Oracle® Containers for J2EE

Security Guide

10*g* (10.1.3.5.0)

E13977-01

July 2009

ORACLE®

Oracle Containers for J2EE Security Guide, 10*g* (10.1.3.5.0)

E13977-01

Primary Author: Joseph Ruzzi

Contributing Author: Brian Wright, Elizabeth Hanes Perry

Contributor: Ganesh Kirti, Raymond Ng, Rachel Chan, Nithya Muralidharan, Kumar Valendhar, Moushmi Banerjee, Dheeraj Goswami, Sam Zhou, Srikant Tirumalai, Sirish Vepa, Vineet Garg, Bill Bathurst, Debu Panda, Steve Button, Tom Snyder, Jeff Trent, Bob Nettleton, Vinay Shukla, Alex Kosowski, Rajbir Chahal, Michael Hwa, Jayanthi Kulkarni, Kavita Tippana, Helen Zhao, Sandeep Bangera, Cania Lee Chung, Deepika Damojipurapu, Lakshmi Thiyagarajan, Soumya Aithal, Serouj Ourishian, Phil Varner, Chaya Ramanujam, Xiaopeng Wu, Jyotsna Laxminarayanan, Lelia Yin, Raghav Srinivisan, Sam Chou, Bhupindra Singh, Ashish Kolli, Frank Nimphius, Dan Hynes, Steve Button, Viresh Garg, Alfred Franci, Peter LaQuerre

Contents

3 Overview of OC4J Security

4 Overview of Security Administration

5 Authorization in OC4J

6 General Tasks for OC4J Security

7 File-Based Security Provider

8 Oracle Identity Management

9 Login Modules

10 External LDAP Security Providers

11 Oracle Access Manager

12 User and Role API Framework

13 Pluggable Identity Management Framework

14 OC4J Java Single Sign-On

15 SSL Communication with OC4J

16 Oracle Security for Client Connections

17 Web Application Security Configuration

18 EJB Security Configuration

21 Configuring Windows Native Authentication

A Tips and Troubleshooting for OC4J Security

B OracleAS JAAS Provider Samples

C OracleAS JAAS Provider Admintool Reference

D OracleAS JAAS Provider Configuration Files

E Third Party Licenses

Index

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 to all users, including users that are disabled. 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.

Deaf/Hard of Hearing Access to Oracle Support Services

To reach Oracle Support Services, use a telecommunications relay service (TRS) to call Oracle Support at 1.800.223.1711. An Oracle Support Services engineer will handle technical issues and provide customer support according to the Oracle service request process. Information about TRS is available at `http://www.fcc.gov/cgb/consumerfacts/trs.html`, and a list of phone numbers is available at `http://www.fcc.gov/cgb/dro/trsphonebk.html`.

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 Services Guide*

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

- *Oracle Containers for J2EE Resource Adapter Administrator's Guide*

  This provides information about resource adapters and the J2EE Connector Architecture.

- *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 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.

Related Javadoc sets:

- *Oracle Containers for J2EE Security Java API Reference*

  Documents APIs of the OracleAS JAAS Provider, identity management framework, and Java SSO.

- *Oracle Containers for J2EE User and Role Java API Reference*

  Documents APIs for accessing user and role information from identity management repositories.

- *Oracle Application Server HTTPClient Java API Reference*

  Documents APIs of the Oracle `HTTPClient` packages.

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*
- *Oracle Application Server Best Practices*

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

- *Oracle Identity Management Infrastructure 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 Access Manager:

- *Oracle Access Manager Introduction*
- *Oracle Access Manager Installation Guide*
- *Oracle Access Manager System Administration Guide*
- *Oracle Access Manager Identity and Common Administration Guide*
- *Oracle Access Manager Developer Guide*
- *Oracle Access Manager Deployment Guide*

For additional information, see:

- Top-level link for Oracle documentation from the Oracle Technology Network:

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

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

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

- 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.
italic	Italic type indicates book titles, emphasis, terms defined in text, 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 since the 10.1.2.0.2 release:

Changes in Release 10.1.3.5

The following security features and enhancements were added for the OC4J 10.1.3.5 implementation:

Noteworthy Changes and Additions

The *Oracle Containers for J2EE Security Guide* now contains instructions for configuring a data source to use TCPS when connecting to the database. See "Configuring a TCPS Data Source" on page 15-24 for detailed information.

Changes in Release 10.1.3.4

The following security features and enhancements were added for the OC4J 10.1.3.4 implementation:

Noteworthy Changes and Additions

Note the following key additions in the OC4J 10.1.3.4 implementation:

- OC4J and Java SSO now supports Windows Native Authentication and Kerberose. Users of OC4J Web applications can be automatically authenticated based on their Windows desktop login. See Chapter 21, "Configuring Windows Native Authentication" for detailed information.
- Additional documentation has been added that describes how to use the HTTP Client API to include basic, digest, and NTLM authentication information in a request header. See "Using Non-SSL Client Authentication" on page 16-2 for detailed information.
- Additional documentation has been added that describes how to set a Subject into the OC4J container. See "Setting a Subject Into the OC4J Container" on page 5-11.
- Document Errata items since the 10.1.3.1 release have been fixed in this release of the *Oracle Containers for J2EE Security Guide*.

Updated Deprecation Notices

Note the following deprecations in the OC4J 10.1.3.4 implementation:

- Oracle Java SSL is deprecated and no longer used by `HTTPClient`. The JDK-default JSSE implementation is now the default SSL implementation for `HTTPClient`.

Changes in Release 10.1.3.1

This section notes changes and updated deprecation notices for the OC4J 10.1.3.1 implementation. Also review the section that follows, "Changes in Release 10.1.3.0.0".

Noteworthy Changes and Additions

Note the following key additions in the OC4J 10.1.3.1 implementation:

- Identity management framework to support heterogeneous third-party identity management systems
- Java SSO, an alternative single sign-on solution packaged with OC4J that does not require additional infrastructure (such as Oracle Single Sign-On and Oracle Access Manager single sign-on do) and decouples OC4J from any identity management system that you use
- `DBTableOraDataSourceLoginModule` to replace `DataSourceUserManager` functionality
- A new user and role API framework, particularly for use with supported LDAP servers. This includes replacement functionality for the deprecated `UserManager`, `User`, and `Group` classes

In addition, note the following changes:

- The JDK-default JSSE implementation is now the default SSL implementation for `HTTPClient`. (This is a step toward deprecating OracleSSL, the previous default SSL implementation, for use with `HTTPClient`.)

Updated Deprecation Notices

Note the following deprecations in the OC4J 10.1.3.1 implementation:

- The `com.evermind.security` package and its classes are deprecated. **They will no longer be supported in the 11*g* release.**
  - `UserManager` class: Use JAAS custom login modules instead of custom `com.evermind.security.UserManager` implementations.
  - `User` class: Use standard JAAS APIs instead.
  - `Group` class: Use standard JAAS APIs instead.
- The `XMLUserManager` class and its data store, `principals.xml`, are deprecated. **They will no longer be supported in the 11*g* release.** For instructions on migration, see "Migrating Principals from the principals.xml File" on page 7-16.

Changes in Release 10.1.3.0.0

The following security features and enhancements were added for the OC4J 10.1.3.0.0 implementation:

- Support for the COREid Access (now Oracle Access Manager) 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 OC4J 10.1.3.x implementations.
- 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 7-16.
- 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 `com.evermind.security.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 Accounts" on page 4-11.

- Required OC4J accounts are created automatically in Oracle Internet Directory when you associate an OC4J instance with an OID instance. See "Required Accounts Created in Oracle Internet Directory" on page 8-7.
- Setting `LD_LIBRARY_PATH` is no longer necessary in 10.1.3.x implementations.
- The `jazn.debug.log.enable` flag is no longer supported for logging. Use regular OC4J logging features. See "Logging" on page A-5.

1

Basic Security Concepts

This chapter provides an overview of security concepts, focusing on the following areas:

- Application-Level Security
- Transport-Level Security

These are two basic categories of security that can be independently configured but are often interrelated. The former mostly determines who can access data and what tasks they are allowed to perform; the latter mostly determines the security of data as it is transmitted.

Note that application-level configuration can include transport-level specifications, such as having an application-level constraint requiring Secure Sockets Layer (which is a transport-level feature, discussed later). And transport-level security can also involve authentication (limiting data access to appropriate users), such as when client certification is requested as part of the transport-level functionality.

Application-Level Security

Application-level security determines who can access an application or its data, and what tasks they can perform. The following topics discuss key areas of functionality:

- About Authentication
- About Authorization

About Authentication

Authentication deals with the question "Who is trying to access 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 is an example of a user repository.

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

About Authorization

Authorization regards the question "Who can perform tasks on which 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.

The following sections discuss types of authorization, or *access control*, and related topics:

- Access Control Lists and the Capability Model of Access Control
- Role-Based Access Control

The capability and role-based models are complementary and often used together.

Access Control Lists and the 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 permissions. With this model, access control information is associated with a resource, and authorization is associated with an entity (referred to as a "principal", defined in "Principals and Subjects" on page 2-12), such as a user named Amy.

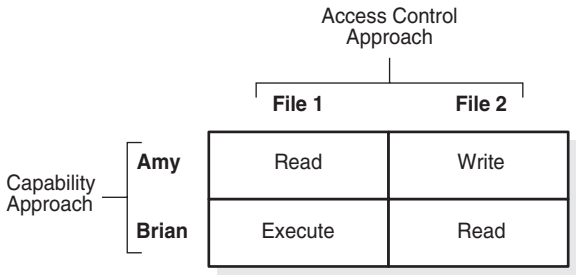
An *access control list* (ACL) is associated with a protected target resource, such as a directory or file, and contains information about which access rights each user has for the particular resource. Each file in a file system may have an ACL, for example. And the capabilities (privileges) specified in an ACL are associated with particular users, specifying which users have what privileges for the file with which the ACL is associated.

When a user Amy logs in and is successfully authenticated, her permissions are retrieved and granted so that she is free to execute the actions permitted by these permissions—for example, to read from a `File1` and write to a `File2`.

The capability model and access control look at the same information from different perspectives. While a capability is associated with a user attempting to access a resource, an access control list is associated with the resource that the user is trying to access.

Figure 1–1 shows the user Amy having permission to read `File1` and write to `File2`, while a user Brian has permission to execute `File1` and read `File2`. An access control list would come from the perspective of `File1` and `File2`, specifying for each file what users have access and what their specific permissions are. The capability model comes from the perspective of Amy and Brian, specifying for each user what files they can access and what they have permission to do with respect to each file.

Figure 1–1 Access Control Approach Versus Capability Approach



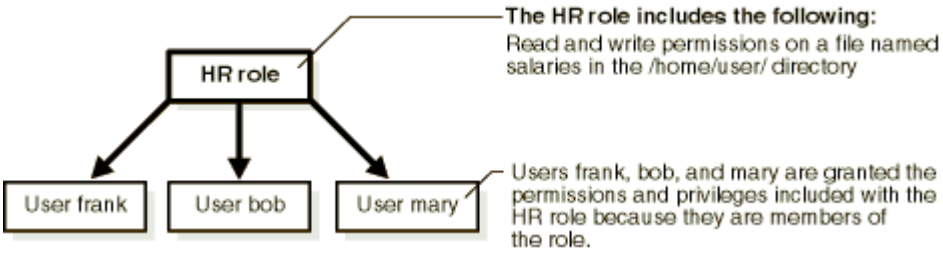
Role-Based Access Control

A *role* is essentially a job function or title that defines an authority level. A role can have multiple users and multiple permissions. 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 1–2 provides an example of role-based access control.

Figure 1–2 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 3-8

Transport-Level Security

Independent of the previously discussed features for authentication and authorization are features for making data secure as it is transmitted. This section provides an overview of features intended to ensure that data transmitted over a network or the Internet cannot be intercepted and read or altered by a third party. OC4J supports secure communications using the HTTP protocol over the Secure Sockets Layer.

The following related topics are discussed here:

- Secure Sockets Layer and HTTPS
- SSL Authentication
- X.509 Certificates

- Key Encryption and Exchange

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.

Note that whether the server configures SSL communication is independent of whether the client configures SSL communication. In this document, Chapter 15, "SSL Communication with OC4J", covers information about enabling SSL at the OC4J end, and about SSL communication between Oracle HTTP Server and OC4J in an Oracle Application Server environment. Chapter 16, "Oracle Security for Client Connections", covers information about the OC4J implementation of HTTPS that provides SSL functionality to client HTTP connections.

SSL Authentication

With SSL communication, any of the following authentication scenarios are possible:

- No SSL authentication (or null authentication): The server does not send a certificate and does not request a certificate from the client. From an SSL perspective, the server does not know who the remote client is, or accepts any certificate that may be presented by the client.
- One-way SSL authentication: Either the server or the client, but not both, requires certificates. "Server authentication", for example, is one-way authentication where the server sends its certificate to the client but does not request a certificate from the client. Alternatively, the server may require a certificate, but does not send one and the client does not require one.
- Two-way SSL authentication: This is "client and server authentication", where the server sends a certificate, required by the client, and also requires the client to send a certificate.

Configuring SSL authentication in the server is independent of configuring SSL authentication in the client.

X.509 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 *trust point*. The trust point signing the certificate can be a *certificate authority* (CA) 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 typically uses a combination of asymmetric (public/private key) and symmetric key encryption to secure communications. The exchange of public keys is used for mutual authentication of the parties involved in the communication. This also allows the parties to securely cooperate in the creation of symmetric keys that will be used in further encryption and decryption of data in the session. The following is a basic example of the creation of an SSL session between a client and a server:

1. The client sends cipher suites, compression methods, highest protocol versions, and random bytes to the server. The server chooses the connection parameters from the choices offered by the client.
2. The public keys (X.509 certificates) are exchanged.
    a. The server sends the client its public key, and the client sends the server its public key.
    b. The keys are used for mutual authentication where each verifies the certificate of the other.
3. The symmetric keys are exchanged. Communications are secured in this step using the exchanged public keys.
    a. A master secret is generated cooperatively between the server and client.
    b. Session keys (bulk encryption keys) are then generated based on the master secret, such as a 128-bit RC4 key.
    c. The client and server each sends a message that it will use the session key for further communication.
4. SSL traffic uses symmetric keys for encryption and decryption.

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, 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.

2

Java Platform Security

This chapter provides an overview of standard security models that can be used with Java and J2EE applications, covering the following topics:

- J2EE Security Model
- Java 2 Security Model
- Java Authentication and Authorization Service
- Security Considerations during Development

> **Note:** The J2EE, Java 2, and JAAS security models are somewhat independent of each other and can be used separately or in combinations. Strategies are discussed in "Authorization Strategies" on page 5-20.

See Also:

- *Oracle Application Server Best Practices* (available post-release) for information about best practices for security

J2EE Security Model

J2EE defines a declarative authorization model for container-managed security that decouples applications from the underlying security infrastructure. Authorization policy (an association between resources and users or roles) 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.

This section discusses the following J2EE security features:

- Web Application Authentication and Authorization
- Enterprise JavaBeans Authentication and Authorization
- Identity Propagation

Web Application Authentication and Authorization

This section discusses topics for Web application security, primarily involving declarative security configuration. There is also discussion of APIs for more advanced programmatic features, where security functionality can be determined at runtime.

The following topics are covered:

- Web Application Standard Authentication Methods
- Web Application URL-Based Authorization
- Run-As Mode and Propagated Identities in Web Applications
- Related Web Application APIs

Web Application Standard Authentication Methods

Several standard methods of authentication can be used to access a J2EE Web application:

- **Basic**

  With basic authentication, the user is prompted directly for a user name and password, without going through a single sign-on implementation.

- **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. (In OC4J, 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.

- **Client-cert**

  This method, used in conjunction with the Secure Sockets Layer (SSL), authenticates the client through HTTPS. The user must possess a public key certificate.

Notes:

- OC4J also supports several Oracle-specific single sign-on authentication methods, as summarized in "Overview of Oracle Application Server Single Sign-On Alternatives" on page 3-6.
- For either the file-based provider or Oracle Identity Management, if you are not using single sign-on, we recommend digest authentication as a more secure solution than basic authentication.

See Also:

- "Specifying the Authentication Method (auth-method)" on page 17-1

Web Application URL-Based Authorization

In the J2EE security model, Web resources to be secured are identified by their URL patterns. This is specified in the `web.xml` file of the Web application. For example, the following excerpt is from the configuration to protect resources under the URL pattern "`/resource`" of an application.

```
<web-resource-collection>
  <web-resource-name>resource access</web-resource-name>
  <url-pattern>/resource</url-pattern>
```

```
  </web-resource-collection>
```

This is part of a security constraint in `web.xml` that also specifies the J2EE logical role that is allowed to access the resource. J2EE logical roles, discussed in the J2EE specification, include developers (application component providers), assemblers, deployers, and system administrators.

For example, assume the J2EE role `sr_developers` is declared in the `web.xml` file. The security constraint to allow this role to access the resource would look like this:

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>resource access</web-resource-name>
     <url-pattern>/resource</url-pattern>
   </web-resource-collection>
   <!-- authorization -->
   <auth-constraint>
     <role-name>sr_developers</role-name>
   </auth-constraint>
</security-constraint>
```

The `sr_developers` role can then be mapped to the appropriate deployment role (a role defined in the security provider) in a later OC4J-specific configuration step.

See Also:

- "Overview of Security Role Mapping" on page 3-8
- For details, "Web Application Security Role and Constraint Configuration" on page 17-6

Run-As Mode and Propagated Identities in Web Applications

For calls from a Web application to an EJB, the default mode is for the Web client's security identity to be propagated to the EJB container.

There are also situations where a Web container must allow Web clients that are unknown to the Web container or EJB container to make calls. This includes the scenario of supporting Web clients that have not authenticated themselves to the container, such as to allow access of resources from the Internet.

For situations such as these, the `web.xml` file of a Web application may specify a "run-as" identity through the `<run-as>` subelement of the `<servlet>` element:

```
<run-as>
   <role-name>sr_developers</role-name>
</run-as>
```

The Web container propagates the security identity for any call from a servlet to the EJB layer, in terms of the role specified in the `<run-as>` element, which must be a role previously declared through a `<security-role>` element. The run-as identity hides the propagated identity.

See Also:

- "Identity Propagation" on page 2-7
- "Specifying a Run-As Security Identity for a Web Application" on page 17-8

Related Web Application APIs

For more advanced uses, there are standard J2EE programmatic features that allow Web applications to retrieve information about a user. You can use these methods in determining if a user should be allowed to access a resource.

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

- `Principal getUserPrincipal()`

  Returns a principal object containing the name of the authenticated user making the request (or null if the user has not been authenticated).

  Where identity propagation is used between a servlet and an EJB, the principal name returned by the `getUserPrincipal()` method of the calling servlet would be the same as that returned by the `getCallerPrincipal()` method of the EJB.

- `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.

Notes:

- APIs documented here can be used with either the file-based provider or Oracle Internet Directory as the user repository.
- The `com.evermind.security.User` and `Group` classes from previous releases are deprecated. (They will not be supported in the 11*g* release.) Use standard JAAS APIs and signatures instead, such as for `getUserPrincipal()`, utilizing populated subjects as appropriate.

See Also:

- "Principals and Subjects" on page 2-12 for a general discussion of principals
- *Oracle Containers for J2EE Security Java API Reference* (Javadoc)

Enterprise JavaBeans Authentication and Authorization

This section discusses topics for EJB security, primarily involving declarative security configuration. There is also discussion of APIs for more advanced programmatic features, where security functionality can be determined at runtime.

The following topics are covered:

- EJB Authentication
- EJB Method-Based Authorization
- Run-As Mode and Propagated Identities in EJB Applications
- Related EJB APIs

EJB Authentication

An EJB being accessed in a remote container requires authentication of the client that is accessing it:

- A standalone Java client can pass credentials through user and password settings in the `jndi.properties` file.
- An EJB or Web client can pass credentials through a `javax.naming.InitialContext` instance, which is created to look up the remote EJB.

In addition, where ORMIS is used (ORMI in conjunction with the Secure Sockets Layer), you can use client-cert authentication with EJBs.

See Also:

- "Specifying Credentials in EJB Clients" on page 18-9
- Chapter 15, "SSL Communication with OC4J" (particularly "Requesting Client Authentication" on page 15-15 and "Enabling ORMIS for OC4J" on page 15-18)

EJB Method-Based Authorization

In the J2EE security model, EJB resources to be secured are identified by their method names or name masks within the particular EJB. This is specified in the `ejb-jar.xml` file of the EJB. For example, the following excerpt is from the configuration to protect all methods of a `PurchaseOrder` bean:

```
<method>
 <ejb-name>PurchaseOrder</ejb-name>
 <method-name>*</method-name>
</method>
```

This is part of a method permission in `ejb-jar.xml` that also specifies the J2EE logical role that is allowed to access the resource. J2EE logical roles, discussed in the J2EE specification, include developers (application component providers), assemblers, deployers, and system administrators.

The following excerpt allows the role `myMgr` to access any method in the `PurchaseOrder` EJB:

```
<method-permission>
 <role-name>myMgr</role-name>
 <method>
  <ejb-name>PurchaseOrder</ejb-name>
  <method-name>*</method-name>
 </method>
</method-permission>
```

The `myMgr` role can then be mapped to the appropriate deployment role (a role defined in the security provider) in a later OC4J-specific configuration step.

See Also:

- "Overview of Security Role Mapping" on page 3-8
- For details, "Specifying J2EE Roles and Method Permissions in the EJB Deployment Descriptor" on page 18-3

Run-As Mode and Propagated Identities in EJB Applications

The `ejb-jar.xml` deployment descriptor can include specification of a "run-as" identify for an EJB, which can be used in conjunction with identity propagation when one EJB calls a second EJB. The run-as identity hides the propagated identity, and applies to the called EJB as a whole and is used as the identity in executing its methods. The identity would be a J2EE logical role previously declared through a `<security-role>` element.

Use the `<run-as>` subelement of the `<security-identity>` element for this purpose:

```
<run-as>
   <role-name>admin</role-name>
</run-as>
```

See Also:

- "Identity Propagation" on page 2-7
- "Specifying a Run-As or Caller Security Identity for an EJB" on page 18-6

Related EJB APIs

For more advanced uses, there are standard J2EE programmatic features that allow EJBs to retrieve information about a caller. You can use these methods in determining if a caller should be allowed to access a resource.

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

- `Principal getCallerPrincipal()`

  Returns a principal object that identifies the caller.

  Where identity propagation is used between a servlet and an EJB, the principal name returned by the `getUserPrincipal()` method of the calling servlet would be the same as that returned by the `getCallerPrincipal()` method of the EJB.

  Where identity propagation is used between EJBs, the client identity is propagated to all EJBs downstream in the call chain, and the principal name returned by `getCallerPrincipal()` would be the same for all EJBs in the call chain. This assumes, however, that `subject.propagation` permission (discussed in "Grant RMI Permission for Subject Propagation" on page 18-14) is granted to all authenticating users in the sequence.

- `boolean isCallerInRole(String rolename)`

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

Notes:

- APIs documented here can be used with either the file-based provider or Oracle Internet Directory as the user repository.
- The `com.evermind.security.User` class from previous releases is deprecated. (It will not be supported in the 11*g* release.) Use standard JAAS APIs instead, utilizing populated subjects as appropriate.

See Also:

- "Principals and Subjects" on page 2-12 for a general discussion of principals

Identity Propagation

Identity propagation in J2EE refers to the forwarding of a security identity from a Web module or EJB to an EJB that the original Web module or EJB invokes:

1. An initiating application client or Web client uses a security identity to access an intermediate EJB or Web module.
2. The intermediate EJB or Web module invokes a target EJB by forwarding, or propagating, a security identity to access the target EJB.

There are typically two models for propagation:

- If the target container trusts the intermediate container, the caller identity of the intermediate EJB or Web module can be propagated to the target EJB.
- If the target container expects access by a particular identity, that identity can be propagated to the target EJB.

In general, the target container must trust the intermediate container, because no data is made available for the target container to use in authenticating the propagated identity. Because the propagated identity will presumably be used for authorization checks, such as `isCallerInRole()`, it is critical for the propagated identity to be authentic.

In OC4J, identity propagation is referred to as *subject propagation*.

See Also:

- "Enabling and Configuring Subject Propagation for ORMI" on page 18-12
- The following tutorial for additional information:

  `http://java.sun.com/j2ee/1.4/docs/tutorial/doc/Security11.html`

Java 2 Security Model

The Java 2 Security Model is supported as part of Oracle Application Server security. Note, however, that it is not implemented in OC4J itself, but rather in the underlying JDK.

This security model provides developers and administrators with increased control over many aspects of enterprise 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 (discussed in "Java 2 Authorization: Java 2 Security Policies" on page 2-10) for these domains.

The Java 2 Security Model by itself, however, has certain limitations. It is based on application code, 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 following sections discuss characteristics and features of the Java 2 Security Model:

- Code-Based Security

- Security Permissions
- Protection Domains
- Java 2 Authorization: Java 2 Security Policies
- Java 2 Authorization: Security Managers and Access Controllers

See Also:

- For a tutorial on Java 2 Security:

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

- For full information on Java 2 Security:

  `http://java.sun.com/javase/technologies/security.jsp`

Code-Based Security

Code-based security restricts the operations that applications can perform, by applying sets of permissions to executing code. This protects against the actions of untrusted or possibly malicious code, and could be to accomplish any of the following, for example:

- Restrict database access to only trusted applications
- Disallow the execution of code downloaded from the Internet
- Limit operations that can be performed by third-party code

Code-based security differs from role-based security in that it is not based on users or user roles. Permissions are granted based on code characteristics, such as where the code is coming from and whether it is digitally signed (and by whom).

A *codebase* is a URL indicating code location, such as the following examples:

- `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`

A *codesource* expands this concept to optionally include an array of certificates (stored in a Java keystore) to verify signed code originating from the location. A codesource is represented by a `java.security.CodeSource` instance, which is constructed by specifying a `java.net.URL` instance and an array of `java.security.cert.Certificate` instances.

The standard `CodeSource` class includes the following methods:

- `Certificate[] getCertificates()`

  Returns an array of the certificates associated with this codesource.

- `URL getLocation()`

  Returns the URL location associated with this codesource.

- `boolean equals(Object)`

  Compares the specified object (presumably a codesource object) with this codesource object for equality. Two codesources are considered equal if their locations are identical and they have the same set of certificates (though it is not necessary for the certificates to be in the same order).

- `boolean implies(CodeSource)`

Performs a series of comparisons to see if this codesource "implies" the specified codesource. For example, codesources with the following locations, and null certificates, all imply the codesource with location `http://java.sun.com/classes/foo.jar` and null certificates:

