -user "Oracle DBA" ...
```

The rest of this discussion covers the following topics:

- Shell Support for Admintool Command-Line Options
- Summary of Admintool Special Shell Commands
- Admintool Shell Directory Structure

Shell Support for Admintool Command-Line Options

The Admintool shell supports the same options as the Admintool command line, but you do not have to include the hyphen ("-") in front of the option name. (If you do, it will be ignored.) Once you have launched the Admintool shell, a shell command line such as the following:

```
JAZN> option1 option2 ... optionN
```

Is equivalent to an Admintool command line (from your system prompt) such as the following:

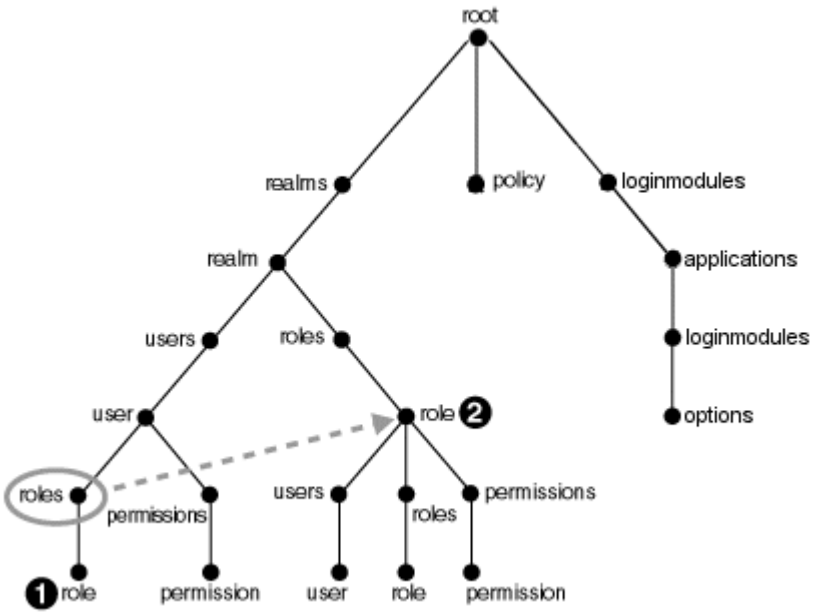
```
% java -jar jazn.jar -option1 -option2 ... -optionN
```

Admintool Shell Directory Structure

The Admintool shell is an interactive interface to the OracleAS JAAS Provider API.

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

Figure C–1 Admintool Shell Directory Structure

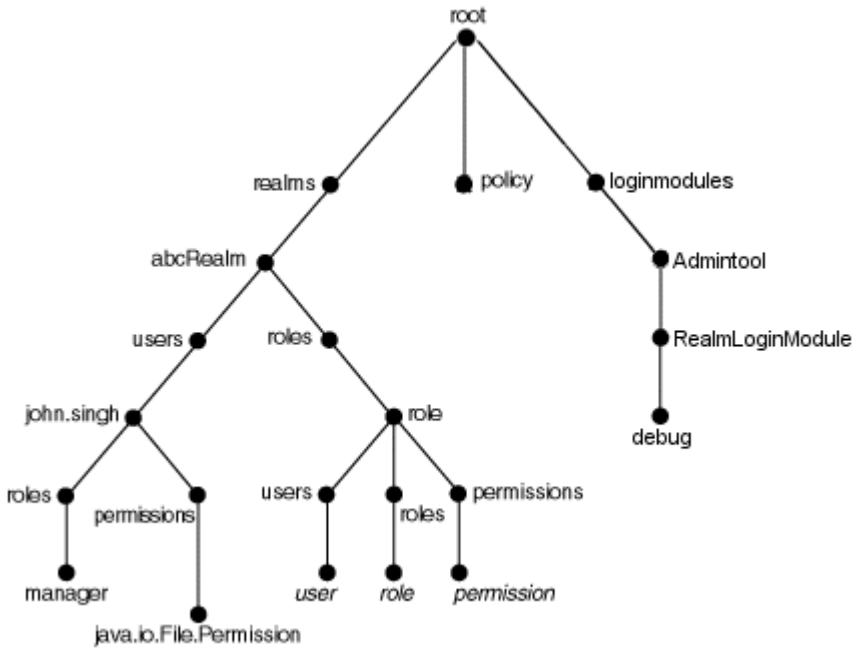


In this structure, the `user` and `role` nodes are linked together. This means that the `roles` link under `user` is the same link as the `roles` link under `realm`. In Unix terms, the `role` at numeral 1 in the diagram is a symbolic link to `role` at numeral 2 in the diagram.

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

Figure C–2 shows nodes of a realm `abcRealm`.

Figure C–2 Sample Shell Directory Structure



Summary of Admintool Special Shell Commands

This section summarizes the following Admintool shell commands:

- add, mkdir, and mk: Creating Provider Data
- cd: Navigating Provider Data
- clear: Clearing the Screen
- exit: Exiting the Admintool Shell
- help: Listing Admintool Shell Commands
- ls: Listing Data
- man: Viewing Admintool man Pages
- pwd: Displaying the Working Directory
- rm: Removing Provider Data
- set: Updating Values

All the Admintool commands support relative and absolute paths.

add, mkdir, and mk: 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 parameters in addition to the name.

cd: Navigating Provider Data

cd *path*

The cd command enables users to navigate the directory tree. Relative and absolute path names are supported.

The path "/" 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 Admintool Shell

exit

The exit command exits the Admintool shell.

help: Listing Admintool Shell Commands

help

The help command displays a list of all valid shell 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 Admintool man Pages

man *command_option*
man *shell_command*

The man command displays detailed usage information for the specified shell command or 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 Administrative Functions

This section documents administrative features of the Admintool. The following topics are covered:

- Adding and Removing Login Modules
- Adding and Removing Realms
- Adding and Removing Roles (File-Based Provider)
- Adding and Removing Users (File-Based Provider)
- Checking Passwords (File-Based Provider)
- Administrative Operations
- Granting and Revoking Permissions
- Granting and Revoking Roles
- Listing Login Modules
- Listing Permissions
- Listing Realms
- Listing Roles
- Listing Users
- Converting from the principals.xml File to JAAS
- Setting Passwords (File-Based Provider)

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 login module for the named application.

The *`control_flag`* must be one of `required`, `requisite`, `sufficient` or `optional`, as specified in the standard `javax.security.auth.login.Configuration` class. The meanings of these flag values are summarized in "Editing a Custom Login Module Configuration during Deployment" on page 8-8.

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

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

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

To delete `MyLoginModule` from `myapp`:

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

Admintool shell:

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

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 file-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
```