```
http:
http://*.sun.com/classes/*
http://java.sun.com/classes/-
http://java.sun.com/classes/foo.jar
```

See Also:

- For information about the `CodeSource` class (and other standard classes):

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

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 whether 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 2–1 identifies characteristics of a Java permission instance.

Important: `AllPermission` should be used with caution, and only when necessary.

Table 2–1 Java Permission Instance Characteristics

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/*`

Protection Domains

A protection domain associates permissions with codesources (defined in "Code-Based Security" on page 2-8). 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.

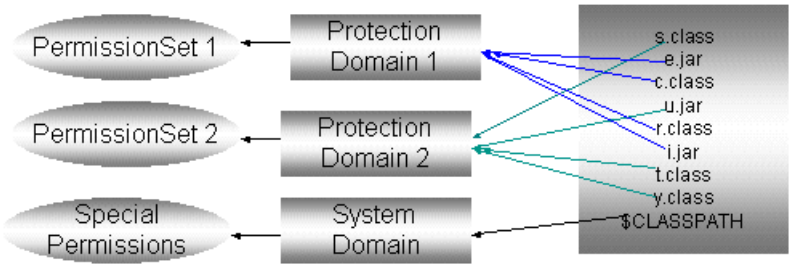
Each Java class is associated with a protection domain when it is loaded. Specifically, each class being loaded is associated with a `java.security.ProtectionDomain` instance. The permissions granted to this protection domain may be statically bound or dynamically determined when an access control check 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 one or more codesources. It may also contain a `Principal` array describing who is executing the code, a classloader reference, and a permission set (`java.security.PermissionCollection` instance) representing a collection of `Permission` objects.

Figure 2–1 shows an example of the relationship between code, protection domains, and permission sets. In this example:

- Protection Domain 1 associates the codesources `e.jar`, `c.class`, `r.class`, and `i.jar` with Permission Set 1.
- Protection Domain 2 associates the codesources `s.class`, `u.jar`, `t.class`, and `y.class` with Permission Set 2.
- The System Domain associates all JAR files in the classpath with Special Permissions.

Figure 2–1 Protection Domains



See Also:

- "Principals and Subjects" on page 2-12 for a general discussion of principals

Java 2 Authorization: Java 2 Security Policies

In Java 2, a *policy* is a mapping between running code and resource access permissions granted to the code. Elements of a policy include codesources, permissions, and protection domains, all described in preceding sections.

In the J2SE implementation, a policy is represented by a `java.security.Policy` instance. Note that this class, and the Java 2 Security Model in general, is implemented in the underlying JDK, not in OC4J itself.

Java 2 policies, for code-based permissions, are declared in `.policy` files, such as `java.policy` or `java2.policy` (typical examples). A policy contains a collection of permission grants to codebases, and may contain a reference to a keystore (described in "Key Encryption and Exchange" on page 1-5). The following are typical locations for Java 2 policy files:

- *JAVA_HOME*`/lib/security/java.policy`
- *USER_HOME*`/java.policy`

- *ORACLE_HOME*/j2ee/home/config/java2.policy

  (When you start OC4J with a security manager, the file in this location contains the relevant permissions.)

  See Also:

  - "Specifying a Java 2 Security Manager and Policy File" on page 5-1
  - "Creating or Updating a Java 2 Policy File" on page 5-3

Java 2 Authorization: Security Managers and Access Controllers

A *security manager* (`java.lang.SecurityManager` instance) allows an application to implement Java 2 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. The `SecurityManager` class has a number of `check`*`Xxx`*`()` 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)`. Note, however, that the `AccessController.checkPermission(Permission)` method does not require a security manager.

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` instance.

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 (an action requiring specific privileges) with privileges enabled.

There is also the concept of an *access control context* (a `java.security.AccessControlContext` instance), which you can use to restrict access to resources based on a particular security context. For example, you can construct an access control context from a particular array of protection domain instances. The `AccessControlContext` class also defines a `checkPermission(Permission)` method, which in this case makes access decisions based on the `AccessControlContext` instance from which it is called, rather than on the context of the current execution thread. So the usefulness of an

`AccessControlContext` instance is for a situation where a security check that should be performed with respect to a particular context must be performed from a different context.

Important: In order to use Java 2 policies, you must specifically enable a security manager, as discussed in "Specifying a Java 2 Security Manager and Policy File" on page 5-1. This allows the JDK and the `Subject` methods `doAs()` and `doAsPrivileged()` (discussed later in this chapter) to check permissions of executing code.

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`

Java Authentication and Authorization Service

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 Java 2 security, basing authorization on who is running code (subject-based security) in addition to what code is running (the code-based security of Java 2).

JAAS also implements a Java version of the standard Pluggable Authentication Module (PAM) framework. This enables an application to remain independent from the authentication service, and supports the use of custom authentication modules.

JAAS extends the access control architecture of the Java 2 Security Model to support subject-based authorization. It also supports declarative security settings, in deployment descriptors, instead of being limited to code-based security settings.

OC4J includes a scalable JAAS provider and uses JAAS as a standard mechanism for fine-grained authorization, as opposed to using a proprietary mechanism. OC4J supports JAAS authorization for both Web-based and EJB-based applications.

The following sections discuss JAAS characteristics and features:

- Principals and Subjects
- JAAS Authentication: Login Modules
- JAAS Authorization: Subject Methods doAs() and doAsPrivileged()

See Also:

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

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

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 identity and roles, and other security-related attributes such as passwords, cryptographic keys, or other credentials. A subject is represented by an instance of the `javax.security.auth.Subject` class.

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.

JAAS Authentication: Login Modules

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 a subject. Possible types of JAAS login modules include a principal-mapping module, a credential-mapping module, or a Kerberos module.

See Also:

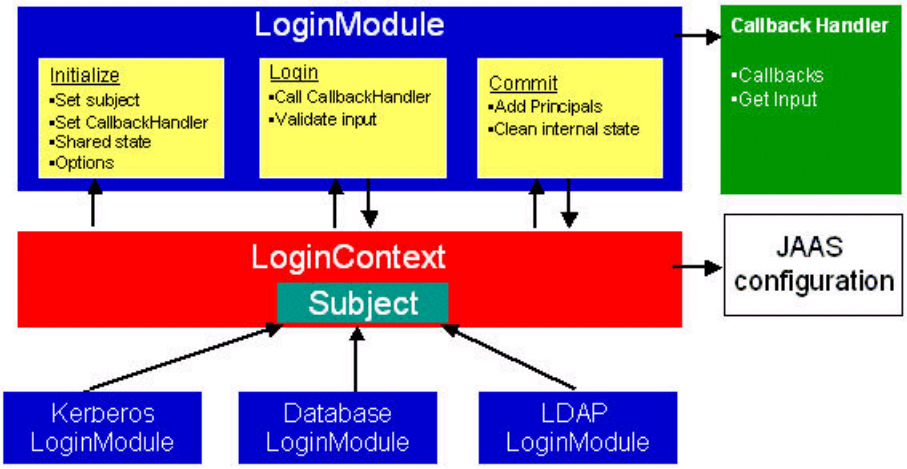
- Chapter 9, "Login Modules"

About Login Modules

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 2–2 summarizes this, and shows functions of the login module.

Figure 2–2 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.

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), `CoreIDLoginModule` (for the Oracle Access Manager), and `DBTableOraDataSourceLoginModule` (to use a database identity store).

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.

Stacking Login Modules

The JAAS PAM architecture allows an enterprise application to customize its authentication mechanism by defining a stack of login modules, each of which is independent and communicates to its own user repository.

The `javax.security.auth.login.Configuration` class is an abstract class to represent the configuration of login modules under an application. A `Configuration` instance specifies which login modules should be used for a

particular application, and in what order the login modules should be invoked. The `Configuration` class is extended to provide an appropriate implementation.

For each login module, a control flag setting in the login module configuration determines whether that login module is "required", "requisite", "sufficient", or "optional", with the meanings of these settings being according to standard functionality of the `Configuration` class, as discussed in Table 9–5, " Login Module Control Flags" on page 9-17.

A login configuration contains an ordered list of login modules specified by fully qualified class names, along with control flag settings and any option settings particular to each login module. Authentication proceeds down the module list in order. (In OC4J, the order is determined by the order in which the login modules are configured in the `system-jazn-data.xml` file.)

Overall authentication is governed by the individual login modules and their control flag settings.

JAAS Authorization: JAAS Security Policies

In JAAS, a *policy* is an association between resources and users or roles. With the integration of JAAS into Java 2 security, the `Policy` API handles queries based on principals, and the default policy implementation supports grant entries based on principals. In extending the Java 2 security model, access control is based not only on the code that is executing, but also on who is executing it. More specifically, a policy is a repository of authorization rules, containing information that answers the question: Given a grantee, what are the granted permissions of the grantee?

In OC4J 10.1.3.x implementations, a policy is represented by a `javax.security.auth.Policy` instance. This class is deprecated as of JDK 1.4, but still fully supported in OC4J 10.1.3.x implementations and the Sun Microsystems JDK and J2SE.

In OC4J, JAAS policies are declared within the `<jazn-policy>` element of the `system-jazn-data.xml` file, or in Oracle Internet Directory if you are using the Oracle Identity Management security provider. (This functionality is comparable to that of a `.policy` file in the Java 2 Security Model.) These declarations grant permissions to users and roles, and are updated automatically when you use the OracleAS JAAS Provider Admintool to grant or revoke permissions.

JAAS Authorization: Subject Methods doAs() and doAsPrivileged()

The `Subject` class includes the following standard methods for authorization in the JAAS model:

- `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 (of the executing codesource), appending the subject's permissions to the permissions of that access control context and creating a new access control context with the combined permissions. Then the `AccessController.doPrivileged()` method is called with the specified action and the new 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

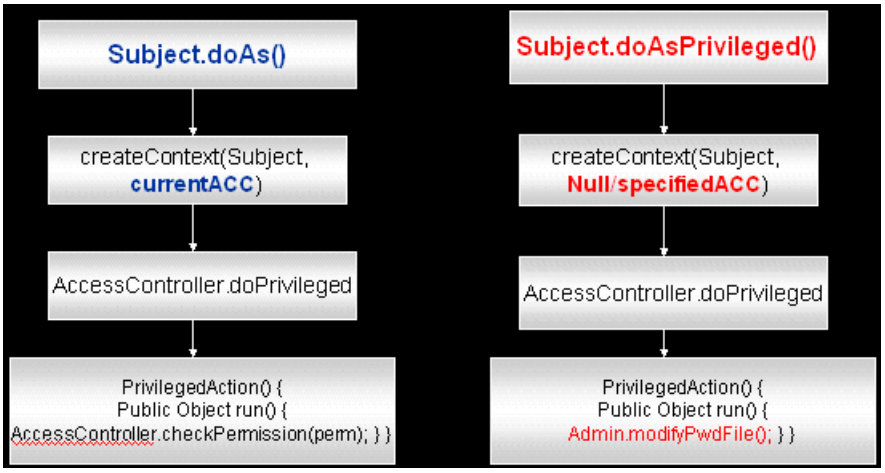
`java.security.PrivilegedAction`, for computations that can throw checked exceptions.

- `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. Typical usage is actually to specify a null access control context, if you prefer that the codesource not be required to have access privileges.

Figure 2–3 shows a sample code stack for the `doAs()` and `doAsPrivileged()` methods, in this case for modifying a password file. In the `doAsPrivileged()` case, if we assume that a null access control context is passed, then the subject must have sufficient privileges to access the password file.

Figure 2–3 Code Stack for doAs() and doAsPrivileged() Methods



Necessary Permissions for Principals for Subject doAs() Method

When you use the subject `doAs()` method with a privileged action, you must grant the necessary permissions for principals associated with the subject. You can use the OracleAS JAAS Provider Admintool to accomplish this; the resulting configuration appears under the `<jazn-policy>` element in the `system-jazn-data.xml` file.

The following example grants runtime permission "`setContextClassLoader`" to a `PrincipalImpl` principal named `myapp`:

```
% java -jar jazn.jar -grantperm sun.security.acl.PrincipalImpl \
      myapp java.lang.RuntimePermission setContextClassLoader
```

This results in the following configuration in `system-jazn-data.xml`:

```
<grant>
 <grantee>
   <principals>
    <principal>
      <class>sun.security.acl.PrincipalImpl</class>
      <name>myapp</name>
    </principal>
   </principals>
 </grantee>
```

```
  <permissions>
    <permission>
      <class>java.lang.RuntimePermission</class>
      <name>setContextClassLoader</name>
    </permission>
  </permissions>
</grant>
```

See Also:

- For the Admintool: "Granting and Revoking Permissions" on page C-14
- For the `system-jazn-data.xml` file: "The system-jazn-data.xml File" on page 4-7 and "Policy Configuration in system-jazn-data.xml" on page 5-15

Security Considerations during Development

This section discusses what you should consider during the development cycle for security. This includes a summary comparison between security models, and specific steps you should take in developing your application to ensure security.

Summary: Comparing Security Models for J2EE, Java 2, and JAAS

To summarize highlights of the J2EE, Java 2, and JAAS security models described in this chapter:

- In the J2EE authorization model, resources to be secured are identified by their URL patterns (for Web applications) or method names or name masks (for EJBs). Authorization is enforced by the container based on declarative role-based security defined in deployment configuration, with no implementation in your code. Once access is granted, any functionality of the resource is available. This model is relatively *coarse-grained*, but suffices for many purposes. It is also static, in that the developer or deployer must know about the user population beforehand.

  In the J2EE model, authentication is managed by the container.

- The Java 2 authorization model is code-based, restricting operations that applications can perform by applying sets of permissions (established in a Java 2 policy file that you define) that determine what code from any given codesource is allowed to do. (A codesource consists of a URL location of the code in question, and optionally an array of certificates.) Authorization checks are implemented in code, and performed by a security manager or access controller object. There is no static configuration for this model, other than the predefined policy files. This is an authorization model that is more dynamic and relatively *fine-grained*, where authorization is according to an adaptable security policy.

  As in the J2EE model, authentication is managed by the container.

- The JAAS authorization model extends Java 2 security, with policy objects able to handle queries based on principals, and the default policy implementation supporting grant entries based on principals. As with Java 2 security, authorization is enforced within application code. With the JAAS extensions, access control is based not only on the code that is executing, but also on who is executing it. A policy object is able to retrieve permissions granted to principals associated with a specified codesource. As with the Java 2 model, there is no static configuration involved, aside from policy files. The JAAS model is more customizable and extensible than the J2EE model, with features such as custom

permission types, and is also more fine-grained than the Java 2 model, given the ability to authorize according to principals. 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.

JAAS is also the only one of these models that supports customized authentication, through custom login modules. In addition, you can authenticate against multiple user repositories.

As appropriate and necessary, you can use any model or a combination of the models within an application. All three models are fully supported in OC4J. It is advisable to limit yourself to the J2EE authorization model whenever it meets your needs, given that it is the least complicated to deploy and administer. You can extend your application to use Java 2 or JAAS security depending on your needs for finer-grained code-based or subject-based security.

See Also:

- "Authorization Strategies" on page 5-20 for details on application security strategies

Steps to Develop a Secure J2EE Application

J2EE software development is based on a develop-deploy-manage cycle. The Oracle Application Server security implementation plays an important part in the deploy-manage part of the cycle. Developers can use the more convenient declarative security model instead of having to integrate security programmatically.

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, servlets, and application clients as desired.

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

2. The developer defines J2EE logical roles and assigns them privileges through security constraints, all through configuration in standard J2EE deployment descriptors.

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 J2EE roles to deployment users and roles, as discussed in "Specifying Security Role Mapping through Application Server Control" on page 6-11.

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

   This task includes creating and managing roles and users in the deployment environment as required by the application customers.

For finer-grained code-based or subject-based access control using Java 2 or JAAS features, there are also the following considerations:

1. The developer identifies any resources that may be accessed and must be protected as appropriate.
2. The developer defines permissions to protect these resources. (J2SE has predefined permissions that you can use where appropriate and sufficient.)
3. The developer implements code for runtime authorization checks.
4. The deployer, in consultation with the developer, defines the appropriate JAAS mode configuration for the application.
5. The system administrator maintains any necessary policy configuration to enforce the desired permissions. Policy provisioning (such as permission grants through the OracleAS JAAS Provider Admintool) should be completed prior to runtime.

See Also:

- *Oracle Application Server Best Practices* (available post-release) for information about best practices for security

3

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 covers the following:

- Introducing the OracleAS JAAS Provider and Security Providers
- Introducing Authentication Features in the OC4J Environment
- Introducing Authorization Features in the OC4J Environment
- Overview of Security Role Mapping
- Overview of General-Use Identity Management Frameworks and APIs

See Also:

- *Oracle Application Server Best Practices* (available post-release) for information about best practices for security
- 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.x 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

OC4J supplies a JAAS implementation, 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, Oracle Identity Management (LDAP directory-based), external LDAP directory, Oracle Access Manager, and custom (using custom login modules).

The rest of this section covers the following topics:

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 *JAVA_HOME*`/jre/lib/security/java.policy` 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. OC4J supplies login modules for the security providers it supports, or you can use custom login modules.
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 3–1 summarizes JAAS framework features implemented by the OracleAS JAAS Provider.

Table 3–1 JAAS Framework Features

Feature	Description	See Also
Authentication	■ Integrates with Oracle single sign-on solutions 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.	"Introducing Authentication Features in the OC4J Environment" on page 3-5
Declarative model	■ Integrates J2EE deployment descriptors, such as `web.xml` and `ejb-jar.xml`, with JAAS security.	
Authorization	■ Supports the J2EE authorization model. ■ Supports the JAAS authorization model. ■ Supports the Java Authorization Contract for Containers.	"Authorization APIs, JAAS Mode, and JACC in the OC4J Environment" on page 5-4
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 the command-line OracleAS JAAS Provider Admintool.	"Tools for Oracle Application Server and OracleAS JAAS Provider" on page 4-2
`JAZNUserManager`	■ Supplies a security provider implementation that integrates with the file-based provider, Oracle Identity Management, and Oracle Access Manager. This class is in the `oracle.security.jazn.oc4j` package.	

Security Realms in the OracleAS JAAS Provider

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 security domain that defines a set of permissions for authenticated users.

Each realm includes a set of configured users and roles. (In OC4J configuration, users and roles can all be configured within a realm definition.)

See Also:

- "Using Security Realms in OC4J" on page 6-3
- "Realm Management for the LDAP-Based Provider" on page 8-16

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 Oracle Internet Directory as your user repository, with or without Oracle Single Sign-On, as described in Chapter 8, "Oracle Identity Management". 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 Oracle 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 10, "External LDAP Security Providers". The external LDAP provider implements the 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 9, "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.

- Oracle Access Manager (formerly Oracle COREid Access and Identity)

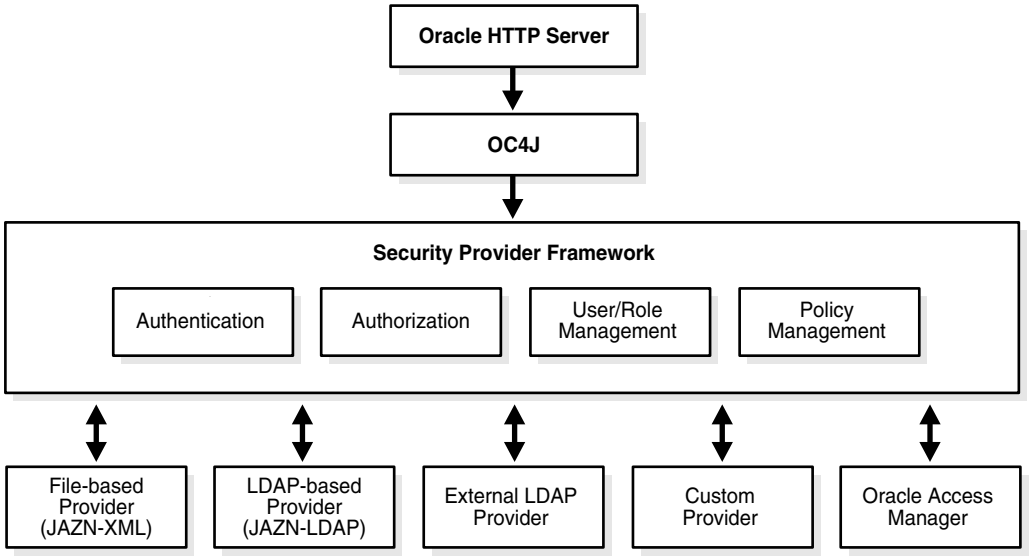
  Beginning with OC4J 10.1.3.x implementations, an additional choice for security provider is Oracle Access Manager, as described in Chapter 11, "Oracle Access Manager". This is an enterprise-class authentication, authorization, and auditing solution that provides centralized security administration. This includes functionality for access control, single sign-on (separate from Oracle Single Sign-On), personalization, and user profile management in heterogeneous application environments across a variety of application servers, legacy applications, and databases. Oracle Access Manager implements the login module `CoreIDLoginModule`.

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 OC4J 10.1.3.x implementations, the term "custom security provider" is essentially synonymous with "custom login module".

Figure 3–1 shows how the supported security providers interact with the overall security provider framework.

Figure 3–1 OC4J Security Architecture



Introducing Authentication Features in the OC4J Environment

This section discusses the following topics relating to authentication in OC4J:

- Supported Web Application Authentication Methods
- Overview of OC4J Login Modules
- Overview of Oracle Application Server Single Sign-On Alternatives
- JAZNUserManager Delegation (File-Based Provider)

Supported Web Application Authentication Methods

OC4J supports the standard basic, digest, form, and client-cert authentication methods. These are summarized in "Web Application Standard Authentication Methods" on page 2-2, and configured in the standard `web.xml` file. Details are provided in "Specifying the Authentication Method (auth-method)" on page 17-1.

OC4J also supports various single sign-on authentication methods supplied with Oracle Application Server and OC4J. These are summarized in "Overview of Oracle

Application Server Single Sign-On Alternatives" below, which includes cross-references for further information.

Overview of OC4J Login Modules

Oracle supplies the following login modules:

- `RealmLoginModule`, the default login module for the file-based provider or Oracle Identity Management, discussed in "RealmLoginModule" on page 9-4
- `LDAPLoginModule`, for external LDAP providers, discussed in "Overview of External LDAP Provider Configuration and Administration" on page 10-2
- `DBTableOraDataSourceLoginModule`, for user repositories in a database (replaces previous functionality of the `DataSourceUserManager` class), discussed in "DBTableOraDataSourceLoginModule" on page 9-5
- `CoreIDLoginModule`, for use with Oracle Access Manager, discussed in "Configure the Oracle Access Manager Login Module" on page 11-18

OC4J also supports any custom login module that adheres to login module standards, as discussed throughout Chapter 9, "Login Modules".

Overview of Oracle Application Server Single Sign-On Alternatives

Single sign-on is a feature that lets a user log in just a single time to access multiple Web applications within an OC4J instance or cluster. The following single sign-on authentication methods are Oracle-specific. Usage of these methods is indicated by configuration in the Oracle `orion-application.xml` file rather than the standard `web.xml` file.

- **SSO**

  For this authentication method, OracleAS Single Sign-On is used to authenticate users. This requires an Oracle Application Server infrastructure that includes Oracle Identity Management, Oracle Internet Directory, and OracleAS Single Sign-On. You can find information about OracleAS Single Sign-On in Chapter 8, "Oracle Identity Management".

- **COREIDSSO**

  For this authentication method, Oracle Access Manager single sign-on (distinct from OracleAS Single Sign-On) is used to authenticate users. This requires an infrastructure that includes Oracle Access Manager. You can find information about Oracle Access Manager SSO in Chapter 11, "Oracle Access Manager".

- **CUSTOM_AUTH** (for Java Single Sign-On)

  New with the OC4J 10.1.3.1 implementation, Java SSO is a an alternative SSO solution packaged with OC4J for customers who desire a smaller deployment environment. When Java SSO is properly configured and enabled, a `CUSTOM_AUTH` setting (which is for the OC4J identity management framework, introduced in "Overview of General-Use Identity Management Frameworks and APIs" on page 3-8) will enable Java SSO (which is a default implementation of the identity management framework). Java SSO is packaged as part of the OC4J container itself and requires no additional infrastructure. You can find information about Java SSO in Chapter 14, "OC4J Java Single Sign-On".

JAZNUserManager Delegation (File-Based Provider)

The OracleAS JAAS Provider `JAZNUserManager` coordinates authentication in OC4J. For a Web application, before HTTP requests can be dispatched to a target servlet, `JAZNUserManager` gets the authenticated user information (set by the Oracle HTTP Server `mod_osso` module for SSO, for example) from the HTTP request object, and sets the JAAS subject in OC4J.

`JAZNUserManager` supports a delegation model as detailed below, but effectively this applies only to the file-based provider. With delegation, if a user or group is not found at the application-level `JAZNUserManager` instance, the request is delegated to the parent user manager.

Specifically, note the following restrictions and additional details:

- If the application and parent application are both configured to use the file-based provider, delegation goes up through the parent hierarchy as far as necessary, until a parent is not configured to use the file-based provider. Delegation is not propagated beyond that point.
- If the application is configured to use the file-based provider, and the parent is configured to use the LDAP-based provider, an external LDAP provider, or a custom login module, there is no delegation support.
- If the application itself is configured to use the LDAP-based provider, an external LDAP provider, or a custom login module, there is no delegation support.

Important: This also applies with a developer-supplied `UserManager` implementation instead of `JAZNUserManager`. However, developer-supplied `UserManager` classes are deprecated in OC4J 10.1.3.x implementations, and will be desupported in a future release. Use custom login modules instead.

Note: In OC4J, the `system` application is at the root of the hierarchy, but the `default` application is the default parent of any deployed application. Both use `system-jazn-data.xml` as the user repository.

See Also:

- "Web Application Standard Authentication Methods" on page 2-2

Introducing Authorization Features in the OC4J Environment

Authorization in the OC4J environment includes the following features, detailed in Chapter 5, "Authorization in OC4J":

- Support for Java 2 security and code-based policy management, including use of the standard `java2.policy` file
- "JAAS mode" setting, which determines security behavior related to functionality of the standard `Subject` class `doAs()` and `doAsPrivileged()` methods
- Granting, checking, and revoking permissions; Oracle permission classes `RMIPermission`, `AdminPermission`, `RoleAdminPermission`, `JAZNPermission`, and `RealmPermission`
- JAAS policy management

- Implementation of the Java Authorization Contract for Containers

Overview of Security Role Mapping

OC4J enables you to map a *J2EE logical role* (or simply "J2EE role") defined in a standard descriptor to a *deployment role* (referred to as a *JAAS role* in a previous version of this document) so that a user who is a member of a given deployment role has access to resources that are accessible from the associated J2EE role. Deployment roles are defined in the security provider—for example, in `system-jazn-data.xml` for the file-based provider, in Oracle Internet Directory for the LDAP-based provider, or in an external LDAP provider, custom login module, or Oracle Access Manager.

The basic steps in security role configuration and mapping are as follows:

1. Specify J2EE logical 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 J2EE role is declared in a `<security-role>` element.
2. As applicable, specify security role references to link *application logical roles* (roles defined in your application code) to J2EE 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 link J2EE logical roles to application roles as desired. There is nothing OC4J-specific in this step.
3. Configure deployment roles, or use default roles. For the file-based provider, for example, deployment roles are defined in the OC4J `system-jazn-data.xml` file, or optionally in an application-specific `jazn-data.xml` file. For the LDAP-based provider, deployment roles are defined in Oracle Internet Directory.
4. Map J2EE roles to deployment 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 Security Roles" on page 6-10
- "Web Application Security Role and Constraint Configuration" on page 17-6
- "Authenticating and Authorizing EJB Applications" on page 18-1

Overview of General-Use Identity Management Frameworks and APIs

The OC4J 10.1.3.1 implementation includes new general-purpose support for third-party identity repositories, with the following features:

- A pluggable identity management framework that allows integration of heterogeneous third-party systems into OC4J, hence allowing any J2EE application to interoperate with these third-party systems. Integration of third-party identity management systems into OC4J is based on standard JAAS login modules.

  Chapter 13, "Pluggable Identity Management Framework" describes how to use this framework, with discussion of its programmatic interfaces, configuration features, and so on.

Note that Java SSO, an alternative Java single sign-on solution that does not rely on additional infrastructure required by other single sign-on products, is implemented through the pluggable identity management framework. Java SSO is discussed in Chapter 14, "OC4J Java Single Sign-On".

- An identity management API framework for accessing user and role information from disparate identity management repositories. This user and role API framework allows applications to access identity information (users and roles) in a uniform and portable manner regardless of the particular underlying identity repository. The underlying repository could be an LDAP directory server such as Oracle Internet Directory, Active Directory (from Microsoft), or Sun Java System Directory Server (from Sun Microsystems), or could be a database, flat file, or some other custom repository. Supported operations include searching, creating, updating, or deleting users and roles.

  To avoid confusion with the pluggable identity management framework discussed in Chapter 13, which is independent, we refer to the identity management API framework as the "user and role APIs" or "user and role API framework". Chapter 12, "User and Role API Framework", describes how to use these APIs.

4

Overview of Security Administration

This chapter 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

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 policies. 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 4-9).

See Also:

- "Summary of Configuration Repositories and Security Management Tools" on page 4-15

Overview of Oracle Enterprise Manager 10*g* Application Server Control

Typically, you should use 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 4-5). 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 when 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 4-10

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, and policies
- For Oracle Identity Management: administration for policies, 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 you use 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 (DAS) 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 DAS 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 DAS. 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 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 4-15
- Reference appendix in the *Oracle Containers for J2EE Developer's Guide* for information about elements of the `orion-application.xml` (and `system-application.xml`) file

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 4-7, 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.

Notes:

- In order to take effect, changes to `orion-application.xml` require an application restart (if the changes were made through Application Server Control or security provider MBean) or an OC4J restart (if the changes were made manually).
- 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 `system-jazn-data.xml` repository and `jazn.com` default realm are the default settings).

The system-application.xml File

This OC4J configuration file is associated with the OC4J `system` application, which is described in "OC4J System Application" on page 4-10. 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 OC4J 10.1.3.x implementations. This file (as well as `system-application.xml`) is associated with the OC4J `system` application, which is described in "OC4J System Application" on page 4-10.

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.)

If you use the file-based provider for an application, you can optionally use `system-jazn-data.xml` as your user repository, or you can use an application-specific `jazn-data.xml` file that you package with your application, as described in the next section, "Application-Specific jazn-data.xml File (Optional)".

The `system-jazn-data.xml` file also stores JAAS login module configuration (under the `<jazn-loginconfig>` element) and JAAS policy configuration (under the `<jazn-policy>` element).

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.
- "`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>
```