The administrator would delete `employees` as follows:

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

Admintool shell:

```
JAZN> addrealm employees martha mypass hr
JAZN> remrealm employees
```

Adding and Removing Roles (File-Based Provider)

-**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.

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

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

To delete the role from the realm:

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

Admintool shell:

```
JAZN> addrole foo fooRole
JAZN> remrole foo fooRole
```

Adding and Removing Users (File-Based Provider)

-**adduser** *realm username password*
-**remuser** *realm username*

The `-adduser` option adds a user to a specified realm; the `-remuser` option deletes a user from the realm.

It is recommended that you add users through the Admintool shell instead of on the command line, as in the following example:

```
% java -jar jazn.jar -shell
AbstractLoginModule username : oc4jadmin
AbstractLoginModule password : adminpassword
JAZN> adduser jazn.com my_user my_password
```

Entering a user on the Admintool command line is less secure. For example, on a UNIX system, any other user on the system could see the password by using the "`ps -ef`" command to list all processes. By contrast, commands entered in the Admintool shell are read only by the Admintool.

However, adding a user on the command line is supported as well. For example, to add the user `martha` to the realm `foo` with the password `mypass`:

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

To insert a user with no password, end the command line with the `-null` option:

```
jazn -jar jazn.jar -adduser foo martha -null
```

To delete `martha` from the realm:

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

Admintool shell:

```
JAZN> adduser foo martha mypass
JAZN> remuser foo martha
```

Checking Passwords (File-Based Provider)

-**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 tis principal" if the user has no password.

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

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

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

Admintool shell:

```
JAZN> checkpasswd foo martha -pw Hello
```

Administrative Operations

-**activateadmin**

Use the `-activateadmin` option to activate the `oc4jadmin` account (formerly `admin`) in the default realm, and to set its password. (This account is initially deactivated for the file-based provider in standalone OC4J.)

```
% java -jar jazn.jar -activateadmin password
```

Admintool shell:

```
JAZN> activateadmin password
```

> **Note:** The `-activateadmin` command is a one-time command. If the administrative account is already active, an error will be thrown to indicate that.

Granting and Revoking Permissions

```
-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_class [permission_params]
-revokeperm {realm {-user user|-role role} | principal_class principal_params}
            permission_class [permission_params]
```

In this syntax, *`principal_class`* is the fully qualified name of a class that implements the principal interface (such as `com.sun.security.auth.NTDomainPrincipal`) and *`principal_params`* 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 the explicit class name of a permission (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.

> **Note:** If the Admintool gives the error message "Permission class not found," it means that the permission you wish to grant is not in the classpath. You must place the JAR containing the permission class in the `jdk/jre/lib/ext` directory so the Admintool can locate it.

For example, to grant `RuntimePermission` to the principal `LDAPPrincipal` (with principal parameter `hobbes` and permission parameter `getProtectionDomain`, values that are understood by `LDAPPrincipal`):

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes
     java.lang.RuntimePermission getProtectionDomain
```

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

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

To revoke the permission:

```
% java -jar jazn.jar -revokeperm foo -user martha java.io.FilePermission
     a.txt read,write
```

Admintool shell:

```
JAZN> grantperm foo -user martha java.io.FilePermission a.txt read,write
JAZN> revokeperm foo -user martha java.io.FilePermission a.txt read,write
```

Granting and Revoking Roles

-**grantrole** *role realm* {*user*|-role *role*}
-**revokerole** *role realm* {*user*|-role *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.

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

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

Or, to grant the role `financial` to the role `finreporter`:

```
% java -jar jazn.jar -grantrole financial foo -role finreporter
```

Admintool shell:

```
JAZN> grantrole editor foo martha
JAZN> revokerole editor foo martha
```

Listing Login Modules

-**listloginmodules** [*application_name*] [*login_module_class*]

The `-listloginmodules` option displays all login modules either in the specified *application_name* or, if no *application_name* is specified, in all applications. Specifying *login_module_class* after *application_name* displays information on only the specified class within the application.

For example, to display all login modules for the application `myapp`:

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

Admintool shell:

```
JAZN> listloginmodules myapp
```

Listing Permissions

-**listperms** {*realm* {-user *user* | -role *role*} | *principal_class principal_params*
permission_class [*permission_params*]

The `-listperms` option displays all permissions that match the list criteria, as follows:

- Permissions that are granted to a user when the `-user` option is used
- Permissions that are granted to a role when a `-role` option is used
- Permissions that are granted to a principal

Important: `PermissionClassManager` and related classes and operations, including `-listperms`, are deprecated in the OC4J 10.1.3 implementation and will be desupported in a future release.

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

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

Admintool shell:

```
JAZN> listperms foo -user martha
```

Listing Realms

-listrealms [*realm*]

The `-listrealms` option displays all realms in the current JAAS environment; or, if a realm argument is specified, the option lists only that realm.

For example, to list all realms:

```
% 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`:

```
% 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`:

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

To list all users in realm `foo` using permission `bar`:

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

The Admintool lists users one to a line, such as:

```
scott
admin
anonymous
```

Admintool shell:

```
JAZN> listusers foo
```

Converting from the principals.xml File to JAAS

-convert *filename realm*

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

```
% java -jar jazn.jar \
       -convert $ORACLE_HOME/j2ee/home/config/principals.xml jazn.com
```

Admintool shell:

```
JAZN> convert ORACLE_HOME/j2ee/home/config/principals.xml jazn.com
```

See Also:

- "Migrating Principals from the principals.xml File" on page 7-15 for important additional information

Setting Passwords (File-Based Provider)

-setpasswd *realm user old_pwd new_pwd*

The `-setpasswd` option enables 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`:

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

Admintool shell:

```
JAZN> setpasswd foo martha mypass a2d3vn
```

D

Third Party Licenses

This appendix includes the Third Party License for third party products included with Oracle Application Server.

Apache

This program contains third-party code from the Apache Software Foundation ("Apache"). Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights.

The Apache license agreements apply to the following included Apache components:

- Apache HTTP Server
- Apache JServ
- mod_jserv
- Regular Expression package version 1.3
- Apache Expression Language packaged in commons-el.jar
- mod_mm 1.1.3
- Apache XML Signature and Apache XML Encryption v. 1.4 for Java and 1.0 for C++
- log4j 1.1.1
- BCEL v. 5
- XML-RPC v. 1.1
- Batik v. 1.5.1
- ANT 1.6.2 and 1.6.5
- Crimson v. 1.1.3
- ant.jar
- wsif.jar
- bcel.jar
- soap.jar
- Jakarta CLI 1.0
- jakarta-regexp-1.3.jar

- JSP Standard Tag Library 1.0.6 and 1.1
- Struts 1.1
- Velocity 1.3
- svnClientAdapter
- commons-logging.jar
- wsif.jar
- commons-el.jar
- standard.jar
- jstl.jar

The Apache Software License

License for Apache Web Server 1.3.29

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 2000-2002 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
```

```
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 *
 * This software consists of voluntary contributions made by many
 * individuals on behalf of the Apache Software Foundation.  For more
 * information on the Apache Software Foundation, please see
 * <http://www.apache.org/>.
 *
 * Portions of this software are based upon public domain software
 * originally written at the National Center for Supercomputing
Applications,
 * University of Illinois, Urbana-Champaign.
```