See Also:

- Appendix D, "OracleAS JAAS Provider Configuration Files" for details about the hierarchy, elements, and attributes of this file

---

Notes:

- If changes to `system-jazn-data.xml` are made through Application Server Control or the security provider MBean, then an OC4J restart is *not* required for the changes to take effect. A restart is required, though, if changes are made manually (which is not generally recommended).
- 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 use a `jazn-data.xml` file as the user and role repository. In OC4J 10.1.3.x implementations, this file is application-specific. You can specify its location in the `<jazn>` element of the `orion-application.xml` file, according to where you deploy it:

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

See Also:

- "Supplying an Application-Specific jazn-data.xml File" on page 7-11
- Appendix D, "OracleAS JAAS Provider Configuration Files" for details about the hierarchy, elements, and attributes of this file

Note that if `orion-application.xml` is configured exactly as follows, but a `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: Changes to `jazn.xml` require an OC4J restart in order to take effect.

See Also:

- Appendix D, "OracleAS JAAS Provider Configuration Files" for details about the hierarchy, elements, and attributes of this file

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

The OC4J `system` application is an internal component defined in OC4J 10.1.3.x implementations. 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, as in previous OC4J implementations.)

By default, the `system` application is configured to use the file-based provider for user and role settings, using `system-jazn-data.xml` for the repository. (Note this means that authorization of all system resources, such as MBeans and `admin_client.jar`,

is based on entries in `system-jazn-data.xml`.) It is not advisable to alter this configuration.

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, and by default this configures the `default` application to also use the `system-jazn-data.xml` repository.) These files are located in the *ORACLE_HOME*`/j2ee/`*instance_name*`/config` directory.

See Also:

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

Summary of OC4J Accounts

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

- Predefined Accounts
- Activation of the oc4jadmin Account (Standalone OC4J)
- Creating and Configuring a New Administrator Account
- Configuring an Anonymous User

Predefined Accounts

OC4J 10.1.3.x implementations include 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 (OID), 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`)

  This is the default administrator account. During the Oracle Application Server or OC4J installation process, you must specify a password for this account. You can then use this account to log in to the Application Server Control Console. This account is also used by Application Server Control when it performs administrative functions (through the OC4J `system` application).

  In a cluster environment, the `oc4jadmin` name and password serve as administrative credentials for the cluster. Because there is only one set of administrative credentials for a cluster, by default each OC4J instance must have an `oc4jadmin` account with the same password. For this reason, use caution when changing the password for this account.

  As a good practice, you may consider creating additional administrative accounts rather than using `oc4jadmin` for day-to-day administrative activities.

See Also:

- "Creating and Configuring a New Administrator Account" on page 4-13

- `oc4j-administrators` role (formerly `administrators`), with member `oc4jadmin`, RMI permission "login" granted, and administration permission "administration" granted (`com.evermind.server.rmi.RMIPermission` and `oracle.security.jazn.policy.AdminPermission`, respectively)
- `oc4j-app-administrators` role (formerly `jmx-users`), with RMI permission "login" granted, to allow access to JMX application-level connectors
- `ascontrol_admin` (administrative role for all SOA controls, including Application Server Control), with member `oc4jadmin`
- `ascontrol_appadmin` (Application Server Control required role)
- `ascontrol_monitor` (Application Server Control required role)

Important: When you use an external LDAP security provider, you must take manual steps (through whatever provisioning tool is appropriate for the provider) to create the preceding user and role accounts and grant the `oc4j-administrators` and `ascontrol_admin` roles to `oc4jadmin`. Also use the OracleAS JAAS Provider Admintool to grant the noted permissions.

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

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

Do not remove any of the accounts described in this section, or the administrative functions of the OracleAS JAAS Provider will not work.

See Also:

- The next section, "Activation of the oc4jadmin Account (Standalone OC4J)"
- Application Server Control online help topics: "About the oc4jadmin User", "About Administrative Users and Roles" (for additional information about the Application Server Control roles), and "Best Practices When Defining Administration Users and Roles".

Activation of the oc4jadmin Account (Standalone OC4J)

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
  ```

Creating and Configuring a New Administrator Account

By default, as noted previously, `oc4jadmin` is the administrator account for OC4J. This section discusses how to use an alternative administrator account, such as to access SOA components (including the Application Server Control Console and Oracle Web Services Manager, for example) and for internal use by the OC4J `system` application. The `system` application is used by Application Server Control for various administrative functions (such as deploying and undeploying applications).

Creating a New Administrator Account

You may choose to create and use a new administrator account, rather than `oc4jadmin`. This could be to access SOA components such as the Application Server Control Console, for example (typically through Java SSO), which is the scenario we consider here.

The steps to take depend on whether the security provider for SOA components is the file-based provider or Oracle Identity Management, both of which have key administrator roles defined by default with necessary RMI permissions, or some other security provider such as an external LDAP provider, where you must configure these roles and permissions manually.

Notes: The following additional points are applicable to Application Server Control:

- For Application Server Control to work properly in a clustered environment, you must create the new user across the cluster.
- If you remove the `oc4jadmin` account, you will see the Application Server Control "post login" page. This is expected, because Application Server Control would no longer be able to make a connection with the previously cached `oc4jadmin` user. Once you enter the new user name and password, you will not see this page again.

For the File-Based Provider or Oracle Identity Management If you use the file-based provider or Oracle Identity Management as the security provider for SOA components, the following steps are required to use a custom administrator account:

1. Create the new administrative user account.
2. Grant the new administrative user account the roles `oc4j-administrators` and `ascontrol_admin`.
3. Optionally set the `admin.user` property to allow the OC4J `system` application to use the new account, as discussed in "Configuring a New Administrator Account for the System Application" on page 4-15. If you are not using `system-jazn-data.xml` as your repository, then this step requires you to also create your new administrator account in `system-jazn-data.xml`, for use by the `system` application, and grant it the necessary administrator roles.

No further steps are required. By default, the `oc4j-administrators` and `ascontrol_admin` roles exist in the file-based provider and Oracle Internet Directory, and have RMI permission "login" granted.

Notes:

- For the file-based provider, use the Application Server Control Console for user and role management, as detailed in "Configuring the File-Based Provider in Application Server Control" on page 7-2.
- For Oracle Internet Directory, use the Oracle Delegated Administration Services (DAS) for user and role management, as detailed in the *Oracle Identity Management Guide to Delegated Administration*.

For Other Security Providers If you use anything other than the file-based provider or Oracle Identity Management as the security provider for SOA components, the following steps are required to use a custom administrator account:

1. Create the new administrative user account, using the appropriate tool for the security provider.
2. Create the administrator roles `oc4j-administrators` and `ascontrol_admin`, using the appropriate tool, if you have not already created them. For an external LDAP provider, these roles must be under the group search base configured for the LDAP provider. (See Table 10–4, " Application Server Control External LDAP Role and Member Options" on page 10-5 and Table 10–7, " External LDAP Role and Member Options" on page 10-8 for information about the group search base.)
3. Grant the new administrative user account the roles `oc4j-administrators` and `ascontrol_admin`, using the appropriate tool, if you have not already done so.
4. Create the additional administrator roles `oc4j-app-administrators`, `ascontrol_appadmin`, and `ascontrol_monitor`. (These do not have to be granted to the administrative user.)
5. If the administrator must access EJBs, grant the roles `oc4j-administrators` and `ascontrol_admin` RMI permission "login", if you have not already done so. You can use the OracleAS JAAS Provider Admintool for this, as in the following example:

   ```
   % java -jar jazn.jar -grantperm myrealm -role oc4j-administrators \
        com.evermind.server.rmi.RMIPermission login
   ```

   Be aware that although the roles are defined in the external LDAP provider, these permission grants are stored in the `system-jazn-data.xml` file.
6. Optionally set the `admin.user` property to allow the OC4J `system` application to use the new account, as discussed in the next section, "Configuring a New Administrator Account for the System Application". This step requires you to also create your new administrator account in `system-jazn-data.xml`, for use by the `system` application, and grant it the necessary administrator roles.

See Also:

- "Creating the Administrative User and Roles and Granting RMI Permission" on page 10-9

Configuring a New Administrator Account for the System Application

You can specify a different administrator account for use by the OC4J `system` application as follows:

1. Create the desired account in the `system-jazn-data.xml` file, if it does not already exist there, and grant it the roles `oc4j-administrators` and `ascontrol_admin`. Use the Application Server Control Console for user and role management, as detailed in "Configuring the File-Based Provider in Application Server Control" on page 7-2.
2. Set the `admin.user` property in the instance-level `jazn.xml` file, as follows:

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

(These steps assume the OC4J `system` application is configured to use the file-based security provider, as should always be the case.)

The specified account will then be used for a variety of administrative functions, including, for example, being used by Application Server Control to deploy and undeploy applications.

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 4–1 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 4–1 Configuration Repositories and Preferred Management Tools

Security Provider	Repository and Management Tool for Realms, Users, 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 (preferred) or OracleAS JAAS Provider Admintool.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	n/a

Table 4–1 (Cont.) Configuration Repositories and Preferred Management Tools

Security Provider	Repository and Management Tool for Realms, Users, Roles	Repository and Management Tool for Policies	Repository and Management Tool for JAAS Login Modules
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 (preferred) 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 (preferred) or OracleAS JAAS Provider Admintool.
Oracle Access Manager	Oracle Access Manager Use the Policy Manager tool.	system-jazn-data.xml Use OracleAS JAAS Provider Admintool.	system-jazn-data.xml Use Application Server Control Console (preferred) or OracleAS JAAS Provider Admintool.

Note: "Policies" in this table refers to subject-based policies. Code-based policies are stored in a standard Java 2 policy file (such as `java2.policy` or `java.policy`). OC4J currently offers no management tool to update or maintain Java 2 policy files.

5

Authorization in OC4J

Chapter 2, "Java Platform Security" provided an overview of three major Java security models: J2EE role-based security, Java 2 code-based security, and JAAS subject-based security.

This chapter shows how to use the authorization features with each of these models. They can be used individually or in combinations. At the end of the chapter there is discussion of how to choose between security models.

The following topics are covered:

- Java 2 Security and Code-Based Policy Management
- Authorization APIs, JAAS Mode, and JACC in the OC4J Environment
- OracleAS JAAS Provider Policy Management
- Authorization Coding and Configuration
- Authorization Strategies

Java 2 Security and Code-Based Policy Management

This section discusses security managers and policy files for Java 2 (code-based) security, covering the following topics:

- Specifying a Java 2 Security Manager and Policy File
- Using PrintingSecurityManager to Determine Required Java 2 Permissions
- Creating or Updating a Java 2 Policy File

Specifying a Java 2 Security Manager and Policy File

For Java 2 (code-based) security policies to be enforced by OC4J and the underlying JDK, a security manager (`java.lang.SecurityManager` instance) must be enabled. This may affect, for example, access to class loaders, access to JDK resources, execution of JDK APIs, or execution of JAAS APIs (such as the ability for the `Subject.doAs()` or `Subject.doAsPrivileged()` method to be executed). With a security manager enabled, the policies specified in the Java 2 policy file determine the resources that executing code can access.

You can specify a security manager in either of the following ways:

- Call the static `setSecurityManager(SecurityManager)` method of the `java.lang.System` class to specify any desired security manager.

- Use the system property `java.security.manager` when starting OC4J, either with no setting to use the default security manager, or with a setting that indicates a desired security manager.

The following example starts OC4J with the default security manager:

```
% java -Djava.security.manager ... -jar oc4j.jar
```

The following example specifies a security manager:

```
% java -Djava.security.manager=com.abc.MySecurityManager ... -jar oc4j.jar
```

Notes:

- Using a security manager is not typically necessary in OC4J. Within an Oracle Application Server installation, OC4J instances run by default with no security manager. If you choose to install a security manager, there may be significant performance impact. If you use a custom security manager, ensure that it does not interfere with OC4J functions.
- Assuming you use the default implementation of `AccessController` (provided with the Sun JDK), a call to `AccessController.checkPermission()`, discussed in "Using the checkPermission() Method" on page 5-17, enforces Java 2 security policy for your application regardless of whether a security manager is enabled.

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 the J2SE. You can specify any desired policy file by setting the system property `java.security.policy` to the full path of the desired file. The following example starts OC4J with the default security manager and the policy file that is supplied with OC4J:

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

Notes:

- Be aware that this `java2.policy` file includes permissions that are required for OC4J to run with a security manager. If you use a different policy file, make sure the same permissions are included in that file.
- A `.policy` file is for Java 2 (code-based) policies only. With the OracleAS JAAS Provider, JAAS (subject-based) policies are declared within the `<jazn-policy>` element of the `system-jazn-data.xml` file, or in Oracle Internet Directory if you are using the Oracle Identity Management security provider, as discussed in "OracleAS JAAS Provider Policy Configuration" on page 5-14.

See Also:

- The following sections, "Using PrintingSecurityManager to Determine Required Java 2 Permissions" and "Creating or Updating a Java 2 Policy File", for related information
- "Java 2 Authorization: Security Managers and Access Controllers" on page 2-11 for an overview of security managers

Using PrintingSecurityManager to Determine Required Java 2 Permissions

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 in the following example, 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 permissions.)

`PrintingSecurityManager` generates output that lists the following:

- Which codesources require which permissions
- Policy grants 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 is for messages about missing permissions; the second is 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 alternatively use it outside of OC4J.

Creating or Updating a Java 2 Policy File

A Java 2 policy file grants permissions to trusted code or applications, allowing them appropriate access privileges to execute properly in your environment.

A preconfigured Java 2 policy file, `java2.policy`, is provided in *ORACLE_HOME*`/j2ee/home/config`. You can modify this file if desired, or create and specify an alternative policy file. Be aware, however, that the `java2.policy` file supplied by Oracle includes permissions that are required for OC4J to run with a security manager. If you use a different policy file, make sure the same permissions are included in that file.

The following policy file sample grants all permissions, such as opening system files and opening sockets or ports, to the trusted `jazn.jar` (the OracleAS JAAS Provider Admintool):

```
/* grant the JAAS library AllPermission */
grant codebase "file:${oracle.home}/j2ee/home/jazn.jar" {
    permission java.security.AllPermission;
};
```

Similarly, the following example grants all permissions to `wssecurity.jar` (for Web services security functions):