License for Apache Web Server 2.0

```
Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use
this file except in compliance with the License.  You may obtain a copy of the
License at http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed
under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
CONDITIONS OF ANY KIND, either express or implied.  See the License for the
specific language governing permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation
                                 Apache License
                           Version 2.0, January 2004
                        http://www.apache.org/licenses/

   TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

   1. Definitions.

      "License" shall mean the terms and conditions for use, reproduction,
      and distribution as defined by Sections 1 through 9 of this document.

      "Licensor" shall mean the copyright owner or entity authorized by
      the copyright owner that is granting the License.

      "Legal Entity" shall mean the union of the acting entity and all
      other entities that control, are controlled by, or are under common
      control with that entity. For the purposes of this definition,
      "control" means (i) the power, direct or indirect, to cause the
      direction or management of such entity, whether by contract or
      otherwise, or (ii) ownership of fifty percent (50%) or more of the
      outstanding shares, or (iii) beneficial ownership of such entity.

      "You" (or "Your") shall mean an individual or Legal Entity
      exercising permissions granted by this License.

      "Source" form shall mean the preferred form for making modifications,
      including but not limited to software source code, documentation
      source, and configuration files.

      "Object" form shall mean any form resulting from mechanical
      transformation or translation of a Source form, including but
      not limited to compiled object code, generated documentation,
```

and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

(a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

(b) You must cause any modified files to carry prominent notices stating that You changed the files; and

(c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

(d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

Apache SOAP

This program contains third-party code from the Apache Software Foundation ("Apache"). Under the terms of the Apache license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Apache software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the Apache software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Apache.

Apache SOAP License

Apache SOAP license 2.3.1

Copyright (c) 1999 The Apache Software Foundation. All rights reserved.
TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION
1. Definitions.

   "License" shall mean the terms and conditions for use, reproduction,
   and distribution as defined by Sections 1 through 9 of this document.

   "Licensor" shall mean the copyright owner or entity authorized by
   the copyright owner that is granting the License.

   "Legal Entity" shall mean the union of the acting entity and all
   other entities that control, are controlled by, or are under common
   control with that entity. For the purposes of this definition,
   "control" means (i) the power, direct or indirect, to cause the
   direction or management of such entity, whether by contract or
   otherwise, or (ii) ownership of fifty percent (50%) or more of the
   outstanding shares, or (iii) beneficial ownership of such entity.

   "You" (or "Your") shall mean an individual or Legal Entity
   exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses

granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

   You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS,

```
   WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
   implied, including, without limitation, any warranties or conditions
   of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A
   PARTICULAR PURPOSE. You are solely responsible for determining the
   appropriateness of using or redistributing the Work and assume any
   risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory,
   whether in tort (including negligence), contract, or otherwise,
   unless required by applicable law (such as deliberate and grossly
   negligent acts) or agreed to in writing, shall any Contributor be
   liable to You for damages, including any direct, indirect, special,
   incidental, or consequential damages of any character arising as a
   result of this License or out of the use or inability to use the
   Work (including but not limited to damages for loss of goodwill,
   work stoppage, computer failure or malfunction, or any and all
   other commercial damages or losses), even if such Contributor
   has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing
   the Work or Derivative Works thereof, You may choose to offer,
   and charge a fee for, acceptance of support, warranty, indemnity,
   or other liability obligations and/or rights consistent with this
   License. However, in accepting such obligations, You may act only
   on Your own behalf and on Your sole responsibility, not on behalf
   of any other Contributor, and only if You agree to indemnify,
   defend, and hold each Contributor harmless for any liability
   incurred by, or claims asserted against, such Contributor by reason
   of your accepting any such warranty or additional liability.

END OF TERMS AND CONDITIONS
```

mod_mm and mod_ssl

This program contains third-party code from Ralf S. Engelschall ("Engelschall"). Under the terms of the Engelschall license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the Engelschall software, and the terms contained in the following notices do not change those rights. Notwithstanding anything to the contrary in the Oracle program license, the mod_mm software is provided by Oracle "AS IS" and without warranty or support of any kind from Oracle or Engelschall.

mod_mm

```
Copyright (c) 1999 - 2000 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall
<rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).
```

mod_ssl

```
Copyright (c) 1998-2001 Ralf S. Engelschall.  All rights reserved.
This product includes software developed by Ralf S. Engelschall
<rse@engelschall.com> for use in the mod_ssl project (http://www.modssl.org/).
```

OpenSSL

This program contains third-party code from the OpenSSL Project. Under the terms of the OpenSSL Project license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the OpenSSL software, and the terms contained in the following notices do not change those rights.

OpenSSL License

```
/* ====================================================================
 * Copyright (c) 1998-2005 The OpenSSL Project.  All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. All advertising materials mentioning features or use of this
 *    software must display the following acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit. (http://www.openssl.org/)"
 *
 * 4. The names "OpenSSL Toolkit" and "OpenSSL Project" must not be used to
 *    endorse or promote products derived from this software without
 *    prior written permission. For written permission, please contact
 *    openssl-core@openssl.org.
 *
 * 5. Products derived from this software may not be called "OpenSSL"
 *    nor may "OpenSSL" appear in their names without prior written
 *    permission of the OpenSSL Project.
 *
 * 6. Redistributions of any form whatsoever must retain the following
 *    acknowledgment:
 *    "This product includes software developed by the OpenSSL Project
 *    for use in the OpenSSL Toolkit (http://www.openssl.org/)"
 *
 * THIS SOFTWARE IS PROVIDED BY THE OpenSSL PROJECT "AS IS" AND ANY
 * EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
 * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE OpenSSL PROJECT OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
 * NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
 * LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT,
 * STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE)
 * ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED
 * OF THE POSSIBILITY OF SUCH DAMAGE.
 * ====================================================================
 *
 * This product includes cryptographic software written by Eric Young
 * (eay@cryptsoft.com).  This product includes software written by Tim
```

```
 * Hudson (tjh@cryptsoft.com).
 *
 */

 Original SSLeay License
 -----------------------

/* Copyright (C) 1995-1998 Eric Young (eay@cryptsoft.com)
 * All rights reserved.
 *
 * This package is an SSL implementation written
 * by Eric Young (eay@cryptsoft.com).
 * The implementation was written so as to conform with Netscapes SSL.
 *
 * This library is free for commercial and non-commercial use as long as
 * the following conditions are aheared to.  The following conditions
 * apply to all code found in this distribution, be it the RC4, RSA,
 * lhash, DES, etc., code; not just the SSL code.  The SSL documentation
 * included with this distribution is covered by the same copyright terms
 * except that the holder is Tim Hudson (tjh@cryptsoft.com).
 *
 * Copyright remains Eric Young's, and as such any Copyright notices in
 * the code are not to be removed.
 * If this package is used in a product, Eric Young should be given attribution
 * as the author of the parts of the library used.
 * This can be in the form of a textual message at program startup or
 * in documentation (online or textual) provided with the package.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the copyright
 *    notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in the
 *    documentation and/or other materials provided with the distribution.
 * 3. All advertising materials mentioning features or use of this software
 *    must display the following acknowledgement:
 *    "This product includes cryptographic software written by
 *     Eric Young (eay@cryptsoft.com)"
 *    The word 'cryptographic' can be left out if the rouines from the library
 *    being used are not cryptographic related :-).
 * 4. If you include any Windows specific code (or a derivative thereof) from
 *    the apps directory (application code) you must include an acknowledgement:
 *    "This product includes software written by Tim Hudson (tjh@cryptsoft.com)"
 *
 * THIS SOFTWARE IS PROVIDED BY ERIC YOUNG "AS IS" AND
 * ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE
 * IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE
 * ARE DISCLAIMED.  IN NO EVENT SHALL THE AUTHOR OR CONTRIBUTORS BE LIABLE
 * FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL
 * DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS
 * OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
 * HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT
 * LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY
 * OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 *
 * The licence and distribution terms for any publically available version or
 * derivative of this code cannot be changed.  i.e. this code cannot simply be
```

```
 * copied and put under another distribution licence
 * [including the GNU Public Licence.]
 */
```

Perl

This program contains third-party code from the Comprehensive Perl Archive Network ("CPAN"). Under the terms of the CPAN license, Oracle is required to provide the following notices. Note, however, that the Oracle program license that accompanied this product determines your right to use the Oracle program, including the CPAN software, and the terms contained in the following notices do not change those rights.

Perl Kit Readme

Copyright 1989-2001, Larry Wall

All rights reserved.

This program is free software; you can redistribute it and/or modify it under the terms of either:

1. the GNU General Public License as published by the Free Software Foundation; either version 1, or (at your option) any later version, or
2. the "Artistic License" which comes with this Kit.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See either the GNU General Public License or the Artistic License for more details.