```
/* grant the WSSecurity AllPermission */
grant codebase "file:${oracle.home}/webservices/lib/wssecurity.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.home}/appdemo/-" {
   permission oracle.security.jazn.JAZNPermission "getPolicy";
   permission oracle.security.jazn.JAZNPermission "getRealmManager";
   permission oracle.security.jazn.policy.AdminPermission;
}
```

You can grant additional permissions for your application code or for classes generated by OC4J, as necessary, by manually creating additional entries in the `.policy` file. (There is currently no tool for this.) The required permissions will depend on the details of your application, and the required codebase will depend on the details of your installation.

Note: Note the use of "`${oracle.home}`" to specify the location of *ORACLE_HOME*. This is an environment variable that is set appropriately by default.

See Also:

- For details about policy file syntax:

  http://java.sun.com/j2se/1.5.0/docs/guide/security/PolicyFiles.html

Authorization APIs, JAAS Mode, and JACC in the OC4J Environment

This section discusses the following authorization features in OC4J:

- JAAS Authorization and OracleAS JAAS Provider JAAS Mode
- Setting a Subject Into the OC4J Container
- Implementation of Java Authorization Contract for Containers

JAAS Authorization and OracleAS JAAS Provider 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 builds upon the J2SE standard, using standard APIs to support the following features related to fine-grained authorization:

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

Note: APIs documented here can be used with either `system-jazn-data.xml` or Oracle Internet Directory as the policy repository.

See Also:

- "Authorization Coding and Configuration" on page 5-16 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/`

- For details about permissions:

  `http://java.sun.com/j2se/1.5.0/docs/guide/security/permissions.html`

- *Oracle Containers for J2EE Security Java API Reference* for Javadoc for the OracleAS JAAS Provider

Introduction to JAAS Mode

OC4J 10.1.3.x implementations provide a fine-grained authorization feature called *JAAS mode*, which is related to standard functionality of the `Subject` class static methods `doAs()` and `doAsPrivileged()`, discussed in "JAAS Authorization: Subject Methods doAs() and doAsPrivileged()" on page 2-15.

These methods are used with 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 (Web modules and EJBs) are executed by OC4J within a `Subject.doAs()` block.

  This mode is useful if you want code-based security together with subject-based security.

- With the setting `jaas-mode="doAsPrivileged"`, application modules are executed within a `Subject.doAsPrivileged()` block, using a null access control context.

  This mode is useful if you want subject-based security only.

- With the setting `jaas-mode="null"` (default), modules are executed under neither method.

  This mode is useful if you want code-based security only.

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

Important:

- Set `jaas-mode` in `orion-application.xml` only, not in `jazn.xml`.
- If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any previous `<jazn>` settings would be lost and would have to be redone.

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

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

These settings are deprecated in OC4J 10.1.3.x implementations, 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:

- "Java 2 Authorization: Security Managers and Access Controllers" on page 2-11 for an introduction to access controllers and access control contexts
- "Configuring and Using JAAS Mode" on page 5-18

OracleAS JAAS Provider **Realm and Policy APIs**

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.

See Also:

- Chapter 12, "User and Role API Framework" for information about new user and role APIs that are available, and in future releases will replace some of the APIs discussed here

OracleAS JAAS Provider APIs for Granting or Revoking 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 JAAS (subject-based) 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` class includes a constructor that takes a `Principal` instance as input:

- `new Grantee(Principal)`

Table 5–1 summarizes permission classes supplied by Oracle.

Note: Features to grant permissions to a role replace functionality in the deprecated `com.evermind.security.Group` class.

See Also:

- For information about these classes, the OracleAS JAAS Provider Javadoc: *Oracle Containers for J2EE Security Java API Reference*

Table 5–1 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`). Extends `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.
`RMIPermission`	`com.evermind.server.rmi`	This permission is required for access to EJBs over the ORMI protocol. Typical usage is `RMIPermission login` (such as from the OracleAS JAAS Provider Admintool).

You can construct instances of these permission classes as follows:

- `new AdminPermission(String)`
- `new AdminPermission(Permission)`

  The `AdminPermission` constructor takes an encoded permission string or a `java.security.Permission` instance.

- `new RoleAdminPermission(String)`
- `new RoleAdminPermission(RealmRole)`

  The `RoleAdminPermission` constructor takes one of the following:

  - A string indicating the name of the role whose administrative permission is to be granted, of the form *realmname/rolename*. Specifying just "*" will match all roles in the system; specifying "*realm/**" will match all roles in the realm.
  - An `oracle.security.jazn.realm.RealmRole` instance, which is an instance of a class that implements the `RealmRole` interface, representing a role associated with a particular realm.

- `new JAZNPermission(String)`

  The `JAZNPermission` constructor takes a symbolic name of the permission, such as "`getRealmManager`", "`getPolicy`", "`getProperty.`*`propertyname`*", and so on.

- `new RealmPermission(String realm, String actions)`

  The `RealmPermission` constructor takes a string for the name of the realm, and a string consisting of a comma-delimited list of the actions applicable to the realm.

- `new RMIPermission(String param, String actions)`

  The `RMIPermission` constructor takes a string for the `RMIPermission` parameter (`login`, for example) and a string consisting of a comma-delimited list of applicable actions.

You can also construct instances of standard permission classes, as needed:

- `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.BasicPermission` 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.

Important: `AllPermission` should be used with caution, and only when necessary.

Note: 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.

APIs for Checking Permissions

Access control, such as checking permissions, was introduced in "Java 2 Authorization: Security Managers and Access Controllers" on page 2-11. Any of the following APIs may be involved in retrieving and checking permissions.

The `java.security.AccessController` class includes the following methods:

- `AccessControlContext getContext()`

  This method examines the current calling context, which includes the inherited access control context of the current thread, and represents the context as a `java.security.AccessControlContext` instance.

- `void checkPermission(Permission)`

  Given a `java.security.Permission` instance, this static method checks whether the access request indicated by the permission should be allowed, and throws an `AccessControlException` if it should be denied. This method uses the default access control context.

  Note that this method enables you to check against Java 2 policy (as applicable) without a security manager enabled.

The `java.security.AccessControlContext` class includes the following method:

- `void checkPermission(Permission)`

  This has the same functionality as the `AccessController.checkPermission()` method, but is called on a particular `AccessControlContext` instance to use that access control context. (You can construct an `AccessControlContext` instance from an array of `ProtectionDomain` instances.)

  Note that this method enables you to check against Java 2 policy (as applicable) without a security manager enabled.

The `java.lang.SecurityManager` class includes the following method:

- `void checkPermission(Permission)`

  This has the same functionality as the `checkPermission()` methods of the `AccessController` and `AccessControlContext` classes, but is used with a security manager in place, is called against the security manager instance, and throws a `SecurityException` if access should be denied.

The `javax.security.auth.Subject` class includes the following method:

- `Subject getSubject(AccessControlContext)`

  This static method returns the subject that is associated with the specified access control context. You can specify the default access control context as follows:

  ```
  mysubject = Subject.getSubject(AccessController.getContext());
  ```

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 or a `java.security.CodeSource` instance or both (inputs can be null), returns a

`java.security.PermissionCollection` instance that indicates the set of permissions allowed. 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. For example, if the permission collection consists of permissions of a subject, this method tells you if the subject has the specified permission, such as in the following example:

```
jaaspolicy = javax.security.auth.Policy.getPolicy();
jaaspolicy.getPermissions(mysubject,null).implies(perm);
```

> **Note:** The `javax.security.auth.Policy` class is deprecated in JDK 1.4 but fully supported in OC4J 10.1.3.x implementations and still supported by the Sun Microsystems JDK and J2SE. As of this release, OC4J itself does not support the `java.security.Policy` class.

See Also:

- "Java 2 Authorization: Java 2 Security Policies" on page 2-10 for an introduction to codesources

Setting a Subject Into the OC4J Container

JAAS based authentication is not well integrated with Java EE. Typically JAAS authentication requires the use of a login module. However, a login module (including custom login modules) do not assert the Subject into the container. This behavior causes applications to not use the authenticating subject. To solve this problem OC4J provides an API that allows a Subject to be asserted into the OC4J container that can be used in place of container managed security (declarative security).

The API is located in the `oracle.oc4j.security` package and is used as follows:

```
Security.setSubject(Subject subject, Longevity longevity);
```

The `setSubject` method asserts the Subject identity into the container for this request and all further requests associated with the Web container if applicable. If the current application context does not have a Web component, then only the current application server thread context will make use of the Subject and upon returning to the pool it will be cleared. This API is only relevant for server-side execution and not for application client-side code.

- `subject` – the identity to assert into the container
- `longevity` – the duration for which the identity assertion should be held. This can be `Longevity.REQUEST` or `Longevity.SESSION`.

Implementation of Java Authorization Contract for Containers

OC4J 10.1.3.x implementations support the Java Authorization Contract for Containers (JACC), 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 JACC provider.

Note that as a result of the JACC contract, J2EE security constraints are translated into Java 2 permissions, so that the J2EE security model fully leverages the J2SE security

model. Yet JACC still fully preserves the existing J2EE declarative security model as well as the J2EE security API. Essentially, you can enable JACC to use an extended version of J2EE authorization, using the same configuration and code for authorization in your application.

JACC is typically invisible to a developer; it is more of a deployment-time consideration.

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

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

- Defining roles 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

See Also:

- "Enabling the Java Authorization Contract for Containers" on page 5-19
- For general information about JACC:

  `http://java.sun.com/j2ee/javaacc/`
  `http://www.jcp.org/en/jsr/detail?id=115`

OracleAS JAAS Provider Policy Management

There is not a set standard for subject-based policy management; the implementation is left up to each vendor. This section discusses how to use OracleAS JAAS Provider features for subject-based policy management—granting permissions, the resulting configuration, checking permissions, and (where necessary) explicitly specifying the OracleAS JAAS Provider policy provider:

- Granting Permissions through the OracleAS JAAS Provider Admintool
- Using OracleAS JAAS Provider Policy Management APIs
- OracleAS JAAS Provider Policy Configuration
- Specification of the Oracle Policy Provider

See Also:

- "Java 2 Security and Code-Based Policy Management" on page 5-1 regarding code-based policy

Granting Permissions through the OracleAS JAAS Provider Admintool

With either `system-jazn-data.xml` or Oracle Internet Directory as the policy repository, you can use the OracleAS JAAS Provider Admintool to grant, revoke, or

list permissions—using the `grantperm`, `revokeperm`, or `listperm` command, respectively. (An alternative, for runtime, is to use APIs discussed in the next section, "Using OracleAS JAAS Provider Policy Management APIs".) Complete syntax for these commands is documented in "Granting and Revoking Permissions" on page C-14 and "Listing Permissions" on page C-15.

You can grant permissions to a user (through the `grantperm -user` option), to a role (through the `-role` option), or to a principal.

For example, to grant RMI permission "login" to the role `developers` in the realm `myrealm`:

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

(RMI permission "login" is a permission you must often grant to allow EJB/RMI access.)

To grant this same permission to the user `JDOE_ENDUSER`:

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

To grant this permission to the LDAP principal `hobbes` (when using an external LDAP provider):

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.LDAPPrincipal hobbes \
       com.evermind.server.rmi.RMIPermission login
```

To grant `FilePermission` for a file `sample.txt` for actions "`read, write`" to user `martha` in realm `foo`:

```
% java -jar jazn.jar -grantperm foo -user martha java.io.FilePermission
     sample.txt read,write
```

Resulting configuration from the `grantperm` command is discussed in "OracleAS JAAS Provider Policy Configuration" on page 5-14.

See Also:

- "OracleAS JAAS Provider Policy Configuration" on page 5-14 regarding configuration resulting from permission grants

Using OracleAS JAAS Provider Policy Management APIs

With either `system-jazn-data.xml` or Oracle Internet Directory as the policy repository, you can use OracleAS JAAS Provider policy management APIs to grant or revoke permissions. (An alternative, for nonruntime, is to use OracleAS JAAS Provider Admintool commands discussed in the preceding section, "Granting Permissions through the OracleAS JAAS Provider Admintool".) In this example, the `doGet()` method shown in "Using J2EE Authorization APIs" on page 5-17 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.isUserInRole('ar_developers') = " +
```

```
                request.isUserInRole("ar_developers") + "<br>");
    try {
    //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 developer 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);
       }
    } catch (Exception e) { /* print stack trace */ }

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

See Also:

- "OracleAS JAAS Provider Realm and Policy APIs" on page 5-6
- "OracleAS JAAS Provider APIs for Granting or Revoking Permissions" on page 5-7
- "APIs for Checking Permissions" on page 5-10
- The next section, "OracleAS JAAS Provider Policy Configuration", regarding configuration resulting from permission grants

OracleAS JAAS Provider **Policy Configuration**

This section documents subject-based policy configuration that results from granting permissions when you use the OracleAS JAAS Provider Admintool or policy management APIs, as discussed in preceding sections. In OC4J, this configuration can be located in either the `system-jazn-data.xml` file or in Oracle Internet Directory, depending on the security provider.

The following topics are covered:

- Policy Repository Setting in jazn.xml
- Policy Configuration in system-jazn-data.xml
- Policy Configuration in Oracle Internet Directory

Note: Configuration discussed here is for subject-based security only. Code-based security is configured in a Java 2 policy file (`.policy`), as discussed in "Specifying a Java 2 Security Manager and Policy File" on page 5-1 and "Creating or Updating a Java 2 Policy File" on page 5-3.

Policy Repository Setting in jazn.xml

The policy repository for security policies for an OC4J instance is the provider specified in the `jazn.xml` file, as indicated by the `provider` setting in the `<jazn>` element, as follows:

- `provider="XML"` to use `system-jazn-data.xml` for policy configuration
- `provider="LDAP"` to use Oracle Internet Directory for policy configuration

By default, `provider` in `jazn.xml` is set to "`XML`" to use `system-jazn-data.xml` as the policy repository. When you use Oracle Identity Management and associate an Oracle Internet Directory instance with the OC4J instance, the provider setting in `jazn.xml` is changed to "`LDAP`", resulting in the use of Oracle Internet Directory as the policy repository.

(Similarly, the `provider` setting in `orion-application.xml` specifies the security provider of the application, which is "`XML`" for the file-based provider, "`LDAP`" for Oracle Identity Management, and by convention is also `"XML"` for an external LDAP provider, custom login module, or Oracle Access Manager.)

> **Note:** Policy repository configuration in `jazn.xml` (such as settings for the `provider` and `location` attributes of the `<jazn>` element) is OC4J instance-level configuration. If you deploy an application to the OC4J instance, and the application configures a different provider, the result would be a mixed usage where the provider configured in `orion-application.xml` would be the identity store used for authentication, while the provider specified in `jazn.xml` would be the policy store used for authorization.

Policy Configuration in system-jazn-data.xml

For the file-based provider, Oracle Access Manager, an external LDAP provider, or a custom login module, policy configuration is located in the `<jazn-policy>` element of the `system-jazn-data.xml` file.

As an example, we repeat one of the `grantperm` examples from "Granting Permissions through the OracleAS JAAS Provider Admintool" on page 5-12:

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

This results in `<jazn-policy>` configuration such as in the following example:

```
<jazn-data>
   ...
   <jazn-policy>
     <grant>
       <grantee>
         <principals>
           <principal>
             <realm-name>myrealm</realm-name>
             <type>role</type>
             <class>oracle.security.jazn.XMLRealmRole</class>
             <name>developers</name>
           </principal>
         </principals>
       </grantee>
       <permissions>
         <permission>
           <class>com.evermind.server.rmi.RMIPermission</class>
```

```
            <name>login</name>
          </permission>
        </permissions>
      </grant>
      ...
    </jazn-policy>
    ...
</jazn-data>
```

Policy Configuration in Oracle Internet Directory

For Oracle Identity Management (the LDAP-based provider), policy configuration is located in Oracle Internet Directory. As with `system-jazn-data.xml`, policy information stored in Oracle Internet Directory is accessible through the OracleAS JAAS Provider Admintool or policy management APIs.

Specification of the Oracle Policy Provider

If you use the Java virtual machine shipped with Oracle Application Server, the JAAS policy provider supplied with OracleAS JAAS Provider is automatically specified as the policy provider to use in OC4J. If you use another JVM (in other words, if you run an application outside OC4J), then the Oracle JAAS policy provider, `oracle.security.jazn.spi.PolicyProvider`, must be explicitly specified as the policy provider. (By default, a non-Oracle JVM uses the Sun Microsystems JAAS provider.)

You can specify Oracle-specific JAAS properties, such as the policy provider, in a security properties file that you supply to the JVM when you run OC4J. Oracle offers a default security properties file, *ORACLE_HOME*`/j2ee/home/config/jazn.security.props`, that specifies `oracle.security.jazn.spi.PolicyProvider` as the policy provider to use in OC4J, with the following configuration:

```
auth.policy.provider=oracle.security.jazn.spi.PolicyProvider
```

To append default Oracle-specific security property settings, including the above specification of the Oracle JAAS policy provider, to existing security properties, set the `java.security.properties` system property as follows:

```
-Djava.security.properties=ORACLE_HOME/j2ee/home/config/jazn.security.props
```

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

```
-Djava.security.properties==ORACLE_HOME/j2ee/home/config/jazn.security.props
```

Authorization Coding and Configuration

This section provides discussion and examples of the following steps to check authorization during runtime:

1. Using J2EE Authorization APIs
2. Obtaining a Subject
3. Using the checkPermission() Method
4. Configuring and Using JAAS Mode
5. Enabling the Java Authorization Contract for Containers (optional extension of J2EE authorization)

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

See Also:

- Appendix B, "OracleAS JAAS Provider Samples" for the complete example

Using J2EE Authorization APIs

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.isUserInRole('ar_developers') = " +
                request.isUserInRole("ar_developers") + "<br>");
    out.println("</BODY>");
    out.println("</HTML>");
}
```

See Also:

- "Related Web Application APIs" on page 2-4
- "Related EJB APIs" on page 2-6

Obtaining a Subject

To perform authorization in your code, you will often have to obtain a `Subject` instance for the authenticated user who is attempting to access resources. You can accomplish this through standard JAAS functionality, using the static `Subject.getSubject()` method and specifying an access control context.

Typically, use the default access control context:

```
mysubject = Subject.getSubject(AccessController.getContext());
```

Alternatively, you can specify a particular access control context (such as one you have constructed from a set of protection domains, for example):

```
mysubject = Subject.getSubject(acc);
```

Note that when you use JAAS mode, OC4J associates an authenticated subject and its permissions with the default access control context and its permissions.

See Also:

- "APIs for Checking Permissions" on page 5-10
- "Configuring and Using JAAS Mode" on page 5-18

Using the checkPermission() Method

You will typically use a `checkPermission(Permission)` call in your authorization code, which checks whether the access request indicated by the specified permission should be allowed, based on the security policy currently in effect, and throws an exception if not.

This method is available in the `AccessController` and `AccessControlContext` classes. Assuming you use the default implementations (provided with the Sun JDK), the `checkPermission()` method of either class can enforce Java 2 (code-based) policy in your application regardless of whether a security manager is enabled.

When you use a security manager, the method is also available in the `SecurityManager` class, but the default `SecurityManager` implementation of `checkPermission()` calls `AccessController.checkPermission()`.

You will typically use the `AccessController.checkPermission()` method, which is static. This call uses the default access control context (the context inherited when the thread was created). If you want the permissions check to be with respect to some other context, however, you can call the instance method `checkPermission()` on a particular `AccessControlContext` instance. The following example uses the `AccessController` method:

```
//create permission
FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
//check permission
AccessController.checkPermission(perm);
```

In OC4J, any JAAS mode setting is relevant with respect to what the access control context consists of when `AccessController.checkPermission()` is called, as follows:

- With no JAAS mode (`jaas-mode="null"`), `checkPermission()` enforces code-based security based on the security policy in effect, as presumably specified in a Java 2 policy file. There is no provision for subject-based security.
- With `doAs` JAAS mode (`jaas-mode="doas"`), `checkPermission()` enforces a combination of code-based and subject-based security according to the new access control context created through the `doAs()` block within which OC4J executes your application code. OC4J appends the permissions of the subject to the permissions of the default access control context.
- With `doAsPrivileged` JAAS mode (`jaas-mode="doasprivileged"`), `checkPermission()` has the same functionality as in `doAs` mode, but OC4J uses a null access control context, as specified when OC4J calls `doAsPrivileged()`. This is to use subject-based security only.

See Also:

- "Introduction to JAAS Mode" on page 5-5
- "APIs for Checking Permissions" on page 5-10
- "JAAS Authorization: Subject Methods doAs() and doAsPrivileged()" on page 2-15

Configuring and Using JAAS Mode

In this example, the `doGet()` method shown in "Using J2EE Authorization APIs" on page 5-17 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:

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

The code follows, using two different ways to check permissions, for demonstration purposes. 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.isUserInRole('ar_developers') = " +
               request.isUserInRole("ar_developers") + "<br>");

  //create Permission
  FilePermission perm = new FilePermission("/home/developer/foo.txt","read");
  {
     // CHECK PERMISSION VIA ACCESS CONTROLLER
     AccessController.checkPermission(perm);

     // CHECK PERMISSION VIA JAAS POLICY
     //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.implies permission: "+ perm +" ? " +
     currPolicy.getPermissions(Subject.getSubject(acc),null).implies(perm));
  }
  out.println("</BODY>");
  out.println("</HTML>");
}
```

See Also:

- "APIs for Checking Permissions" on page 5-10

Enabling the Java Authorization Contract for Containers

This section describes how to enable the Oracle JACC provider in OC4J. With JACC, J2EE security constraints are translated into Java 2 permissions to effectively provide an extended version of J2EE authorization, using the same configuration and code for authorization in your application.

The following topics are covered:

- System Property to Enable JACC Features
- System Properties to Specify the JACC Provider

Note: JACC 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 5-11 for an overview

System Property to Enable JACC Features

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

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

System Properties to Specify the JACC Provider

To employ a JACC provider, the system properties described in Table 5–2 must be set appropriately at application server startup. For the Oracle JACC provider, this happens automatically when you enable JACC, with the properties being set as shown in parentheses.

Table 5–2 System Properties for the JACC 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)

Authorization Strategies

For each of the key Java security models discussed earlier—J2EE, Java 2, and JAAS—this section summarizes when it may be advantageous to use it, and how it works. Complete operational details are provided elsewhere in this manual (mostly earlier in this chapter).

See Also:

- "Security Considerations during Development" on page 2-17

Considering J2EE Security

J2EE (static role-based) security is a coarse-grained model that specifies what security roles can access a Web application or EJB.

When Should You Use It?

In a J2EE application, this is the simplest and most basic form of security. Almost any J2EE application will use it, and it will often be enough to suit your needs. It is standard and therefore platform-independent.

Because of its limitations, however, it is sometimes used in conjunction with the other security models. Note that with J2EE security alone, you cannot control access to particular resources or define particular permissions for a role. You also cannot control

access by particular code entities. In addition, the J2EE security model is a static model; policies cannot be changed at runtime.

How Do You Set It Up?

Set up J2EE security through standard specifications for security roles, role-linking, and security constraints through the `web.xml` and `ejb-jar.xml` files, in addition to specifications for role-mapping through the OC4J-specific descriptors such as `orion-application.xml`.

If you use only J2EE security, JAAS mode is unnecessary (`jaas-mode="null"`).

How Is It Enforced?

J2EE security is enforced by the J2EE container (OC4J).

Considering Java 2 Security

Java 2 (code-based) security controls access to resources based on the location of executing code or on code signers.

When Should You Use It?

Use Java 2 security when your application must check code-based permissions, perhaps regardless of the user or role trying to access it. With Java 2 security, an administrator can enable or disable a security manager to control when the security manager performs security checks.

Generally, code-based security and the use of a security manager is required only if a situation may arise where the application server is exposed to untrusted code. Also be aware that there may be a performance impact with the use of a security manager.

Java 2 security may be used in conjunction with either J2EE security, JAAS security, or both.

How Do You Set It Up?

Specify Java 2 policies through a standard `.policy` file, typically named `java.policy` or `java2.policy`. The policy file *`ORACLE_HOME`*`/j2ee/home/config/java2.policy` is supplied with OC4J and includes permissions that are required for OC4J to run with a security manager.

Oracle provides no tools to maintain Java 2 policy files; you must do so manually or use tools provided by the JDK vendor or a third-party vendor.

If you use Java 2 security alone or with J2EE security only, JAAS mode is unnecessary (`jaas-mode="null"`).

How Is It Enforced?

For Java 2 security policies to be enforced by an application server and the underlying JDK, a security manager must be enabled.

For Java 2 security policies to be enforced within your application:

- If you want the capability of code-based security in your application, but with an administrator being able to control when the security checks are performed, you can choose to enforce code-based security only when a security manager is enabled. In this case, you can use the `checkPermission()` method of the `SecurityManager` instance.

  Be aware that for this scenario, the JVM must be started with a security manager enabled.

- If code-based security requirements in your application are independent of a security manager, you can choose to enforce code-based security within your application regardless of the presence of a security manager. In this case, you can use the static `AccessController.checkPermission()` method to check permissions.

  In this scenario, it is important to note that the overall environment will not be secure without a security manager. Other code in the environment, including JDK classes, will not be running in a secure mode.

- To specify the particular protection domains to be checked, you can construct an `AccessControlContext` instance from an array of `ProtectionDomain` instances, and call the `checkPermission()` method on the `AccessControlContext` instance. (This is not a common usage scenario, however.)

Considering JAAS Security

JAAS (subject-based) security is a relatively fine-grained model that controls access to resources according to the particular permissions of the authenticated subject.

When Should You Use It?

Usually, the coarser-grained security of the J2EE model will suffice. JAAS security is more complicated to administer and deploy; however, JAAS security is valuable if you want finer control over resources, such as:

- According to factors other than whether the resource is accessed as part of the execution of a Web module or EJB
- According to individual permissions that can be granted or revoked either by an administrator beforehand or in your code at runtime

J2EE security, by contrast, is more static—access control cannot be modified at runtime.

JAAS security is typically used together with J2EE security, and may also be used together with Java 2 security.

How Do You Set It Up?

You can grant (or revoke) permissions either beforehand, using the OracleAS JAAS Provider Admintool (as shown in "Granting Permissions through the OracleAS JAAS Provider Admintool" on page 5-12) or at runtime through OracleAS JAAS Provider APIs (as shown in "Using OracleAS JAAS Provider Policy Management APIs" on page 5-13). The resulting JAAS (subject-based) policy is reflected in `system-jazn-data.xml`, or Oracle Internet Directory if you use Oracle Identity Management as the security provider.

How you set the JAAS mode is also relevant. To use JAAS authorization without Java 2 (code-based) authorization, use the `doAsPrivileged` JAAS mode by setting `jaas-mode="doasprivileged"`, and check permissions by using the `Policy.implies()` method.

To use JAAS authorization together with Java 2 authorization:

- Use the `doAs` JAAS mode by setting `jaas-mode="doas"`.
- Specify Java 2 policies through a standard `.policy` file, typically the `java2.policy` file supplied by Oracle.

Note: If a resource being accessed in `doAsPrivileged` mode is a resource defined by the JVM vendor, additional security checks may be performed if a security manager is enabled.

How Is It Enforced?

You can enforce JAAS security through the `AccessController.checkPermission()` method or `Policy.implies()` method.

If you are also using Java 2 security and have a security manager enabled, you can alternatively use the `SecurityManager.checkPermission()` method.

(Also refer to earlier discussion regarding enforcement of Java 2 policy.)