You should have received a copy of the Artistic License with this Kit, in the file named "Artistic". If not, I'll be glad to provide one.

You should also have received a copy of the GNU General Public License along with this program in the file named "Copying". If not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307, USA or visit their Web page on the internet at http://www.gnu.org/copyleft/gpl.html.

For those of you that choose to use the GNU General Public License, my interpretation of the GNU General Public License is that no Perl script falls under the terms of the GPL unless you explicitly put said script under the terms of the GPL yourself. Furthermore, any object code linked with perl does not automatically fall under the terms of the GPL, provided such object code only adds definitions of subroutines and variables, and does not otherwise impair the resulting interpreter from executing any standard Perl script. I consider linking in C subroutines in this manner to be the moral equivalent of defining subroutines in the Perl language itself. You may sell such an object file as proprietary provided that you provide or offer to provide the Perl source, as specified by the GNU General Public License. (This is merely an alternate way of specifying input to the program.) You may also sell a binary produced by the dumping of a running Perl script that belongs to you, provided that you provide or offer to provide the Perl source as specified by the GPL. (The fact that a Perl interpreter and your code are in the same binary file is, in this case, a form of mere aggregation.) This is my interpretation of the GPL. If you still have concerns or difficulties understanding my intent, feel free to contact me. Of course, the Artistic License spells all this out for your protection, so you may prefer to use that.

mod_perl 1.29 License

```
/* ====================================================================
 * The Apache Software License, Version 1.1
 *
 * Copyright (c) 1996-2000 The Apache Software Foundation.  All rights
 * reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 *
 * 1. Redistributions of source code must retain the above copyright
 *    notice, this list of conditions and the following disclaimer.
 *
 * 2. Redistributions in binary form must reproduce the above copyright
 *    notice, this list of conditions and the following disclaimer in
 *    the documentation and/or other materials provided with the
 *    distribution.
 *
 * 3. The end-user documentation included with the redistribution,
 *    if any, must include the following acknowledgment:
 *       "This product includes software developed by the
 *        Apache Software Foundation (http://www.apache.org/)."
 *    Alternately, this acknowledgment may appear in the software itself,
 *    if and wherever such third-party acknowledgments normally appear.
 *
 * 4. The names "Apache" and "Apache Software Foundation" must
 *    not be used to endorse or promote products derived from this
 *    software without prior written permission. For written
 *    permission, please contact apache@apache.org.
 *
 * 5. Products derived from this software may not be called "Apache",
 *    nor may "Apache" appear in their name, without prior written
 *    permission of the Apache Software Foundation.
 *
 * THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED
 * WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES
 * OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
 * DISCLAIMED.  IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR
 * ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
 * SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
 * LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF
 * USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND
 * ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY,
 * OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT
 * OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF
 * SUCH DAMAGE.
 * ====================================================================
 */
```

mod_perl 1.99_16 License

```
Copyright (c) 1999-2004, The Apache Software Foundation
Licensed under the Apache License, Version 2.0 (the "License"); you may not use
this file except in compliance with the License.  You may obtain a copy of the
License at http://www.apache.org/licenses/LICENSE-2.0
Unless required by applicable law or agreed to in writing, software distributed
under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR
```

CONDITIONS OF ANY KIND, either express or implied. See the License for the
specific language governing permissions and limitations under the License.
Copyright (c) 1999-2004, The Apache Software Foundation

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

"Licensor" shall mean the copyright owner or entity authorized by the copyright owner that is granting the License.

"Legal Entity" shall mean the union of the acting entity and all other entities that control, are controlled by, or are under common control with that entity. For the purposes of this definition, "control" means (i) the power, direct or indirect, to cause the direction or management of such entity, whether by contract or otherwise, or (ii) ownership of fifty percent (50%) or more of the outstanding shares, or (iii) beneficial ownership of such entity.

"You" (or "Your") shall mean an individual or Legal Entity exercising permissions granted by this License.

"Source" form shall mean the preferred form for making modifications, including but not limited to software source code, documentation source, and configuration files.

"Object" form shall mean any form resulting from mechanical transformation or translation of a Source form, including but not limited to compiled object code, generated documentation, and conversions to other media types.

"Work" shall mean the work of authorship, whether in Source or Object form, made available under the License, as indicated by a copyright notice that is included in or attached to the work (an example is provided in the Appendix below).

"Derivative Works" shall mean any work, whether in Source or Object form, that is based on (or derived from) the Work and for which the editorial revisions, annotations, elaborations, or other modifications represent, as a whole, an original work of authorship. For the purposes of this License, Derivative Works shall not include works that remain separable from, or merely link (or bind by name) to the interfaces of, the Work and Derivative Works thereof.