6

General Tasks for OC4J Security

This chapter discusses general tasks and related guidelines that apply across the various security providers you can use with OC4J:

- Tasks for Password Management
- Using Security Realms in OC4J
- Deployment Tasks for Security
- Post-Deployment Tasks for Security
- Tasks to Share a Library

See Also:

- *Oracle Application Server Best Practices* (available post-release) for information about best practices for security
- The following Web site for OC4J "how-to" examples:

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

Tasks 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:

- Using Password Indirection
- Specifying a Password Manager in system-application.xml
- Password Obfuscation in OC4J Configuration Files

Using Password Indirection

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

- `data-sources.xml`: `password` attribute of `<native-data-source>` and `<managed-data-source>` elements
- `ra.xml`: `<res-password>` element
- `rmi.xml`: `keystore-password` attribute of `<ssl-config>` element
- `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="mypass">`

  The literal string "`mypass`" is the password; the password is not indirect.

> **Note:** If you choose to use indirect passwords in the current OC4J implementation, an indirect user is created in the `system-jazn-data.xml` file. If an indirect user account is created, either automatically or manually, undeploying an application does not automatically remove the account; you must manually remove the account.

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, "Using Password Indirection"). 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 the `system-application.xml` file can be different from the `<jazn>` element at the top level of that file.

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 />
</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 "Using Password Indirection" on page 6-1.

Note: In `system-jazn-data.xml` or an application-specific `jazn-data.xml` file, you can specify clear (human-readable) passwords in one of two ways, although this is discouraged:

- Set the `clear` attribute of the `<credentials>` element to `"true"`:

  ```
  <user>
     <name>myname</name>
     <credentials clear="true">welcome</credentials>
     ...
  </user>
  ```

- Precede the password with "!" (in which case "!" is not considered part of the password):

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

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 in the OracleAS JAAS Provider" on page 3-3. 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
- Evaluation of Default Realm for File-Based Provider, 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 `jazn.xml` file of the OC4J `home` instance.

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 6-5 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 Default Realm for File-Based Provider, 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 user name with a realm 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 J2EE Connector Architecture.

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 (in other words, a custom realm)—such as `acme.com`, for example, in 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 J2EE Connector Architecture.

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`".

Also be aware that when you use a custom realm, and JAAS policies are granted to users or roles in the custom realm, you should do the following:

1. In the `<jazn>` element of your application `orion-application.xml` file, specify a `default-realm` setting of "*`custom_realm_name`*".
2. Do *not* specify a `location` attribute setting in the `<jazn>` element.
3. Set the `jaas.username.simple` property to "`false`" in `jazn.xml`, using a `<property>` subelement of the `<jazn>` element, as discussed in "Omitting the Realm Name When Retrieving an Authenticated Principal" on page 6-6.

These steps allow the custom realm and its users, roles, and policies to be persisted.

Note that to use JAAS authorization, in particular to grant permissions to users or roles in a custom realm, the custom realm and its users and groups must be defined and persisted in `system-jazn-data.xml`, not in an application-specific `jazn-data.xml` file.

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 J2EE Connector Architecture. 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:

- If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any prior settings within the `<jazn>` element would be lost and would have to be redone.

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

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

Deployment Tasks 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 Security 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 OC4J 10.1.3.x implementations, the Oracle Access Manager.
- 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 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 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

General information about deploying an application to OC4J, including information about deployment plans and using Application Server Control Console, is 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. (In an Oracle Application Server environment, from the Cluster Topology page, choose the desired OC4J instance to get to its home page.)
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 (such as for loading shared libraries)

   For security, selecting a security provider and mapping security roles are of particular interest. You may also want to configure class loading for 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 6-9
- "Specifying Security Role Mapping through Application Server Control" on page 6-11
- "Providing Login Modules as OC4J Shared Libraries" on page 9-15

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 during development and for deployed applications with a small user population, such as in a standalone OC4J environment. Use Oracle Identity Management in larger production environments.

The file-based provider is a lighter-weight implementation, while 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. It also simplifies access to the user repository from multiple OC4J instances; with the file-based provider, user data updates have to be coordinated between instances, given that each instance has its own repository.

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 an Oracle Internet Directory instance 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 6-8 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 necessitates its own set of configuration tasks, documented in the following locations:
   - "Configuring the File-Based Provider during Application Deployment" on page 7-3
   - "Specifying Oracle Identity Management during Deployment" on page 8-13
   - "Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15
   - "Specifying and Configuring an External LDAP Provider during Deployment" on page 10-3

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 6-8.

> **Note:** The Oracle Access Manager security provider is not supported through Application Server Control (although its login module can be configured through Application Server Control like any other login module). See Chapter 11, "Oracle Access Manager" for information about how to configure Oracle Access Manager.

Mapping Security Roles

This section discusses various aspects of the following:

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

The information is organized as follows:

- Application Role Definitions and References
- Specifying Security Role Mapping through Application Server Control
- Mapping J2EE Roles to Deployment Roles in OC4J Configuration Files
- Using the OC4J PUBLIC Role to Allow General Access by Authenticated Users

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

See Also:

- "Overview of Security Role Mapping" on page 3-8
- "Web Application Security Role and Constraint Configuration" on page 17-6
- "Mapping J2EE Roles to Deployment Users and Roles" on page 18-6

Application Role Definitions and References

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

1. In your standard deployment descriptor (`web.xml` or `ejb-jar.xml`), use `<security-role>` elements to define J2EE logical roles, such as in the following example:

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

2. Use `<security-role-ref>` elements in the standard deployment descriptor to link from the roles you have developed in your application to the J2EE roles you have defined in the standard descriptors, such as in the following example, where `ar_developers` is an application role:

```
<security-role-ref>
   <role-name>ar_developers</role-name>
   <role-link>sr_developers</role-link>
</security-role-ref>
```

After these steps, mappings to deployment roles defined in the security provider (such as in a `jazn-data.xml` file or the `system-jazn-data.xml` file for the file-based security provider, or in Oracle Internet Directory for the LDAP-based provider) are defined in the OC4J-specific 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, 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 Deployment Roles in OC4J Configuration Files".

> **See Also:** The preceding discussion leaves out some details that differ between Web applications and EJBs. Refer to the following for additional information:
>
> - "Web Application Security Role and Constraint Configuration" on page 17-6
> - "Authenticating and Authorizing EJB Applications" on page 18-1

Specifying Security Role Mapping through Application Server Control

In Application Server Control Console, map J2EE roles to deployment roles during the deployment process, from the Deploy: Deployment Settings page (see "Deploying an Application through Application Server Control" on page 6-8 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 J2EE role you want to map. (You can also choose **Clear All Mappings**.)
3. In the Deployment Settings: Map Security Role page for the role, you can do any of the following:
   - Map all users and groups (deployment roles) to the J2EE role.
   - Map selected users to the J2EE 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, use the Add User feature in the Deployment Settings: Map Security Role page.
   - Map selected groups to the J2EE 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, use the Add Group feature in the Deployment Settings: Map Security Role page.
   - 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 6-8.

These actions create <security-role-mapping> elements in the applicable OC4J configuration file, such as orion-application.xml, orion-web.xml, or orion-ejb-jar.xml, as shown in the next section, "Mapping J2EE Roles to Deployment 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 Deployment Roles" on page 17-9 and "Mapping J2EE Roles to Deployment Users and Roles" on page 18-6) and then restart or redeploy the application.

Mapping J2EE Roles to Deployment Roles in OC4J Configuration Files

Portable J2EE logical roles defined in a standard deployment descriptor are mapped to deployment 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 J2EE role sr_developers is mapped to the deployment role developers. Note that a <group> subelement under a <security-role-mapping> element corresponds to a deployment role, such as one defined in system-jazn-data.xml or Oracle Internet Directory. You can also have <user> subelements for mapping individual users.

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

This association permits the developer role to access resources that are accessible to the sr_developers role according to security constraints configured in the standard deployment descriptors.

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

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 to Share a Library

The OracleAS JAAS Provider is integrated with the OC4J class loading architecture. You can make libraries available to applications by loading them as OC4J shared libraries. This is useful to share the following among applications, for example:

- Login modules
- Principal classes for authorization and subject propagation
- Identity management framework implementation classes

There are two main steps to sharing and using a library (considering functionality of the Application Server Control Console in particular):

1. Loading the Library as an OC4J Shared Library
2. Importing the Library into Your 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

Loading the Library as an OC4J Shared Library

To load a library as an OC4J shared library through Application Server Control, use the Shared Libraries task.

1. Go to the **Administration** tab for the OC4J instance.
2. Choose the Shared Libraries Task (under Properties).
3. From the Shared Libraries page, choose **Create**.
4. From the Create Shared Library: Attributes page, specify the desired library name and version, then proceed to the next page.
5. From the Create Shared Library: Add Archives page, choose **Add** to add a library.
6. From the Add Archives: Add Archive page, you can choose to upload a library from the local host, upload a library from the server where Application Server Control is located, or specify a location on the target server where the library already exists. Repeat this process for each library you want to add, then continue.
7. Again from the Create Shared Library: Add Archives page, you can either finish, or go to the next page, Create Shared Library: Import Shared Libraries page. There you can import additional libraries into your library, then finish. This takes you back to the Shared Libraries page.

Loading a library 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>
```

Importing the Library into Your Application

You can import a library into your application through Application Server Control in the process of deploying your application:

1. When you reach the Deploy: Deployment Settings page (as discussed in "Deploying an Application through Application Server Control" on page 6-8), you can choose the Configure Class Loading task to import shared libraries.
2. From the Deployment Settings: Configure Class Loading page, choose the libraries to import, then choose **OK**.
3. Proceed with the deployment.

Importing a library 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>
```

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. Use of this file-based provider is typical during development and in small production environments (such as when using standalone OC4J), and 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.

See Also:

- "Creating a New Administrator Account" on page 4-13 if you want to use an administrator account other than `oc4jadmin`

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-4.

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 application 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 Roles through Application Server Control
- Administering Instance-Level Security through Application Server Control

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).
- 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.

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 6-8 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:
   - Choose whether to use the default file-based provider of the OC4J instance, or an application-specific file-based provider.
   - Specify the location of your repository, optionally an application-specific `jazn-data.xml` file, for user and role configuration. By default, the `system-jazn-data.xml` file will be used.
   - 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 6-8.

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 6-13.

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"):
   - Choose whether to use the default file-based provider of the OC4J instance, or an application-specific file-based 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.
   - 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. You are prompted to restart your application for the change to take effect.

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 6-13.

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 Roles through Application Server Control" on page 7-7.

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 administrative user of the realm.
- Specify and confirm the desired password for the administrative user.
- Specify the desired administrator role of the realm. The administrative 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 6-13.

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 by navigating 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 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 6-13.

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 from 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

You can specifically configure realms, users, and roles for the OC4J instance-level file-based security provider. Changes made in this way will always affect the `system-jazn-data.xml` file, rather than any application-level `jazn-data.xml` file (if any have been specified).

(The instance-level file-based provider is specified to be `system-jazn-data.xml` according to settings in the `<jazn>` element of the OC4J `system-application.xml` file.)

You can administer the instance-level file-based provider 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 Roles through Application Server Control" on page 7-7.

> **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 (discussed earlier in this chapter) for configuration and administration, instead of manipulating the files directly. Using this tool results in the appropriate entries automatically being made in the configuration files.

The rest of this discussion covers the following:

- Settings in the <jazn> Element for the File-Based Provider
- Realm Configuration in the Repository File
- Policy Configuration in the Repository File
- Predefined OC4J Accounts in system-jazn-data.xml

Settings in the <jazn> Element for the File-Based Provider

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.

This section discusses the following related topics:

- Scenarios for <jazn> Settings in orion-application.xml
- Configuration to Automatically Create an Application-Specific jazn-data.xml File
- Supplying an Application-Specific jazn-data.xml File

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

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.

Configuration to Automatically Create an Application-Specific jazn-data.xml File

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" />
```

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" />
   ```

   If you specify a relative location, the location is relative to that of the `orion-application.xml` file within which the `<jazn>` element is contained, typically the `/META-INF` directory of the application EAR file.

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

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-14.

Policy data is stored in the file `system-jazn-data.xml`. In the following example, a segment of this file shows the result of granting RMI permission "login" to the `admin` principal. (This example assumes `admin` is a user in the `jazn.com` realm.)

```
<jazn-policy>
  <grant>
   <grantee>
     <principals>
      <principal>
        <realm-name>jazn.com</realm-name>
        <type>user</type>
        <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 (with member `oc4jadmin`)
- `oc4j-app-administrators` role
- `ascontrol_admin` role (with member `oc4jadmin`)
- `ascontrol_appadmin` role
- `ascontrol_monitor` role
- `anonymous` user, initially deactivated
- `users` role
- `jtaadmin` user

See Also:

- "Predefined Accounts" on page 4-11 for additional information about these accounts
- "Activation of the oc4jadmin Account (Standalone OC4J)" on page 4-12

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`.

This section covers the following topics:

- Overview of the Migration Tool
- Migration Tool Command Syntax
- Migration Tool APIs

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: Be aware of the following when you use the migration tool:

- To migrate to Oracle Internet Directory, the directory server must be running and available when you run the tool.
- 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).
- When in policy mode or all mode: 1) When migrating permissions for a non-realm custom principal, the JAR file containing the class files for the custom principal must be available in the classpath. 2) 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)	

Table 7–1 (Cont.) OracleAS JAAS Provider Migration Tool Options

Option	Description	Default (where applicable)
`-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`
`-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`

Table 7–2 (Cont.) JAZNMigrationTool Constants

Key Constant	Corresponds to Option
DEST_TYPE	-dt
OID_USER	-D
OID_PASSWORD	-w
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 deployment users and converts `principals.xml` groups to deployment roles. All permissions that were previously granted to a `principals.xml` group are mapped to the deployment 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. (It will not be supported in the 11*g* 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 "Settings in the <jazn> Element for the File-Based Provider" on page 7-9.
5. Stop and restart OC4J.

Using the File-Based Provider Across an OC4J Group

The OC4J 10.1.3.1 implementation adds features to include OC4J instances in a "group" (where previously, instances could only be in a group if they had the same instance name).

Through use of these features, and use of OC4J `J2EEServerGroup` MBean, you can coordinate changes to the `system-jazn-data.xml` file in each OC4J instance in a group.

OC4J Basic Group Features

In an OC4J cluster, you can create a new group through Application Server Control as follows:

1. In the Cluster Topology page, under Groups, choose **Create**.
2. In the Create Group page:
   a. Specify the desired name of the group.
   b. Select the OC4J instances to move to the group. Be aware that an OC4J instance you move into the new group is removed from the group it was in previously (if applicable), and that an instance must be stopped before it can be moved.

Important: When you stop an OC4J instance, at least one other OC4J instance must be running on the application server that hosts the instances. If the check box to stop an OC4J instance is disabled, then no other OC4J instances in that application server are running.

---

   c. Choose **Create**.

Note: You can also move an OC4J instance into a group after the group is created:

1. In the Cluster Topology page, under Groups, select the group.
2. In the Group: *groupname* page, in the OC4J Instances tab, choose **Add**.
3. In the Add OC4J Instances to Group page, select and add OC4J instances to the group as desired.

You can administer a group through Application Server Control as follows:

1. In the Cluster Topology page, under Groups, choose the desired group.
2. Choose the **Administration** tab.
3. The Administration page provides administration features for the group as a whole. Note, however, that this does *not* include Security Provider administration.

See Also:

- For additional information about OC4J group features, the topic "Group OC4J Instances Page" in the Application Server Control online help

Cluster MBean Browser Features and the J2EEServerGroup MBean

Once you have created a group and populated it with OC4J instances, you can use the Cluster MBean Browser to coordinate settings to each `system-jazn-data.xml` file across the group, by invoking operations on the `J2EEServerGroup` MBean for the group. This also involves the `J2EEApplication` MBean for the `system` application.

You can accomplish this as follows:

1. In the Cluster Topology page, under Groups, chose the group.
2. In the Group: *groupname* page, choose the **Administration** tab.
3. In the Administration page, under JMX, go the Cluster MBean Browser task.
4. In the Cluster MBean Browser page:
   a. Under the `J2EEServerGroup` MBean, select the OC4J group.
   b. Choose the **Operations** tab.
   c. Choose the `invoke` operation.
5. In the Operation: invoke page (shown in Figure 7–1 at the end of this procedure), use the flashlight icon to search for MBeans by name.
6. In the Search and Select: MBean page (shown in Figure 7–2 at the end of this procedure):
   a. Search by MBean name "SecurityProvider".
   b. In the search results, select the result with `J2EEApplication=system` (the last result shown in the figure).
7. Back in the Operation: invoke page:
   a. For the parameter name `operationName`, choose the desired operation from the dropdown menu. Available operations for the `J2EEApplication` MBean include (among many others): `addUser`, `addRole`, `remUser`, `remRole`, `revokeUserRole`, `revokeUserPerm`, `grantUserPerm`, and `grantRolePerm`, for example.
   b. Use the pencil icon to specify parameter settings for the operation. For example, for `addUser`, the parameters are `username`, `passwd`, and `realm`.
   c. In the Edit Params page, specify the desired settings.
8. Back in the Operation: invoke page, choose **Invoke Operation**.

Results of the operation will apply to `system-jazn-data.xml` files across all instances of the group.

Figure 7–1 Operation: Invoke

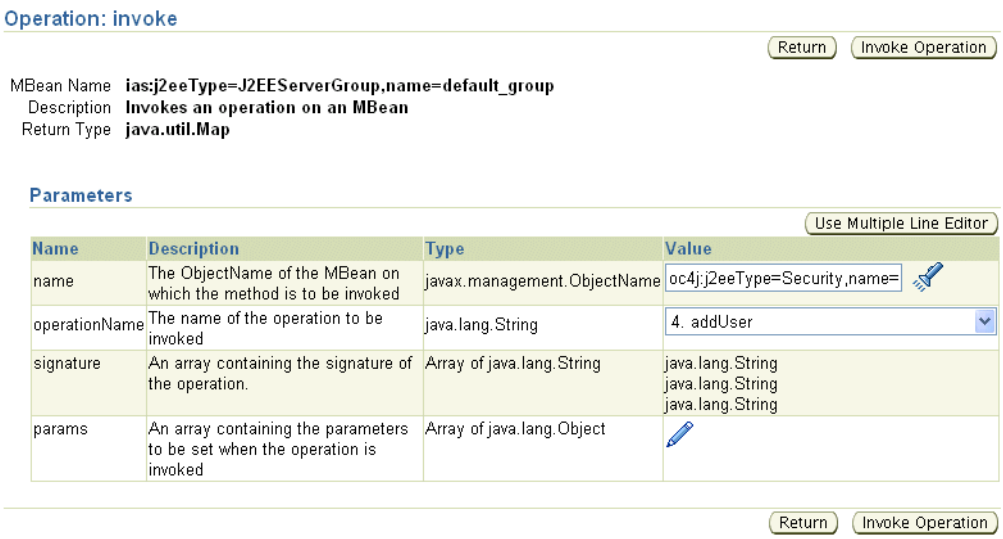
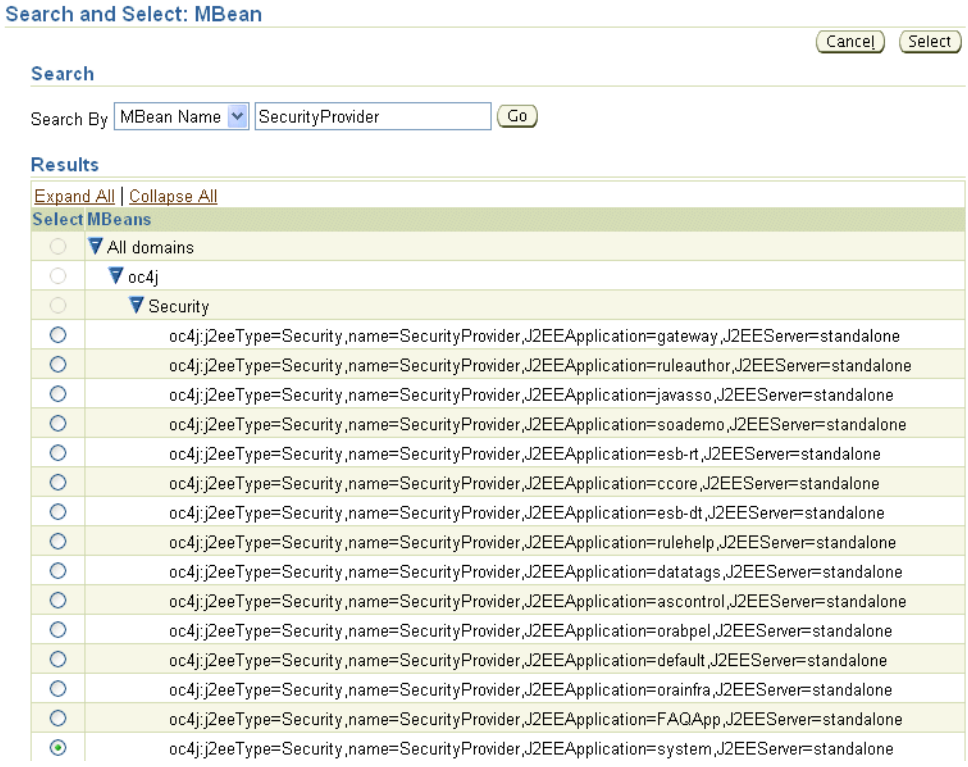


Figure 7–2 Search and Select: MBean



8

Oracle Identity Management

In Oracle Application Server, Oracle Identity Management with Oracle Internet Directory and (optionally) Oracle 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. The following topics are covered:

Initial Considerations for OC4J Support of Oracle Identity Management

Be aware of the following notes regarding the use of Oracle Identity Management with OC4J:

- Beginning with OC4J 10.1.3.x implementations, the LDAP-based provider is supported in standalone OC4J as well as in an Oracle Application Server environment.
- The OracleAS JAAS Provider supports the identity management realm type in Oracle Internet Directory. (This realm type is discussed in "Overview of OracleAS JAAS Provider Realms for Oracle Identity Management" on page 8-16.) The external and application realm types are deprecated. Specifically, 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.
- Managing users and roles in Oracle Internet Directory is beyond the scope of this document. Consult the *Oracle Identity Management Guide to Delegated Administration*.
- OC4J provides a login module, `LDAPLoginModule`, for use with LDAP servers. For Oracle Internet Directory, however, use the default `RealmLoginModule`.

(Configuring `LDAPLoginModule` for use with Oracle Internet Directory would result in the loss of optimizations and integrations that are otherwise available.)

Notes:

- Be aware that with the LDAP-based provider, role comparisons for authorization are *not* case-sensitive.
- 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 8-5.

See Also:

- For information about migrating from a file-based repository to an Oracle Internet Directory repository, "OracleAS JAAS Provider Migration Tool" on page 7-13
- For information about user and role APIs that you can use with Oracle Internet Directory, Chapter 12, "User and Role API Framework"
- "Creating a New Administrator Account" on page 4-13 if you want to use an administrator account other than `oc4jadmin`.

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 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 Oracle Single Sign-On. This section provides an overview of these features, covering the following topics:

- Overview of Oracle Internet Directory
- About Distinguished Names
- Overview of Oracle Single Sign-On
- SSO-Enabled J2EE Environment: Typical Scenario

See Also:

- *Oracle Identity Management Infrastructure Administrator's Guide*
- *Oracle Identity Management Application Developer'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, thereby 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 OC4J 10.1.3.x implementations, 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*

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.

Overview of Oracle Single Sign-On

Oracle 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.

There is also support for global logout and session timeout.

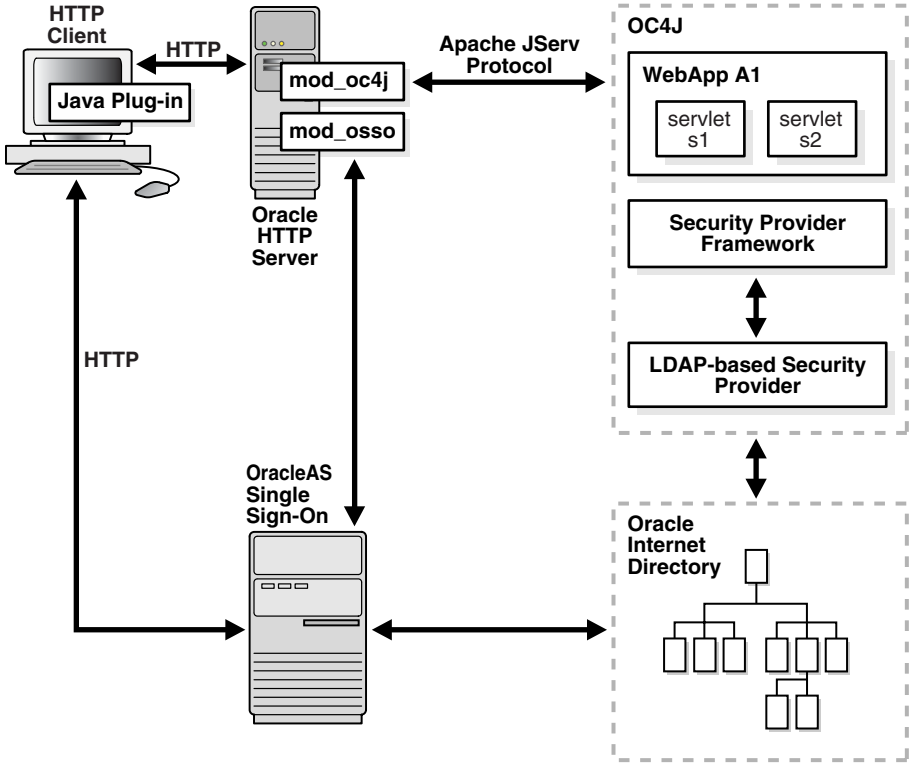
See Also:

- *Oracle Application Server Single Sign-On Administrator's Guide*
- *Oracle Identity Management Application Developer's Guide* (particularly the chapter on developing applications for single sign-on)

SSO-Enabled J2EE Environment: Typical Scenario

Oracle Single Sign-On lets a user access multiple applications with a single set of login credentials. Figure 8–1 shows JAAS integration in an application running in an SSO-enabled J2EE environment.

Figure 8–1 Oracle 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 Oracle 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 Oracle Single Sign-On for authenticating HTTP clients.
   - Redirects the HTTP client request to the Web-based Oracle 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 Oracle 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 Oracle Single Sign-On user.

See Also:

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

Prerequisite: Oracle Application Server Infrastructure

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

For using Oracle Identity Management (with Oracle Internet Directory) as the security provider under OracleAS JAAS Provider in the OC4J 10.1.3.1 implementation, the supported versions of the Oracle Application Server infrastructure are 10.1.2.0.1, 10.1.2.0.2, and 10.1.4.x.

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

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 Oracle Single Sign-On for authentication:

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

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 Accounts Created in Oracle Internet Directory
- Oracle Internet Directory Association in jazn.xml
- Considering Multiple OC4J Instances when Associating Oracle Internet Directory

Important:

- When you associate an OC4J instance with an Oracle Internet Directory instance, the `<jazn>` element configuration in the `jazn.xml` file of the OC4J `home` instance is rewritten and any previous settings are lost.
- Associating an Oracle Internet Directory instance with an OC4J instance results in provider settings at the instance level—such as the `provider` and `location` attribute settings in the `<jazn>` element of the `jazn.xml` file. If you deploy an application to the OC4J instance, and the application configures a different provider, the result would be a mixed usage where the provider configured in `orion-application.xml` would be the identity store used for authentication, while the provider specified in `jazn.xml` would be the policy store used for authorization.

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 (OID), 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 optionally specify Oracle Identity Management as the security provider for Application Server Control. (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 (actually, Oracle Identity Management), 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:

- Using SSL requires appropriate SSL configuration for OC4J and Oracle Internet Directory, as discussed in Chapter 15, "SSL Communication with OC4J", and the *Oracle Internet Directory Administrator's Guide*, respectively.
- 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 OracleAS JAAS Provider administrator account (for internal use) 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. You are prompted to restart OC4J for the change to take effect.

Required Accounts Created in Oracle Internet Directory

Oracle Internet Directory does not by default include certain accounts that are required by OC4J and Application Server Control 10.1.3.x implementations. Therefore, the accounts listed below are created automatically, 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. In fact, if any of these accounts are found to already exist in Oracle Internet Directory at the time of the OC4J-OID association process, the account creation step is skipped.

- `oc4jadmin` user
- `oc4j-administrators` role, with member `oc4jadmin`
- `oc4j-app-administrators` role
- `ascontrol_admin` (administrative role for all SOA controls, including Application Server Control), with member `oc4jadmin`
- `ascontrol_appadmin` (Application Server Control required role)
- `ascontrol_monitor` (Application Server Control required role)

Notes:

- In addition, the `JAZNAdminGroup` role with its OracleAS JAAS Provider administrator member are shipped with Oracle Internet Directory for internal use.
- 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 Accounts" on page 4-11 for additional information about the OC4J accounts
- "Activation of the oc4jadmin Account (Standalone OC4J)" on page 4-12

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 `jazn.xml` file of the OC4J `home` instance. 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:** At runtime, the LDAP-based provider connects as the OracleAS JAAS Provider administrator to Oracle Internet Directory. This user is a member of `JAZNAdminGroup`.

See Also:

- "Configuring LDAP User and SSL Properties" on page 8-21

Considering Multiple OC4J Instances when Associating Oracle Internet Directory

If you are using Oracle Internet Directory in an environment with multiple OC4J instances (such as the home and SOA instances in a SOA installation), then after you complete the steps for OC4J-OID association detailed earlier (in "Associating Oracle Internet Directory with OC4J" on page 8-6), you must manually copy the relevant `<jazn>` element configuration from the `jazn.xml` file of the `home` instance to the `jazn.xml` file of any other instance. This includes settings of the `provider` and `location` attributes, and any relevant property settings in `<property>` subelements.

Consider the following example, shown earlier:

```
<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>
```

All of this configuration would have to be copied. Be careful, however, to not overwrite any special settings in the `jazn.xml` file of the target instance.

Configure SSO (Optional)

This step is required only if you want to use Oracle 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

> **Important:** Do not confuse this SSO with Java SSO, which is a separate feature (documented in Chapter 14, "OC4J Java Single Sign-On"). You can use one SSO product or the other, but not both.

See Also:

- "OC4J Configuration for Oracle Single Sign-On Authentication" on page 8-15

Run the SSO Registration Tool

The first task in configuring Oracle 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 syntax for `ssoreg` options required for this usage, with options described in Table 8–1:

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

Table 8–1 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 Oracle 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 Oracle 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 an 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 Oracle 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. 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 infrastructure host where you run the command.

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 Oracle Single Sign-On (acting as the login, timeout, and logout service), OC4J 10.1.3.x implementations support 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 `sso.session.synchronize` property, which can be set under the `<jazn-web-app>` element either in the `orion-application.xml` file or in the `orion-web.xml` file that is used to configure a single Web application. In the event of competing settings, the `orion-web.xml` setting takes precedence for the particular Web application.

The following example enables the synchronization in the `orion-application.xml` file:

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

```
      </jazn-web-app>
      ...
   </jazn>
   ...
</orion-application>
```

If the `sso.session.synchronize` property is set to `true`, then the `ldap.cache.session.enable` property must also be set to `true`. For example:

```
<jazn-web-app auth-method="SSO">
   <property name="sso.session.synchronize" value="true" />
   <property name="ldap.cache.session.enable" value="true" />
</jazn-web-app>
```

The synchronization feature depends on the session to retrieve a previously stored authenticated user and assumes that the LDAP session cache is enabled. Setting the `ldap.cache.session.enable` property to `false` effectively disables using the session to store authenticated users. See "Configuring LDAP Caching Properties" on page 8-23 for more information on setting Oracle Internet Directory caching properties.

Note: If the `sso.session.syncronize` property is set to `true` and the `ldap.cache.session.enable` property is set to `false`, then the application eventually becomes unable to authenticate using Oracle Identity Management/Oracle Single Sign-On and the OC4J instance must be restarted.

Restart the Oracle HTTP Server and OC4J Instances

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

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 application access to EJBs over 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 completed the OC4J-OID association as discussed earlier, you can specify Oracle Identity Management (the LDAP-based provider) as the security 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 6-8 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 Oracle Single Sign-On authentication. This results in the configuration `auth-method="SSO"` in `orion-application.xml` for your application, as discussed in "OC4J Configuration for Oracle Single Sign-On Authentication" on page 8-15.

     Important: Do not confuse this with enabling Java SSO, which is a separate feature (documented in Chapter 14, "OC4J Java Single Sign-On") that has its own Application Server Control configuration page. You can use one SSO product or the other, but not both.
4. Choose **OK** to finish the security provider selection.
5. Confirm the JAAS mode setting, as appropriate:
   a. Back in the Deploy: Deployment Settings page, under "Advanced Deployment Plan Editing", choose **Edit Deployment Plan**.
   b. In the Deploy: Deployment Settings: Edit Deployment Plan page, in the Edit OC4J Descriptor tab, for the `jazn` descriptor, choose **Edit jazn**.
   c. In the Deploy: Deployment Settings: Edit Deployment Plan: jazn page, be sure the `jaasMode` attribute is set appropriately (such as to `doAsPrivileged` if your application requires that mode). Then choose **Continue**.
   d. Back in the Deploy: Deployment Settings: Edit Deployment Plan page, choose **OK**.

   Refer to "Introduction to JAAS Mode" on page 5-5 and "Configuring and Using JAAS Mode" on page 5-18 for more information on when and how to use this mode.
6. 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 6-8.

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 in the preceding section. You can also change to a different security provider after deployment. Assuming you have completed the OC4J-OID association discussed earlier, you can change to the Oracle Identity Management 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 6-13.
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 Oracle Single Sign-On authentication. This results in the configuration `auth-method="SSO"` for your application, as discussed in "OC4J Configuration for Oracle Single Sign-On Authentication" on page 8-15.

     Important: Do not confuse this with enabling Java SSO, which is a separate feature (documented in Chapter 14, "OC4J Java Single Sign-On") that has its own Application Server Control configuration page. You can use one SSO product or the other, but not both.
5. Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you are prompted to restart the application for the change to take effect.

Settings for Authentication Method with Oracle Identity Management

This section discusses Oracle-specific settings for certain authentication methods you can use when Oracle Identity Management is the security provider for a Web application:

- OC4J Configuration for Oracle Single Sign-On Authentication

- Using Digest Authentication with Oracle Internet Directory

OC4J Configuration for Oracle Single Sign-On Authentication

For use of Oracle Single Sign-On for authentication, the `auth-method` attribute is set 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>
```

Important: If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any prior settings within the `<jazn>` element would be lost and would have to be redone.

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

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 Oracle 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 jazn.xml file of the OC4J home instance, 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 4-4
- "Configuring LDAP User and SSL Properties" on page 8-21

Realm Management for the LDAP-Based Provider

This section discusses realm management for the LDAP-based provider, Oracle Identity Management. The following topics are covered:

- Overview of OracleAS JAAS Provider Realms for Oracle Identity Management
- Realm Management for Oracle Identity Management

Overview of OracleAS JAAS Provider Realms for Oracle Identity Management

The OracleAS JAAS Provider supports the *identity management* realm type in Oracle Internet Directory.

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

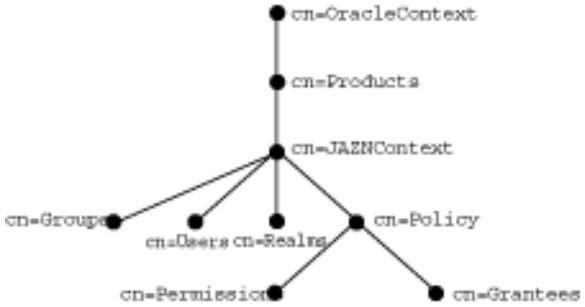
This section covers the following topics:

- Realm Hierarchy for the OracleAS JAAS Provider
- Relation of JAAS Provider Realms to Oracle Internet Directory Realms
- Access Control Lists and OracleAS JAAS Provider Directory Entries

Realm Hierarchy for the OracleAS JAAS Provider

As Figure 8–2 illustrates, the OracleAS JAAS Provider stores directory entries within the product container cn=JAZNContext. Beneath cn=JAZNContext is the 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.

Figure 8–2 Global JAZNContext Subtree



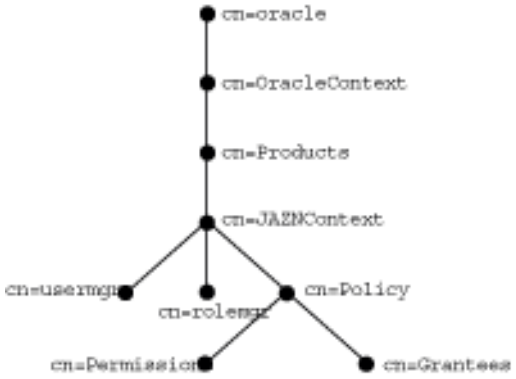
Note that the OracleAS JAAS Provider has its own `Groups` and `Users` containers. The `Groups` container contains the role `JAZNAdminGroup`. The `Users` container contains the OracleAS JAAS Provider administrative user that populates this role. Both the role and its member user are for internal use only, and note that the administrative user `JAZNAdminUser` is deprecated. An administrative user is created for each Oracle Application Server middle tier associated with the Oracle Internet Directory. A typical DN for the administrative user is something like the following:

```
orclapplicationcommonname=jaznadmin1,cn=jazncontext,cn=products,cn=oraclecontext
```

Using the identity management realm DN, the OracleAS JAAS Provider locates the realm-specific Oracle context and creates a corresponding `cn=JAZNContext` subtree.

In Figure 8–3, `cn=oracle` is an identity management realm. The OracleAS JAAS Provider stores the `cn=usermgr` entry, `cn=rolemgr` entry, and policy-related entries under the `JAZNContext` entry corresponding to the identity management realm.

Figure 8–3 Identity Management Realm JAZNContext Subtree



Relation of JAAS Provider Realms to Oracle Internet Directory Realms

For each identity management realm that you use, a corresponding OracleAS JAAS Provider realm is created. This is the mechanism through which identity management realms in Oracle Internet Directory are visible to the OracleAS JAAS Provider.

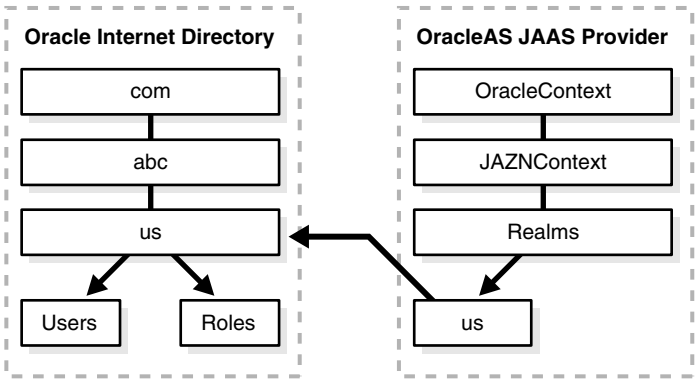
In OracleAS JAAS Provider, as shown earlier, a `Realms` container object exists under the site-wide JAAS context. For each Oracle Internet Directory realm instance, a corresponding realm entry is created under the `Realms` container to store the realm attributes. The directory hierarchy is known to the OracleAS JAAS Provider, which

enables it to create new realm entries in the appropriate directory location and find all the registered realms at runtime.

Oracle Internet Directory has a default identity management realm, depending on the system domain. In United States locations, for example, the default realm has a distinguished name such as "`dc=us,dc=abc,dc=com`" for company `abc`. OracleAS JAAS Provider creates a corresponding realm called "`us`", under `cn=Realms,cn=JAZNContext,cn=OracleContext` in the directory information tree. This is shown in Figure 8–4 below.

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

Figure 8–4 Simplified Directory Information Tree for the Identity Management Realm



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 full read and write privileges for OracleAS JAAS Provider directory objects to the role `JAZNAdminGroup` and its member OracleAS JAAS Provider administrative superuser (both for internal use only). 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.

Realm Management for Oracle Identity Management

This section discusses realm management when you use the LDAP-based provider (Oracle Identity Management), requiring administration tools of the Oracle Internet Directory. The following topics are covered:

- Managing Realms in Oracle Internet Directory
- Changing Your Default Realm
- Using Multiple Realms and Oracle Single Sign-On with OC4J

Managing Realms in Oracle Internet Directory

Manage users and roles in an Oracle Internet Directory identity management realm by using administrative features of the Oracle Delegated Administration Services (DAS), as detailed in the *Oracle Identity Management Guide to Delegated Administration*.

In addition, perform any further configuration of an Oracle Internet Directory realm through DAS as well. For example, this includes configuration of the user search base, group search base, user creation base, group creation base, and user nickname attribute.

OracleAS JAAS Provider itself does not perform any management of Oracle Internet Directory realms; it merely looks up existing information in Oracle Internet Directory as necessary.

Changing Your Default Realm

Oracle Internet Directory is shipped with a default realm, but you can optionally use a different realm as your default realm, using the following basic steps:

1. Create a new identity management realm in Oracle Internet Directory to use as your default realm. This is accomplished through DAS, as detailed in the *Oracle Identity Management Guide to Delegated Administration*. When you do this, the corresponding OracleAS JAAS Provider realm is provisioned automatically.
2. Set the OracleAS JAAS Provider `default-realm` attribute to the desired realm, as described in "Default Realm with the File-Based Provider or Oracle Identity Management" on page 6-4.

Important: Do not use the OracleAS JAAS Provider Admintool to create realms for Oracle Internet Directory. Realms created with this tool are suitable for the file-based provider only, and would not include sufficient information for use with Oracle Internet Directory.

Using Multiple Realms and Oracle Single Sign-On with OC4J

Creating additional identity management realms in Oracle Internet Directory is accomplished much as noted in the preceding section, "Changing Your Default Realm". Use DAS, as detailed in the *Oracle Identity Management Guide to Delegated Administration*. The corresponding OracleAS JAAS Provider realm is provisioned automatically.

Important:

- Multiple Oracle Internet Directory realms are supported by OracleAS JAAS Provider only in conjunction with the use of Oracle Single Sign-On.
- Do not use the OracleAS JAAS Provider Admintool to create realms for Oracle Internet Directory. Realms created with this tool are suitable for the file-based provider only, and would not include sufficient information for use with Oracle Internet Directory.
- When you add a realm, you may need to make existing applications aware of it. The procedure for doing this is specific to each application; refer to the application documentation.

See Also: For important additional information about using multiple realms with the OracleAS JAAS Provider:

- "Using a Nondefault Realm" on page 6-5
- "Using Multiple Realms" on page 6-6

Additional steps are required for using Oracle Single Sign-On in a multi-realm environment, to make the Oracle Single Sign-On server aware of the realms. In all, using multiple realms with Oracle Single Sign-On consists of the following steps, documented elsewhere (as referenced):

1. Create the realms, as summarized immediately above.
2. Configure the single sign-on server for multiple realms. This consists of the following steps, covered in detail in the *Oracle Application Server Single Sign-On Administrator's Guide*.
   a. Enable hosting on the single sign-on server. This is accomplished through a script called `enblhstg.csh`.
   b. Create an entry for each realm in the Oracle Single Sign-On database. This is accomplished through a script called `addsub.csh`.
   c. Update the sample login page to create a version of the page for multiple realms.
   d. Stop and restart the Oracle Single Sign-On middle tier.
3. Grant administrative privileges for multiple realms. This is also discussed in the *Oracle Application Server Single Sign-On Administrator's Guide*.
4. Configure Oracle Single Sign-On as described in "Configure SSO (Optional)" on page 8-9.
5. Configure the SSO authentication method setting in OC4J, as described in "OC4J Configuration for Oracle Single Sign-On Authentication" on page 8-15.

The authentication sequence for single sign-on to multiple realms is much the same as it is for single sign-on in a single, default realm. The only difference from the user's perspective is that, when a user affiliated with the first type of realm is presented with the login screen, the user must enter not only a user name and password but also a new credential, the realm nickname.

Once a user has entered his credentials, both his realm nickname and user name are mapped to entries in Oracle Internet Directory. More specifically, the single sign-on server uses directory metadata to find the realm entry in the directory. Once it finds this entry, the single sign-on server uses realm metadata to locate the user. Once the user's entry is found, his password, an attribute of his entry, is validated. And once his password is validated, he is authenticated.

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

Important: Do not make property settings in the `jazn.xml` file of the OC4J `home` instance until after you have associated the OC4J instance with the Oracle Internet Directory instance. When you do the association, the `<jazn>` element configuration in the `home` instance `jazn.xml` file is rewritten and any previous settings are lost.

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 8–2 summarizes LDAP user and SSL properties, supported through `<property>` subelements under the `<jazn>` element in the `jazn.xml` file of the OC4J `home` instance. 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.

Note: This discussion assumes appropriate SSL configuration has been completed for OC4J and Oracle Internet Directory, as discussed in Chapter 15, "SSL Communication with OC4J", and the *Oracle Internet Directory Administrator's Guide*, respectively.

The resulting configuration is as follows:

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

You must restart OC4J for the changes to take effect.

Table 8–2 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 6-3 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:

- "SSL Authentication" on page 1-4

Here is a sample configuration:

```
<jazn provider="LDAP" location="ldap://www.example.com:389" default-realm="us">
   <property name="ldap.protocol" value="no-ssl"/>
   ...
</jazn>
```

Configuring LDAP Connection Properties

Table 8–3 summarizes LDAP connection properties. Table 8–4 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 8–3 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 8–4 LDAP JNDI Connection Pool Properties

Property Name	Property Definition	Default Value
`jndi.ctx_pool.init_size`	Initial size of 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
`jndi.ctx_pool.timeout`	Timeout value, in milliseconds, for the LDAP JNDI connection pool. (This may be useful, for example, when there is a firewall between the middle tier, including the OracleAS JAAS Provider, and the Oracle Internet Directory. The timeout on the firewall connection could be coordinated with the timeout of the directory connection.)	0 (no timeout)

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, which allows 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 8–5 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 8–5 LDAP Cache Properties

Property	Description	Default
`ldap.cache.policy.enable`	If set to `true`, enables policy cache; if set to `false`, disables policy cache.	`true`
`ldap.cache.realm.enable`	If set to `true`, enables realm cache; if set to `false`, disables realm cache.	`true`
`ldap.cache.session.enable`	If set to `true`, enables session cache; if set to `false`, disables the session cache. This property must be set to `true` when synchronizing the user context with a servlet session. See "Synchronization of OracleAS JAAS Provider User Context with Servlet Sessions" on page 8-11.	`true`
`ldap.cache.initial.capacity`	Initial capacity for the hashmap. This property affects performance; it is important to not set it too low.	20

Table 8–5 (Cont.) LDAP Cache Properties

Property	Description	Default
`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`	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 certain management and administrative tasks. In particular:

- 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.

Tips and Troubleshooting for the LDAP-Based Provider

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 6-13.
   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: In the `jazn.xml` file of the OC4J `home` instance, the `<jazn>` element has the setting `provider="LDAP"` to use the LDAP-based provider. This element also reflects the Oracle Internet Directory location and port.

Using ldapsearch to Retrieve Realm Names from Oracle Internet Directory

As an alternative to the OracleAS JAAS Provider Admintool, you can use LDAP search commands to retrieve a realm name from Oracle Internet Directory, as follows.

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 Internet Directory Administrator's Guide* for information about the ldapsearch command

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 8-23 for how to disable caching.

Accessing the Oracle Single Sign-On Administration Pages

In the Oracle Identity Management 10.1.4 implementation, you can access the Oracle Single Sign-On administration pages as follows:

```
http://host:port/sso
```

You can use this to check Oracle Single Sign-On setup.

Note: In previous releases, the administration pages were accessed as follows:

```
http://host:port/pls/orasso
```

See Also:

- *Oracle Application Server Single Sign-On Administrator's Guide* for additional information about the Oracle Single Sign-On administration pages

9

Login Modules

This chapter discusses login modules supplied with OC4J, as well as how to implement, install, and configure a custom login module. The following topics are covered:

- Initial Login Module Considerations
- Login Modules Supplied with OC4J
- Introducing Custom JAAS Login Modules
- Summary of Choices for Packaging Login Modules
- Configuring the Custom Security Provider in Application Server Control
- Using Admintool to Configure Login Modules and Grant RMI Permission
- Summary of Login Module Configuration in OC4J Configuration Files
- Step by Step: Integrating a Custom Login Module with OC4J
- Custom Login Module Example

See Also:

- "JAAS Authentication: Login Modules" on page 2-13

Initial Login Module Considerations

This section covers the following:

- Specification of the Oracle Login Configuration Provider
- Login Module Notes and Tips

Specification of the Oracle Login Configuration Provider

By default, OC4J specifies usage of the OracleAS JAAS Provider login configuration provider (`oracle.security.jazn.spi.LoginConfigProvider`) by the OC4J JVM, overriding usage of the Sun Microsystems default JAAS login configuration provider.

This is accomplished through the following configuration in the file *ORACLE_HOME*`/j2ee/home/config/jazn.security.props`:

```
login.configuration.provider=oracle.security.jazn.spi.LoginConfigProvider
```

The Oracle login configuration provider uses the `system-jazn-data.xml` file for login module configuration.