"Contribution" shall mean any work of authorship, including the original version of the Work and any modifications or additions to that Work or Derivative Works thereof, that is intentionally submitted to Licensor for inclusion in the Work by the copyright owner or by an individual or Legal Entity authorized to submit on behalf of the copyright owner. For the purposes of this definition, "submitted" means any form of electronic, verbal, or written communication sent to the Licensor or its representatives, including but not limited to communication on electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, the

Licensor for the purpose of discussing and improving the Work, but excluding communication that is conspicuously marked or otherwise designated in writing by the copyright owner as "Not a Contribution."

"Contributor" shall mean Licensor and any individual or Legal Entity on behalf of whom a Contribution has been received by Licensor and subsequently incorporated within the Work.

2. Grant of Copyright License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable copyright license to reproduce, prepare Derivative Works of, publicly display, publicly perform, sublicense, and distribute the Work and such Derivative Works in Source or Object form.

3. Grant of Patent License. Subject to the terms and conditions of this License, each Contributor hereby grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free, irrevocable (except as stated in this section) patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work, where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) alone or by combination of their Contribution(s) with the Work to which such Contribution(s) was submitted. If You institute patent litigation against any entity (including a cross-claim or counterclaim in a lawsuit) alleging that the Work or a Contribution incorporated within the Work constitutes direct or contributory patent infringement, then any patent licenses granted to You under this License for that Work shall terminate as of the date such litigation is filed.

4. Redistribution. You may reproduce and distribute copies of the Work or Derivative Works thereof in any medium, with or without modifications, and in Source or Object form, provided that You meet the following conditions:

   (a) You must give any other recipients of the Work or Derivative Works a copy of this License; and

   (b) You must cause any modified files to carry prominent notices stating that You changed the files; and

   (c) You must retain, in the Source form of any Derivative Works that You distribute, all copyright, patent, trademark, and attribution notices from the Source form of the Work, excluding those notices that do not pertain to any part of the Derivative Works; and

   (d) If the Work includes a "NOTICE" text file as part of its distribution, then any Derivative Works that You distribute must include a readable copy of the attribution notices contained within such NOTICE file, excluding those notices that do not pertain to any part of the Derivative Works, in at least one of the following places: within a NOTICE text file distributed as part of the Derivative Works; within the Source form or documentation, if provided along with the Derivative Works; or, within a display generated by the Derivative Works, if and wherever such third-party notices normally appear. The contents of the NOTICE file are for informational purposes only and do not modify the License. You may add Your own attribution

notices within Derivative Works that You distribute, alongside or as an addendum to the NOTICE text from the Work, provided that such additional attribution notices cannot be construed as modifying the License.

You may add Your own copyright statement to Your modifications and may provide additional or different license terms and conditions for use, reproduction, or distribution of Your modifications, or for any such Derivative Works as a whole, provided Your use, reproduction, and distribution of the Work otherwise complies with the conditions stated in this License.

5. Submission of Contributions. Unless You explicitly state otherwise, any Contribution intentionally submitted for inclusion in the Work by You to the Licensor shall be under the terms and conditions of this License, without any additional terms or conditions. Notwithstanding the above, nothing herein shall supersede or modify the terms of any separate license agreement you may have executed with Licensor regarding such Contributions.

6. Trademarks. This License does not grant permission to use the trade names, trademarks, service marks, or product names of the Licensor, except as required for reasonable and customary use in describing the origin of the Work and reproducing the content of the NOTICE file.

7. Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Licensor provides the Work (and each Contributor provides its Contributions) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the Work and assume any risks associated with Your exercise of permissions under this License.

8. Limitation of Liability. In no event and under no legal theory, whether in tort (including negligence), contract, or otherwise, unless required by applicable law (such as deliberate and grossly negligent acts) or agreed to in writing, shall any Contributor be liable to You for damages, including any direct, indirect, special, incidental, or consequential damages of any character arising as a result of this License or out of the use or inability to use the Work (including but not limited to damages for loss of goodwill, work stoppage, computer failure or malfunction, or any and all other commercial damages or losses), even if such Contributor has been advised of the possibility of such damages.

9. Accepting Warranty or Additional Liability. While redistributing the Work or Derivative Works thereof, You may choose to offer, and charge a fee for, acceptance of support, warranty, indemnity, or other liability obligations and/or rights consistent with this License. However, in accepting such obligations, You may act only on Your own behalf and on Your sole responsibility, not on behalf of any other Contributor, and only if You agree to indemnify, defend, and hold each Contributor harmless for any liability incurred by, or claims asserted against, such Contributor by reason of your accepting any such warranty or additional liability.