Login Module Notes and Tips

Be aware of the following notes regarding the use of login modules with OC4J:

- By convention, a setting of `provider="XML"` is required for an application using custom login modules (custom security provider). See related information about the `custom.loginmodule.provider` property in "Settings in <jazn> for Login Modules" on page 9-23.
- When using custom login modules, it is possible to instruct 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. See related information about the `role.mapping.dynamic` property in "Settings in <jazn> for Login Modules" on page 9-23.

  To ensure that all relevant principals are taken into consideration during authorization, the login module must add the relevant principals (including any roles that the authenticated user belongs to) to the subject during the `commit` phase of the authentication process.
- Login module configuration (under the `<jazn-loginconfig>` element) cannot be in an application-specific `jazn-data.xml` file; it must be in `system-jazn-data.xml`. Configuration will be written to `system-jazn-data.xml` automatically if you configure login modules through Application Server Control or in your `orion-application.xml` file. (Also see "Settings in <jazn-loginconfig> in orion-application.xml" on page 9-23.)
- If you use any identity repository other than the file-based provider or Oracle Internet Directory, you must define an administrative user account and administrator roles, grant the roles to the user, and grant necessary permissions to the roles, as discussed in "Creating the Administrative User and Roles and Granting RMI Permission" on page 10-9.
- 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.
- See "OracleAS JAAS Provider Policy Management" on page 5-12 regarding subject-based policy management when using a custom login module. The policy configuration must be in `system-jazn-data.xml`.
- "Using the File-Based Provider Across an OC4J Group" on page 7-17 discusses how to maintain `system-jazn-data.xml` settings across multiple OC4J instances. This functionality also applies to login modules, using MBean operations such as `setLoginModule`, `remLoginModule`, and `getLoginModuleControlFlagTypes`.

Troubleshooting Tips:

- If an application is configured to use a custom login module but the login module is not found in the classpath, a "class not found" exception is thrown with a message such as the following: "WARNING unable to find LoginModule class: Missing class: Xxxxxx..." See "Summary of Choices for Packaging Login Modules" on page 9-13 for related information.
- 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" />
```

Login Modules Supplied with OC4J

OC4J supplies the set of login modules, including standard J2EE login modules, listed in Table 9–1.

Table 9–1 Login Modules Supplied with OC4J

Login Module	Description
oracle.security.jazn.login.module. RealmLoginModule	OC4J login module for the file-based provider or Oracle Identity Management. (Additional information follows.)
oracle.security.jazn.login.module.db. DBTableOraDataSourceLoginModule	OC4J login module for user data in a database. (Additional information follows.)
oracle.security.jazn.login.module. LDAPLoginModule	OC4J login module for external LDAP providers. **See Also:** Chapter 10, "External LDAP Security Providers"
oracle.security.jazn.login.module.coreid. CoreIDLoginModule	OC4J login module for Oracle Access Manager. **See Also:** Chapter 11, "Oracle Access Manager"

Notes:

- This table does not show additional login modules internal to OC4J.
- Oracle does not currently validate Sun login modules for use with OC4J. (These login modules, in package `com.sun.security.auth.module`, are not currently part of the JDK.)

This rest of this section offers additional information about the following login modules:

- RealmLoginModule
- DBTableOraDataSourceLoginModule

RealmLoginModule

The `RealmLoginModule` class is the default login module, for use with the file-based provider or Oracle Identity Management, and is configured when you configure these security providers through Application Server Control. This configuration is reflected in a `<login-module>` element under `<jazn-loginconfig>` in 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).

`RealmLoginModule` supports the options shown in Table 9–2 (reflected in `<name>` and `<value>` subelements of an `<option>` element under `<login-module>`).

Table 9–2 RealmLoginModule Options

Option	Description	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
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

Notes:

- `RealmLoginModule` does not have to be enabled if your application uses Oracle Single Sign-On authentication.
- `RealmLoginModule` is used only with declarative security, not programmatic security.
- The use of `RealmLoginModule` as a custom login module—in other words, as a custom security provider—is not supported.

See Also:

- "Login Module Settings in system-jazn-data.xml" on page 9-21
- "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19 for information on using the Admintool

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>
```

DBTableOraDataSourceLoginModule

The OC4J 10.1.3.1 implementation supplies a login module you can use if you have a user identity store in a database:

`oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule`

This replaces previous functionality of the `com.evermind.sql.DataSourceUserManager` class, now deprecated (but still supported for backward compatibility, and still documented at the end of this section for completeness). It also replaces some authentication functionality of the deprecated `com.evermind.security.User` class.

Once you have created your database schema and an Oracle data source to connect to the database (as described in the data sources chapter of the *Oracle Containers for J2EE Services Guide*), you are ready to configure the login module.

`DBTableOraDataSourceLoginModule` supports a number of options for specifying such items as data location (table and column names) and password encryption. You can set these options through Application Server Control or the OracleAS JAAS Provider Admintool, with the settings being reflected in the `<jazn-loginconfig>` element of the `system-jazn-data.xml` file.

This section covers the following topics:

> **Note:** `DBTableOraDataSourceLoginModule` is available with the OC4J product and automatically included in the classpath.

DBTableOraDataSourceLoginModule Options

Table 9–3 summarizes the options supported by `DBTableOraDataSourceLoginModule`, noting which options are required, and including default values as applicable and examples as appropriate. There is no configuration of this login module in OC4J as shipped; refer to the sections that follow, "Configuring DBTableOraDataSourceLoginModule in Application Server Control" and "Configuring DBTableOraDataSourceLoginModule in the Admintool", for information about tools you can use in configuring the login module.

Note that `DBTableOraDataSourceLoginModule` supports encrypted passwords in the database table, and you can use custom password encryption algorithms. To use this feature, you must use an implementation of the following interface, discussed in "Implementing DBLoginModuleEncodingInterface for Password Encryption" on page 9-10:

```
oracle.security.jazn.login.module.db.DBLoginModuleEncodingInterface
```

Oracle supplies implementations for the SHA1 and MD5 algorithms.

Table 9–3 DBTableOraDataSourceLoginModule Options

Option	Description
data_source_name (required)	Name of the data source for the database, as configured in the `data-sources.xml` file. Default: n/a Example: jdbc/OracleDS
table (required)	Name of the database table containing user authentication information (user names, passwords, and so on). Default: n/a Example: userinfo
groupMembershipTableName (required)	Name of the database table containing role information. Default: n/a Example: groupinfo
usernameField (required)	Name of the column containing user names, in the table specified in the `table` option. Default: n/a Example: userName

Table 9–3 (Cont.) DBTableOraDataSourceLoginModule Options

Option	Description
passwordField (required)	Name of the column containing passwords, in the table specified in the `table` option. Default: n/a Example: passWord
pw_encoding_class	Name of your password encryption class, if you use password encryption. (See the discussion immediately preceding this table.) Default: oracle.security.jazn.login.module.db.util.DBLoginModuleClearTextEncoder (no encryption) Example: oracle.security.jazn.login.module.db.util.DBLoginModuleSHA1Encoder **See Also**: "Implementing DBLoginModuleEncodingInterface for Password Encryption" on page 9-10
pw_key	The password encryption key, if you use password encryption. This key is accessed by the class specified in the `pw_encoding_class` option. Default: n/a Example: xyz
groupMembershipGroupFieldName (required)	Name of the column containing role names, in the table specified in the `groupMembershipTableName` option.
user_pk_column	Column name for the primary key in the table specified in the `table` option. Default: Value of the `usernameField` option. Example: userName
roles_fk_column	Column name for the foreign key in the table specified in the `groupMembershipTableName` option. Default: Value of the `usernameField` option. Example: userName
casing	The case-sensitivity when comparing login user names to names in the database. Use `sensitive` to require case-sensitive comparisons, `toupper` to convert the login user name to all-uppercase, or `tolower` to convert the login user name to all-lowercase. (If anything other than these three values is specified, the default value will be used.) Default: sensitive Example: toupper

Notes:

- It is permissible to use the same table for user information and role information; in other words, to specify the same table for the `table` and `groupMembershipTableName` options. Notice that in this scenario, however, each user can have only one role. It is typical, and advisable, to use separate tables.
- `DBTableOraDataSourceLoginModule` does not support null or empty passwords.
- If you use password encryption, passwords are compared according to their encrypted values, not the unencrypted strings. When a user logging in types the password, it is encrypted according to the encrypting method of the class you specify in the `pw_encoding_class` option, and compared to the encrypted password stored in the database. No attempt is made to decrypt the password stored in the database.

Configuring DBTableOraDataSourceLoginModule in Application Server Control

"Configuring the Custom Security Provider in Application Server Control" on page 9-15 discusses how to specify and configure a custom login module during deployment, change to a custom login module after deployment, add a login module, or update a login module. You can use these procedures to configure `DBTableOraDataSourceLoginModule`. The Application Server Control Console login module configuration page enables you to specify the login module class, and to specify names and values for any number of properties corresponding to the options documented in the preceding section, "DBTableOraDataSourceLoginModule Options".

Configuring DBTableOraDataSourceLoginModule in the Admintool

As an alternative to using the Application Server Control Console, you can configure `DBTableOraDataSourceLoginModule` through the OracleAS JAAS Provider Admintool. Use of the Admintool for login modules is discussed in "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19.

Here is an example:

```
java -jar jazn.jar -addloginmodule application_name \
     oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule \
     required data_source_name="jdbc/OracleDS" roles_fk_column="username" \
     table="userinfo"  groupMembershipTableName="groupinfo" \
     groupMembershipGroupFieldName="role" usernameField="username" \
     user_pk_column="username" passwordField="password" casing="sensitive"
```

Sample DBTableOraDataSourceLoginModule Settings in system-jazn-data.xml

As with any login module, option settings and other configuration are stored within the `<jazn-loginconfig>` element of the `system-jazn-data.xml` file. Here is an example:

```
<jazn-loginconfig>
   <application>
      <name>application_name</name>
      <login-modules>
         <login-module>
            <class>
              oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
```

```
            </class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>data_source_name</name>
                  <value>jdbc/OracleDS</value>
               </option>
               <option>
                  <name>table</name>
                  <value>userinfo</value>
               </option>
               <option>
                  <name>roles_fk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>groupMembershipGroupFieldName</name>
                  <value>role</value>
               </option>
               <option>
                  <name>user_pk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>passwordField</name>
                  <value>passWord</value>
               </option>
               <option>
                  <name>groupMembershipTableName</name>
                  <value>groupinfo</value>
               </option>
               <option>
                  <name>usernameField</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>casing</name>
                  <value>sensitive</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
   ...
</jazn-loginconfig>
```

See Also:

- "Summary of Login Module Configuration in OC4J Configuration Files" on page 9-21

Principals for DBTableOraDataSourceLoginModule

`DBTableOraDataSourceLoginModule` uses the following principals:

- `oracle.security.jazn.login.module.db.principals.DBUserPrincipal` (for users)
- `oracle.security.jazn.login.module.db.principals.DBRolePrincipal` (for roles)

A subject passed to the DBTableOraDataSourceLoginModule is populated with instances of these principal types.

For a user name in the user table (a name in the column indicated by the usernameField option, in the table indicated by the table option), there would be a corresponding DBUserPrincipal instance in the subject.

For a role name in the role table (a name in the column indicated by the groupMembershipGroupFieldName option, in the table indicated by the groupMembershipTableName option), there would be a corresponding DBRolePrincipal instance in the subject.

You can use the Admintool to grant permissions to a user or role, as follows:

```
% java -jar jazn.jar -grantperm \
       oracle.security.jazn.login.module.db.principals.DBUserPrincipal name \
       permissionclass [permission_parameters]

% java -jar jazn.jar -grantperm \
       oracle.security.jazn.login.module.db.principals.DBRolePrincipal name \
       permissionclass [permission_parameters]
```

In this syntax, *name* is the name of the DBUserPrincipal or DBRolePrincipal instance, *permissionclass* is the fully qualified name of the permission class for the permission you are granting, and *permission_parameters* are any appropriate parameters of the permission class (login for RMIPermission, for example).

Important: Ensure that permission classes are in the classpath.

Implementing DBLoginModuleEncodingInterface for Password Encryption

To use password encryption with DBTableOraDataSourceLoginModule, you must use an implementation of the following interface to perform the encryption:

```
oracle.security.jazn.login.module.db.DBLoginModuleEncodingInterface
```

The implementing class must implement the following method, which is called by the login module:

- `String getKeyDigestString(String text, String key)`

This method takes a text string to encrypt (such as a password) and a string that specifies a key, as set in the pw_key option for the login module (if applicable). The method would then encrypt the text string with any desired algorithm (with the key if specified), and preferably also formats the post-encryption binary data using any desired encoding standard (Base64Encoding, for example). It outputs the encrypted digest string.

For algorithms that do not use a key, such as MD5 or SHA1, any key passed could be ignored. If the pw_key option is not set, then its value is null and null is passed.

Oracle provides the following implementations in the oracle.security.jazn.login.module.db.util package:

- DBLoginModuleSHA1Encoder

  This class generates a hash value for the given password string using the SHA1 algorithm, and encodes the binary hash using Base64Encoding. The binary hash of the password must be stored in the database using Base64Encoding.

- DBLoginModuleMD5Encoder

This class generates a hash value for the given password string using the MD5 algorithm, and encodes the binary hash using `Base64Encoding`. The binary hash of the password must be stored in the database using `Base64Encoding`.

- `DBLoginModuleClearTextEncoder`

  If you do not specify an encryption class, `DBLoginModuleClearTextEncoder` is used, resulting in no encryption. This class simply passes the password string as it is given.

Previous Functionality: DataSourceUserManager (Deprecated)

Prior to implementation of `DBTableOraDataSourceLoginModule` in the OC4J 10.1.3.1 implementation, equivalent functionality was available through the `com.evermind.sql.DataSourceUserManager` class. While this feature is currently still supported for backward compatibility, it is deprecated and will be desupported in future releases. You should use `DBTableOraDataSourceLoginModule` instead.

However, for completeness, we include the following documentation for the `DataSourceUserManager` feature.

Notes:

- The database you use 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.
- In OC4J 10.1.3.x implementations, `DataSourceUserManager` obtains group information only from the database, which differs from the behavior in previous implementations. Therefore, you must map groups to users in the database, as applicable.

When you configure `DataSourceUserManager` (as described shortly), you can specify values for the properties described in Table 9–4, 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 9–4 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.

Table 9–4 (Cont.) DataSourceUserManager Properties

Property	Description
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.

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 Application Server Control 10.1.3.x implementations.

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.

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>
```

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 9-28 includes debugging functionality.)
- Packaging and deployment: Are you using login modules that come with OC4J or J2SE 1.4? Or are you deploying custom or third-party login modules? And are you packaging the login modules with your application, or making them available separately as an optional package or library?

When you associate a custom login module with an application (through configuration, as shown later), 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 9-23, 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. You can also configure login modules through `orion-application.xml`, in which case the configuration is copied to `system-jazn-data.xml`.

See Also:

- "JAAS Authentication: Login Modules" on page 2-13
- Sun Microsystems JAAS documentation for general information about developing custom login modules:

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

Summary of Choices for Packaging Login Modules

If you are using one or more of the default login modules provided with OC4J or the J2SE, 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:

- Packaging Login Modules within the J2EE Application
- Providing Login Modules as Optional Packages
- Providing Login Modules as OC4J Shared Libraries

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

Important: If an application is configured to use a custom login module but the login module is not found in the classpath, a "class not found" exception is thrown with a message such as the following: "WARNING unable to find LoginModule class: Missing class: Xxxxxx..."

Packaging 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 9-21
- In the `orion-application.xml` file in your application EAR file, as discussed in "Settings in <jazn-loginconfig> in orion-application.xml" on page 9-23

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, or make it available as an optional package or shared library.

See Also:

- "Configuring the Custom Security Provider in Application Server Control" on page 9-15
- "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19

Providing 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 the `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 9-21.

See Also:

- For general information about the standard "optional package" mechanism:

  `http://java.sun.com/j2se/1.4.2/docs/guide/extensions`

Providing 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, as documented in "Tasks to Share a Library" on page 6-14.

Configuring the Custom Security Provider in Application Server Control

This section discusses the following administration tasks for the custom security provider (custom 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

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).
- Any login module you specify must be in the classpath at runtime. (One option is to load it as a shared library, as described in "Tasks to Share a Library" on page 6-14.)
- Custom login modules must not be named `other`, `ascontrol`, and `default`. These names are reserved for the predefined JAAS login modules.

Specifying and Configuring a Custom Security Provider during Deployment

When you plan to use a 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 6-8 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 9-17.
   - 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 9-16.
   - 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 6-8.

Deploying or configuring a custom login module through Application Server Control results in the following required settings being inserted automatically 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.
- By convention, there is a `<jazn>` setting of `provider="XML"` when you use custom login modules.
- 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 9-21.

See Also:

- "Summary of Login Module Configuration in OC4J Configuration Files" on page 9-21 and "Step by Step: Integrating a Custom Login Module with OC4J" on page 9-25 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 earlier in "Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15 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 9–5 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 9-15.

Table 9–5 Login Module Control Flags

Flag	Meaning
`Required`	The login module is nominally required to succeed. Whether or not it succeeds, however, authentication proceeds down the login module list.
`Requisite`	The login module nominally 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.
`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.

These control flag settings are used according to standard functionality of the `javax.security.auth.login.Configuration` class. The overall authentication succeeds only if all "required" and "requisite" login modules succeed, possibly unless a "sufficient" login module is configured and succeeds—in that case, only the required and requisite login modules prior to the sufficient login module in the login module list must succeed.

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 earlier in "Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15 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 Table 9–5, " Login Module Control Flags" in the preceding section 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 9-15.

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 6-13.
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 Table 9–5, " Login Module Control Flags" on page 9-17 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 are prompted to restart your application for the changes to take effect.

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:

Note: Custom login modules must not be named `other`, `ascontrol`, and `default`. These names are reserved for the predefined JAAS login modules.

---

1. Go to the Security Provider page for your application, as described in "Navigating to the Security Provider Page for Your Application" on page 6-13.
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 Table 9–5, " Login Module Control Flags" on page 9-17 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 6-13.
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 Table 9–5, " Login Module Control Flags" on page 9-17 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 6-13.
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.

Using Admintool to Configure Login Modules and Grant RMI Permission

This section describes how to use the OracleAS JAAS Provider Admintool in the following ways:

- As an alternative to the Application Server Control Console for adding and configuring login modules

- To grant RMI permission "login" to appropriate principals to access EJBs through RMI

See Also:

- Appendix C, "OracleAS JAAS Provider Admintool Reference"

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 Table 9–5, " Login Module Control Flags" on page 9-17.

  If the login module supports 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.

Important: Restart OC4J for changes to take effect.

See Also:

- "Adding and Removing Login Modules" on page C-10
- "Listing Login Modules" on page C-15

Granting RMI Permission through the Admintool

In case your application includes EJBs, note that to access an EJB through RMI, you must grant RMI permission "login" to the appropriate user, role, or principal. You can accomplish this through the Admintool, as in the following examples:

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

```
% java -jar jazn.jar -grantperm oracle.security.jazn.samples.SamplePrincipal \
       managers com.evermind.server.rmi.RMIPermission login
```

(Always specify a realm when granting permissions to a user or role, but not when granting permissions to a principal.)

If you are in an Oracle Application Server environment with multiple OC4J instances, then you must specify the appropriate instance name (and hence the appropriate home directory) for the applicable OC4J instance, so that the `system-jazn-data.xml` file for the appropriate OC4J instance is updated. Set `%oracleas.oc4j.instance%` appropriately for your environment.

```
% java -jar -Doracle.j2ee.home=%ORACLE_HOME%/j2ee/%oracleas.oc4j.instance% \
       jazn.jar -grantperm myrealm -role managers \
       com.evermind.server.rmi.RMIPermission login

% java -jar -Doracle.j2ee.home=%ORACLE_HOME%/j2ee/%oracleas.oc4j.instance% \
       jazn.jar -grantperm oracle.security.jazn.samples.SamplePrincipal managers \
       com.evermind.server.rmi.RMIPermission login
```

Important: Restart OC4J for changes to take effect.

See Also:

- "Granting and Revoking Permissions" on page C-14

Summary of 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
- Login Module Settings in oc4j-ra.xml (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 9-23, the `system-jazn-data.xml` file will be updated accordingly.

Example 9–1 Example jazn-loginconfig element

```
<jazn-data>
  ..
  <jazn-loginconfig>
    <application>
      <name>myapp</name>
      <login-modules>
        <login-module>
          <class>mypath.MyLoginModule</class>
          <control-flag>required</control-flag>
          <options>
            <option>
              <name>myoptionname</name>
              <value>myoptionvalue</value>
            </option>
          </options>
        </login-module>
      </login-modules>
    </application>
  </jazn-loginconfig>
  ..
</jazn-data>
```

This fragment associates the login module `MyLoginModule` with the application `myapp`.

Note: Do not remove login configuration information for `RealmLoginModule` from the `system-jazn-data.xml` file.

See Also:

- "The system-jazn-data.xml File" on page 4-7
- Table 9–5, " Login Module Control Flags" on page 9-17 for information about control flag settings
- "Configuring the Custom Security Provider in Application Server Control" on page 9-15
- "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19

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 Access to JNDI Context

See Also:

- *Oracle Containers for J2EE Developer's Guide* for general reference information about `orion-application.xml`

Settings in <jazn-loginconfig> in orion-application.xml

Settings in the `<jazn-loginconfig>` element in `system-jazn-data.xml` were documented in "Login Module Settings in system-jazn-data.xml" on page 9-21. You can add this element to `orion-application.xml` prior to 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`, or when you deploy and configure your custom login module using the Application Server Control Console as discussed in "Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15.

```
<jazn provider="XML" ... >
   <property name="role.mapping.dynamic" value="true" />
   <property name="custom.loginmodule.provider" value="true" />
</jazn>
```

Settings in <namespace-access> for Access to JNDI Context

If an application contains an EJB, remote clients must be given namespace access to read (look up) and write (bind) objects as required on the server-side JNDI context of the application.

The following example, which would appear in `orion-application.xml`, shows how the namespace access is granted for read operations to roles named `managers` and `developers`.

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
```

```
            </security-role-mapping>
            <security-role-mapping name="sr_manager">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

This assumes the indicated role mappings had already been set up elsewhere in `orion-application.xml`.

See Also:

- "Configuring Namespace Access" on page 18-8

Specifying the Mapping Attribute

For login modules, JAZN sets the mapping attribute for the Oracle Internet Directory (OID) provider to `"DN"` by default.

To change the default value, you can configure the value of the `mapping.attribute` property:

1. Locate the `$Oracle_Home/j2ee/<oc4j_inst>/config/jazn.xml` file.
2. Add the `mapping.attribute` property configuration to the <jazn> tag. For example:

```
<jazn provider... >
...
   <property name = "mapping.attribute" value="cn"/>
...
</jazn>
```

3. Restart the OC4J instance.

The login modules that rely on the `mapping.attribute` attribute in the `jazn.xml` file, include:

- WS-Security Username token with no password (`WSSLoginModule`)
- SAML (`SAMLLoginModule`)

  `SAMLLoginModule` first looks at the `SubjectNameIdentifier` to determine the mapping. If blank, then the `mapping.attribute` is used.
- X.509 client certificate authentication (`X509LoginModule`)
- Third party login module (`LDAPLoginModule`)

The mapping attribute for these login modules must be set in the `jazn.xml` file as described here.

Login Module Settings in oc4j-ra.xml (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.

```
<oc4j-connector-factories ... >
  ...
  <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>
  ...
</oc4j-connector-factories>
```

See Also:

- *Oracle Containers for J2EE Resource Adapter Administrator's Guide* for additional information about configuring resource adapters

Step by Step: Integrating a Custom Login Module with OC4J

Developing a login module follows the standard development, packaging, and deployment cycle. For applications with EJBs, you must also grant RMI permission and configure namespace access as necessary. The following sections discuss each step in the cycle:

1. Develop the Login Module
2. Configure and Package the Login Module
3. Configure Namespace Access and Role Mappings (as applicable)
4. Deploy the Login Module
5. Grant RMI Permission (as applicable)
6. Set JNDI Properties (as applicable)

Develop the Login Module

Develop a JAAS-compliant `LoginModule` implementation according to the JAAS service provider interface.

When a login module is associated with an application (which occurs in the `<jazn-loginconfig>` configuration shown earlier), OC4J uses the principals set in the subject to determine if the client has access to the requested J2EE resource. Thus, it is imperative that the login module correctly set all the roles the user is a member of.

In the following example, the login module authenticates the `developer` user. After the user is successfully authenticated, a principal corresponding to the roles associated with the user is added to the current authenticated subject. Assume the `developer` user is a member of the `developers` role.