Perl Artistic License

The "Artistic License"

Preamble

The intent of this document is to state the conditions under which a Package may be copied, such that the Copyright Holder maintains some semblance of artistic control over the development of the package, while giving the users of the package the right to use and distribute the Package in a more-or-less customary fashion, plus the right to make reasonable modifications.

Definitions

"Package" refers to the collection of files distributed by the Copyright Holder, and derivatives of that collection of files created through textual modification.

"Standard Version" refers to such a Package if it has not been modified, or has been modified in accordance with the wishes of the Copyright Holder as specified below.

"Copyright Holder" is whoever is named in the copyright or copyrights for the package.

"You" is you, if you're thinking about copying or distributing this Package.

"Reasonable copying fee" is whatever you can justify on the basis of media cost, duplication charges, time of people involved, and so on. (You will not be required to justify it to the Copyright Holder, but only to the computing community at large as a market that must bear the fee.)

"Freely Available" means that no fee is charged for the item itself, though there may be fees involved in handling the item. It also means that recipients of the item may redistribute it under the same conditions they received it.

1. You may make and give away verbatim copies of the source form of the Standard Version of this Package without restriction, provided that you duplicate all of the original copyright notices and associated disclaimers.
2. You may apply bug fixes, portability fixes and other modifications derived from the Public Domain or from the Copyright Holder. A Package modified in such a way shall still be considered the Standard Version.
3. You may otherwise modify your copy of this Package in any way, provided that you insert a prominent notice in each changed file stating how and when you changed that file, and provided that you do at least ONE of the following:
    a. place your modifications in the Public Domain or otherwise make them Freely Available, such as by posting said modifications to Usenet or an equivalent medium, or placing the modifications on a major archive site such as uunet.uu.net, or by allowing the Copyright Holder to include your modifications in the Standard Version of the Package.
    b. use the modified Package only within your corporation or organization.
    c. rename any non-standard executables so the names do not conflict with standard executables, which must also be provided, and provide a separate manual page for each non-standard executable that clearly documents how it differs from the Standard Version.
    d. make other distribution arrangements with the Copyright Holder.
4. You may distribute the programs of this Package in object code or executable form, provided that you do at least ONE of the following:

a. distribute a Standard Version of the executables and library files, together with instructions (in the manual page or equivalent) on where to get the Standard Version.

b. accompany the distribution with the machine-readable source of the Package with your modifications.

c. give non-standard executables non-standard names, and clearly document the differences in manual pages (or equivalent), together with instructions on where to get the Standard Version.

d. make other distribution arrangements with the Copyright Holder.

5. You may charge a reasonable copying fee for any distribution of this Package. You may charge any fee you choose for support of this Package. You may not charge a fee for this Package itself. However, you may distribute this Package in aggregate with other (possibly commercial) programs as part of a larger (possibly commercial) software distribution provided that you do not advertise this Package as a product of your own. You may embed this Package's interpreter within an executable of yours (by linking); this shall be construed as a mere form of aggregation, provided that the complete Standard Version of the interpreter is so embedded.

6. The scripts and library files supplied as input to or produced as output from the programs of this Package do not automatically fall under the copyright of this Package, but belong to whoever generated them, and may be sold commercially, and may be aggregated with this Package. If such scripts or library files are aggregated with this Package through the so-called "undump" or "unexec" methods of producing a binary executable image, then distribution of such an image shall neither be construed as a distribution of this Package nor shall it fall under the restrictions of Paragraphs 3 and 4, provided that you do not represent such an executable image as a Standard Version of this Package.

7. C subroutines (or comparably compiled subroutines in other languages) supplied by you and linked into this Package in order to emulate subroutines and variables of the language defined by this Package shall not be considered part of this Package, but are the equivalent of input as in Paragraph 6, provided these subroutines do not change the language in any way that would cause it to fail the regression tests for the language.

8. Aggregation of this Package with a commercial distribution is always permitted provided that the use of this Package is embedded; that is, when no overt attempt is made to make this Package's interfaces visible to the end user of the commercial distribution. Such use shall not be construed as a distribution of this Package.

9. The name of the Copyright Holder may not be used to endorse or promote products derived from this software without specific prior written permission.

10. THIS PACKAGE IS PROVIDED "AS IS" AND WITHOUT ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTIBILITY AND FITNESS FOR A PARTICULAR PURPOSE.

The End

Index

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X