```
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");
}
```

See Also:

- `javax.security.auth.spi.LoginModule` Javadoc:

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

- "Custom Login Module Example" on page 9-28

Configure and Package the Login Module

"Summary of Choices for Packaging Login Modules" on page 9-13 summarizes the various ways you can package a login module so that it is accessible to your application. The choices are:

- Package the login module with the application.
- Provide the login module as an optional package.
- Provide the login module as an OC4J shared library.

There are also choices for how you can specify the configuration of a login module:

- Manually configure the `<jazn-loginconfig>` element in your `orion-application.xml` file, as discussed in "Settings in <jazn-loginconfig> in orion-application.xml" on page 9-23. This may be particularly appropriate if you are packaging the login module with your application. The `<jazn-loginconfig>` element in `system-jazn-data.xml` is updated accordingly.
- Use the Application Server Control Console to specify the login module settings, such as when you deploy your application, as discussed in "Configuring the Custom Security Provider in Application Server Control" on page 9-15. The `<jazn-loginconfig>` element in `system-jazn-data.xml` is configured automatically.
- Use the OracleAS JAAS Provider Admintool to specify the login module settings, as discussed in "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19. The `<jazn-loginconfig>` element in `system-jazn-data.xml` is configured automatically.

The following sections highlight key configuration details.

Configuration to Enable Login Module Usage

OC4J requires the following configuration in the `orion-application.xml` file in order to use a custom login module to perform security checks. This occurs automatically when you specify login module configuration through Application Server Control or in `orion-application.xml`.

```
<jazn provider="XML" >
   <property name="role.mapping.dynamic" value="true"/>
   <property name="custom.loginmodule.provider" value="true"/>
</jazn>
```

You can examine the `orion-application.xml` file after deployment to confirm this.

Configuration of the Login Module

The following configuration is for the sample login module shown in "Custom Login Module Example" on page 9-28.

```
<!-- Configuring a Login Module in an Application EAR file.  -->
<jazn-loginconfig>
  <application>
    <name>cutomjaas</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>
```

You could manually put this configuration into the `orion-application.xml` file before deployment, or set the parameters (class name, control flag, and option names and values) as prompted during deployment when you deploy your application through Application Server Control. In either case, the configuration will automatically be written to `system-jazn-data.xml`.

Configure Namespace Access and Role Mappings (as applicable)

Configure namespace access in `orion-application.xml` if the application includes any EJBs. This is to allow remote client access to JNDI bindings for read (lookup) or write (bind) operations as required on objects in the server-side JNDI context of the application. The following example shows how the namespace access is granted to roles named `managers` and `developers`.

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
            </security-role-mapping>
            <security-role-mapping name="sr_manager">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

This assumes the role mappings—mapping logical roles defined in the application to roles supported by the custom login module—have been properly defined, as in the following example from `orion-application.xml`:

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

```
   <!-- Mapping the logical roles to the container roles -->
   <security-role-mapping name="sr_manager">
      <group name="managers" />
   </security-role-mapping>
   <security-role-mapping name="sr_developer">
      <group name="developers" />
   </security-role-mapping>
   ...
</orion-application>
```

You can specify role mappings through Application Server Control, as discussed in "Mapping Security Roles" on page 6-10. This results in the `orion-application.xml` entries being added automatically.

Deploy the Login Module

"Summary of Choices for Packaging Login Modules" on page 9-13 discusses how you can deploy your login module either with your application or separately (as an optional package or shared library).

"Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15 discusses how to configure and deploy your login module through Application Server Control when you deploy your application.

Grant RMI Permission (as applicable)

If your application includes EJBs, be aware that to access an EJB through RMI, you must grant RMI permission "login" to the appropriate users, roles, and principals. You can accomplish this through the Admintool, as shown in "Granting RMI Permission through the Admintool" on page 9-20.

Set JNDI Properties (as applicable)

To access resources through JNDI, you must configure the `jndi.properties` file to set properties such as the provider URL, principal, and credential, as appropriate for your environment. Here is an example:

```
java.naming.factory.initial=
                    oracle.j2ee.naming.ApplicationClientInitialContextFactory
java.naming.provider.url=ormi://localhost:23791/customjaas
#java.naming.provider.url=opmn:ormi://localhost:6003:home/customjaas
java.naming.security.principal=manager
java.naming.security.credentials=welcome
```

(The `ormi` protocol is for standalone OC4J, while the `opmn:ormi` protocol is for an Oracle Application Server environment. Uncomment the appropriate one.)

Custom Login Module Example

This section shows the login module code and principal code for an OC4J "how-to" example at the following location:

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

Look for the login module how-to under "J2EE Security / JAAS".

SampleLoginModule Code

This section contains the code for the sample login module.

```
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 LoginModule.
     *
     * subject         the Subject to be authenticated.
     * callbackHandler a CallbackHandler for communicating
     *                 with the end user (prompting for usernames and
     *                 passwords, for example).
     * sharedState     shared LoginModule state.
     * options         options specified in the login
     *                 Configuration for this particular
     *                 LoginModule.
     */
    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.
 *
 * return true if the authentication succeeded, or false if this
 *        LoginModule should be ignored.
 * throws FailedLoginException if the authentication fails.
 * throws LoginException       if this LoginModule
 *                             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
    // element 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");
        }

      if (username.equals("sirish") && password.equals("sirish"))
      {
            _succeeded = true;
            _password = password.toCharArray();
            _name = "sirish";
            _authPrincipals = new SamplePrincipal[1];
            _authPrincipals[0] = new SamplePrincipal("sirish");
       }

        ((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;
    }

    /*
     * This method is called if the LoginContext's
     * overall authentication succeeded
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * succeeded).
     *
     * If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * login method, then this method associates a
     * Principal with the Subject located in the
     * LoginModule.  If this LoginModule's own
     * authentication attempted failed, then this method removes
     * any state that was originally saved.
```

```
     *
     * 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());
        }
    }

    /*
     * This method is called if the LoginContext's
     * overall authentication failed.
     * (the relevant REQUIRED, REQUISITE, SUFFICIENT and OPTIONAL LoginModules
     * did not succeed).
     *
     * If this LoginModule's own authentication attempt
     * succeeded (checked by retrieving the private state saved by the
     * login and commit methods),
     * then this method cleans up any state that was originally saved.
     *
     * 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.
 *
 * This method removes the Principal
 * that was added by the commit method.
 *
 * return true in all cases since this LoginModule
 *        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) {
        }
    }
}
```

SamplePrincipal Code

This section contains the code for the sample principal.

The login module sets the `SamplePrincipal` instance in the current `Subject` instance. To create the `SamplePrincipal` instance, the login module directly invokes the `SamplePrincipal` constructor, so the constructor is defined as public.

```
package oracle.security.jazn.samples;

import java.security.Principal;

/*
 * This class implements the Principal interface
 * and represents a Sample user.
 *
 * Principals such as this SamplePrincipal
```

```
 * may be associated with a particular Subject
 * to augment that Subject with an additional
 * identity.  Authorization decisions can then be based upon
 * the Principals associated with a Subject.
 *
 */
public class SamplePrincipal implements Principal {

    private String _name = null;


    /*
     * Create a SamplePrincipal with a Sample username.
     *
     */
    public SamplePrincipal(String name) {
       if (name == null)
       throw new NullPointerException("name cannot be null");
        _name = name;
    }

    /*
     * Return a string representation of this SamplePrincipal.
     *
     */
    public String getName() {
        return _name;
    }

    /*
     * Return a hash code for this SamplePrincipal.
     *
     */
    public int hashCode() {
        return _name.hashCode();
    }

    /*
     * Return a string representation of this SamplePrincipal.
     *
     */
    public String toString() {
        return "[SamplePrincipal] : " + _name;
    }

    /*
     * Compares the specified Object with this SamplePrincipal
     * for equality.  Returns true if the given object is also a
     * SamplePrincipal and the two SamplePrincipals
     * have the same username.
     *
     */
    public boolean equals(Object o) {
      if (o == null)
          return false;

        if (this == o)
            return true;

        if (!(o instanceof SamplePrincipal))
```

```
            return false;
          SamplePrincipal that = (SamplePrincipal)o;

        if (this.getName().equals(that.getName()))
            return true;
        return false;
      }
  }
```

10

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
- Creating Necessary Accounts and Granting Necessary Permissions
- Sample Configuration for Sun Java System Directory Server
- Using SSL with External LDAP Providers

OC4J 10.1.3.x implementations support 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.
- Beginning with OC4J 10.1.3.x implementations, external LDAP providers are supported in standalone OC4J as well as in an Oracle Application Server environment.
- The concept of security realms is not supported when using external LDAP providers.
- See "OracleAS JAAS Provider Policy Management" on page 5-12 regarding subject-based policy management when using an external LDAP provider. The policy configuration must be in `system-jazn-data.xml`.

See Also:

- For information about user and role APIs that you can use with external LDAP providers, Chapter 12, "User and Role API Framework"

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 6-9.

> **Note:** This is assuming 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 that by convention, the `<jazn>` setting `provider="XML"` is used for external LDAP providers.
> - 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 would 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 to configure `LDAPLoginModule` are reflected within a `<login-module>` element under `<jazn-loginconfig>` in `system-jazn-data.xml`.

> **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 6-8 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 10–1, " Application Server Control External LDAP Provider Options" on page 10-4
   - Table 10–2, " Application Server Control External LDAP Connection Pool Options" on page 10-4 (if you enable connection pooling)
   - Table 10–3, " Application Server Control External LDAP User Options" on page 10-5
   - Table 10–4, " Application Server Control External LDAP Role and Member Options" on page 10-5

Or, alternatively, choose **Set Values to Vendor Defaults** for the vendor specified through the LDAP Directory Vendor setting. Apart from this, you must still specify LDAP Location, User DN, User Search Base, and Group Search Base.

4. You can optionally choose **Test LDAP Authorization** prior to deployment. This tests whether the LDAP session can successfully be created, thereby confirming key settings such as the LDAP host, port, administrative user, and password.
5. Choose **OK** to finish the security provider selection.
6. 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 6-8.

Table 10–1 Application Server Control External LDAP Provider Options

Option	Required? Or Settings / Default	Equivalent Option in Table 10–5 (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
User Password	No default	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 10–2 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
```

These are described at:

http://java.sun.com/products/jndi/tutorial/ldap/connect/config.html

Table 10–3 Application Server Control External LDAP User Options

Option	Required? Or Settings / Default	Equivalent Option in Table 10–6 (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 10–4 Application Server Control External LDAP Role and Member Options

Option	Required? Or Settings / Default	Equivalent Option in Table 10–7 (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 in the preceding section. 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 6-13.
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 the following tables in the preceding section:
   - Table 10–1, " Application Server Control External LDAP Provider Options"
   - Table 10–2, " Application Server Control External LDAP Connection Pool Options" (if you enable connection pooling)
   - Table 10–3, " Application Server Control External LDAP User Options"

- Table 10–4, " Application Server Control External LDAP Role and Member Options"

Or, alternatively, choose **Set Values to Vendor Defaults** for the vendor specified through the LDAP Directory Vendor setting. Apart from this, you must still specify LDAP Location, User DN, User Search Base, and Group Search Base.

5. You can optionally choose **Test LDAP Authorization** prior to deployment. This tests whether the LDAP session can successfully be created, thereby confirming key settings such as the LDAP host, port, administrative user, and password.
6. Choose **OK** to finish the change.

This takes you back to the Security Provider page, where you are prompted to restart your application for the changes to take effect.

Important: Also confirm that role mapping is set up appropriately, to correctly map deployment roles (roles defined in the external LDAP provider) to J2EE logical roles. Refer to "Mapping Security Roles" on page 6-10 for additional information.

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 setting in a `<login-module>` element is specified in an `<option>` element (subelement of `<options>`) and corresponds to a configuration setting in the external LDAP provider. The `<name>` subelement of an `<option>` element specifies the name of the option, and the `<value>` subelement specifies the corresponding value.

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 10-3, 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 10–5, Table 10–6 , and Table 10–7 following. 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 10-11 for examples of some option settings

Table 10–5 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.
oracle.security.jaas.ldap.context.referral	(Optional) Can have one of the following values: `follow`, `throw`, or `ignore`. It indicates how referrals are handled by the login module. If not specified, the provider's default one will be used. By default, this property is not specified.
oracle.security.jaas.ldap.object.category	(Optional) This is to be used only with Active Directory and specifies the `objectCategory` used in the group search filter. If not specified, `objectCategory` will not be included in the filter. By default, this property is not specified.

Table 10–6 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 semicolon-delimited distinguished names (DNs) in the LDAP directory that contains users. Here is a sample DN:
	`ou=bpelgroup,dc=ad,dc=vm,dc=oracle,dc=com;`
	`ou=applgroup,dc=ad,dc=vm,dc=oracle,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 10–7 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 semicolon-delimited distinguished names (DN) in the LDAP directory that contains roles. For example:
	`ou=bpelgroup,dc=ad,dc=vm,dc=oracle,dc=com;`
	`ou=applgroup,dc=ad,dc=vm,dc=oracle,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`.

Here is an example:

```
<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.provider.url</name>
                     <value>ldap://myhost.example.com:389</value>
                  </option>
                  ...more options...
               </options>
            </login-module>
            ...
         </login-modules>
      </application>
      ...
   </jazn-loginconfig>
   ...
</jazn-data>
```

See "Settings in system-jazn-data.xml for Sun Java System Directory Server" on page 10-11 for the complete example.

Creating Necessary Accounts and Granting Necessary Permissions

This section discusses steps you must take when using an external LDAP provider to create necessary accounts and grant necessary permissions, covering the following topics:

- Creating the Administrative User and Roles and Granting RMI Permission
- Granting RMI Permission to an LDAP Principal
- Granting Additional Permissions to External LDAP Principals
- Using JAAS Mode with External LDAP Providers

Creating the Administrative User and Roles and Granting RMI Permission

When you use an external LDAP provider, you must define an administrative user account and administrator roles, grant the roles to the user, and grant necessary permissions to the roles. (These steps are handled automatically in the file-based security provider and Oracle Internet Directory.)

1. Create an administrative user account in the external LDAP directory, using the appropriate tool for the security provider. (The name `oc4jadmin` is used for the administrator account by convention, but you can use any name.)
2. Create the administrator roles `oc4j-administrators` and `ascontrol_admin` in the external LDAP directory, using the appropriate tool. These roles must be under the group search base configured for the LDAP provider. (See Table 10–4 on page 10-5 and Table 10–7 on page 10-8 for information about the group search base.)
3. Grant these roles to the administrative user, using the appropriate tool.
4. Create the additional administrator roles `oc4j-app-administrators`, `ascontrol_appadmin`, and `ascontrol_monitor`. (These do not have to be granted to the administrative user.)
5. If the administrator must access EJBs, grant these roles RMI permission "login". You can use the OracleAS JAAS Provider Admintool for this, as in the following example:

   ```
   % java -jar jazn.jar -grantperm myrealm -role oc4j-administrators \
        com.evermind.server.rmi.RMIPermission login
   ```

   Be aware that although the roles are defined in the external LDAP provider, these permission grants are stored in the `system-jazn-data.xml` file.

   See Also:

   - "Predefined Accounts" on page 4-11

Granting RMI Permission to an LDAP Principal

When using an external LDAP provider for an EJB application, it is necessary to grant RMI permission "login" for the appropriate LDAP principal for EJB access.

The following example uses the OracleAS JAAS Provider Admintool to accomplish this for an LDAP principal `hobbes`:

```
% 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, assuming that is your policy repository.

```
<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>
```

Granting Additional Permissions to External LDAP Principals

Any additional permission required by any external LDAP principal can be granted in the same way as RMI permission, shown in the preceding section.

Using JAAS Mode with External LDAP Providers

The use of OC4J JAAS mode is supported for an application that uses an external LDAP provider, in case that is required by your application in checking authorizations with respect to the permissions you have granted. It is configured as shown in the following example:

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

Refer to "Introduction to JAAS Mode" on page 5-5 and "Configuring and Using JAAS Mode" on page 5-18 to gain an understanding of when and how to use this mode.

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 10–1 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 an 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 10–1 on page 10-11. The corresponding `<jazn-loginconfig>` entries in the `system-jazn-data.xml` file are shown in the following example:

Example 10–2 JAAS Login Module Configuration Corresponding to Example 10–1

```
<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.provider.url</name>
                     <value>ldap://myhost.example.com:389</value>
                  </option>
                  <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>
```

Settings in orion-application.xml for an 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`.

Using SSL with External LDAP Providers

This section discusses how to use Secure Sockets Layer communication when using an external LDAP provider with OC4J, covering the following topics:

- Initial SSL Considerations for External LDAP Providers
- Configuring OC4J to Use SSL with an External LDAP Provider
- Configuring the External LDAP Provider for SSL

See Also: For information about OC4J server-side SSL support:

- Chapter 15, "SSL Communication with OC4J"

Initial SSL Considerations for External LDAP Providers

"SSL Authentication" on page 1-4 discusses the various authentication modes for Secure Sockets Layer communication—no authentication, one-way authentication, and two-way authentication. Be aware that most LDAP providers do not support "no authentication" mode (although Oracle Internet Directory does).

Configuring OC4J to Use SSL with an External LDAP Provider

To enable SSL communication to an external LDAP provider, set the LDAP provider option `oracle.security.jaas.ldap.provider.url` to a URL value that specifies the LDAPS protocol, the host name, and an appropriate port for SSL communication. The syntax is similar to that for plain LDAP URLs discussed earlier, but the default port for LDAPS is 636 instead of 389.

When you use the Application Server Control Console for external LDAP provider configuration, this option corresponds to the setting for LDAP Location, as noted in "Specifying and Configuring an External LDAP Provider during Deployment" on page 10-3. This option is also discussed in "External LDAP Provider Settings in system-jazn-data.xml" on page 10-6.

For example, setting LDAP Location to `ldaps://myhost.example.com:636` results in `oracle.security.jaas.ldap.provider.url` being set in `system-jazn-data.xml` as follows:

```
<login-module>
   <class>oracle.security.jazn.login.module.LDAPLoginModule</class>
   <control-flag>...</control-flag>
   <options>
      <option>
         <name>oracle.security.jaas.ldap.provider.url</name>
         <value>ldaps://myhost.example.com:636</value>
      </option>
      ...more options...
   </options>
</login-module>
```

Important: This configuration is sufficient for SSL in "no authentication" mode, but any external LDAP provider presumably requires at least one-way authentication (for communication to the LDAP server). The additional steps for that are discussed in the next section, "Configuring the External LDAP Provider for SSL".

Configuring the External LDAP Provider for SSL

SSL communication between OC4J and an external LDAP provider requires the following:

- The external LDAP provider must be configured to use SSL. It must have a wallet or keystore that contains its certificate as well as the certificate of the CA that issued its certificate (issued the external LDAP provider certificate).
- OC4J must be configured to use SSL, as discussed in Chapter 15, "SSL Communication with OC4J". Its wallet or keystore must contain a certificate that identifies the container, as well as the certificate of the CA that issued its certificate (issued the OC4J certificate).
- All trustpoints (CAs) used in the process must be imported into the wallets or keystores used by the external LDAP provider and by OC4J. If different CAs are used to sign the OC4J and external LDAP provider certificates, then the OC4J and external LDAP provider wallet or keystore each must contain a copy of the certificate of each of these CAs.

You can use the following general steps to configure an external LDAP provider, such as Active Directory or Sun Java System Directory Server, for SSL using standard JSSE functionality and one-way authentication (two-way authentication is not supported for use with external LDAP providers in the current release):

1. Import the root CA certificate from the directory server to your local machine. For example, create your own keystore and then import the root CA certificate to this keystore. You can accomplish this using the JSSE `keytool`.
2. Set Java system properties as necessary for your truststore:
   - Set the `-Djavax.net.ssl.trustStore` property to indicate the path to your wallet or keystore that serves as truststore.
   - If you need to secure the truststore, also set the `-Djavax.net.ssl.trustStorePassword` property. (This may not necessary for one-way authentication, as the truststore would typically contain only public keys.)

   For standalone OC4J, accomplish this on the JVM command line. In an Oracle Application Server environment, accomplish this through Java property settings for the OC4J instance in the `opmn.xml` file.

Here is a sample `opmn.xml` entry for the OC4J `home` instance:

```
<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
                  -Dhttp.webdir.enable=false"
                  -Djavax.net.ssl.trustStore=pathtotruststore/>
         </category>
         ...
      </module-data>
      ...
   </process-type>
</ias-component>
```

See Also:

- "Using Keys and Certificates with OC4J and Oracle HTTP Server" on page 15-3 and "Using SSL with Standalone OC4J" on page 15-5 for introductory information about `keytool`
- For detailed information about the `keytool` utility:

  `http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html`

- *Oracle Containers for J2EE Configuration and Administration Guide*, in the chapter on OC4J runtime configuration, for general information about setting system properties

11

Oracle Access Manager

This chapter discusses how to use the Oracle Access Manager security provider, formerly known as Oracle COREid Access and Identity, for authentication, authorization, and single sign-on. The chapter is useful for those who have already deployed, or plan to deploy, the Oracle Access Manager system. It describes integration between the Oracle Access Manager 10.1.4 implementation and the OC4J 10.1.3.1 implementation, and how to secure applications deployed on OC4J through features of Oracle Access Manager. 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 Access Manager with Oracle Application Server. Some prior familiarity with Oracle Access Manager is assumed, and there is no attempt to thoroughly document the use of its features or tools. In terms of what to do for necessary Oracle Access Manager 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 Oracle Access Manager documentation for details on its features and administration.
- Only the 10.1.4 implementation of Oracle Access Manager is supported with the 10.1.3.1 implementation of OC4J.

See Also:

- *Oracle Access Manager Installation Guide*
- *Oracle Access Manager System Administration Guide* for information related to settings of the Access Server (described later in this chapter) and definition of access controls and user access to applications and data
- *Oracle Access Manager Identity and Common Administration Guide* for information related to Identity Server settings and functions common to both the Access Server and Identity Server (both described later in this chapter)
- *Oracle Access Manager Developer Guide*

These and other Oracle Access Manager documents are available through:

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

They are available with the Oracle Identity Management 10.1.4 documentation release, listed under "Application Servers".

Getting Started with Oracle Access Manager

This section provides an overview of Oracle Access Manager and discusses prerequisites and architecture:

Overview of Oracle Access Manager

Oracle Access Manager 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 Oracle Single Sign-On), personalization, and user profile management in heterogeneous application

environments across a variety of application servers, legacy applications, and databases. Oracle Access Manager 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, Oracle Access Manager 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 Access Manager 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 Access Manager instance, allowing centralized single sign-on and auditing functionality, as well as more robust authentication options.
- Oracle Access Manager offers superior identity administration through workflow, fine-grained attribute control, and delegation of administration.
- Oracle Access Manager supports access control based on dynamic groups, with members based on a given identity profile.
- Oracle Access Manager allows realtime access and identity integration, with runtime changes made through Oracle Access Manager being automatically populated into the Access Server cache to eliminate security loopholes.

In OC4J 10.1.3.x implementations, OracleAS JAAS Provider supports Oracle Access Manager integration through a custom login module and a special authentication method setting.

Oracle Access Manager 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 Oracle Access Manager 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 an Oracle Access Manager Identity Server.
- Oracle Access Manager 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.
- Policy Manager writes policy data to Oracle Internet Directory (or other suitable LDAP server), 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 10.1.3.1 implementation, Application Server Control does not support configuration of Oracle Access Manager.

Oracle Access Manager Prerequisites

This section describes what you must have installed to use Oracle Access Manager. Oracle Access Manager components are version 10.1.4.

See Also:

- *Oracle Access Manager Installation Guide* for installation instructions

At a high level, prerequisites include installing Oracle Access Manager and Oracle Application Server, and configuring the Access Manager SDK and your applications on OC4J.

Detailed requirements on the Oracle Access Manager side:

1. A suitable LDAP repository, such as Oracle Internet Directory (included in the Oracle Application Server 10.1.4 or 10.1.2 infrastructure).
2. A Web server, such as Oracle HTTP Server (included in the Oracle Application Server 10.1.3.x middle-tier infrastructure).
3. The Oracle Access Manager Identity Server and Access Server. When you install Oracle Access Manager, you will be asked to specify the LDAP repository you are using, which must be accessible for communication with Oracle Access Manager Identity Server and Access Server during runtime.
4. Oracle Access Manager WebGate, WebPass, and Policy Manager installed on the Web server. WebGate is the SSO interceptor and communicates with Access Server during runtime. WebPass communicates with Oracle Access Manager Identity Server. Policy Manager communicates with the LDAP repository. When you install WebGate and WebPass, you will be asked to specify the Access Server and Identity Server you are using.

Detailed requirements on the OC4J side:

1. Oracle Application Server 10.1.3.x 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 Oracle Access Manager side, which may or may not be Oracle HTTP Server.

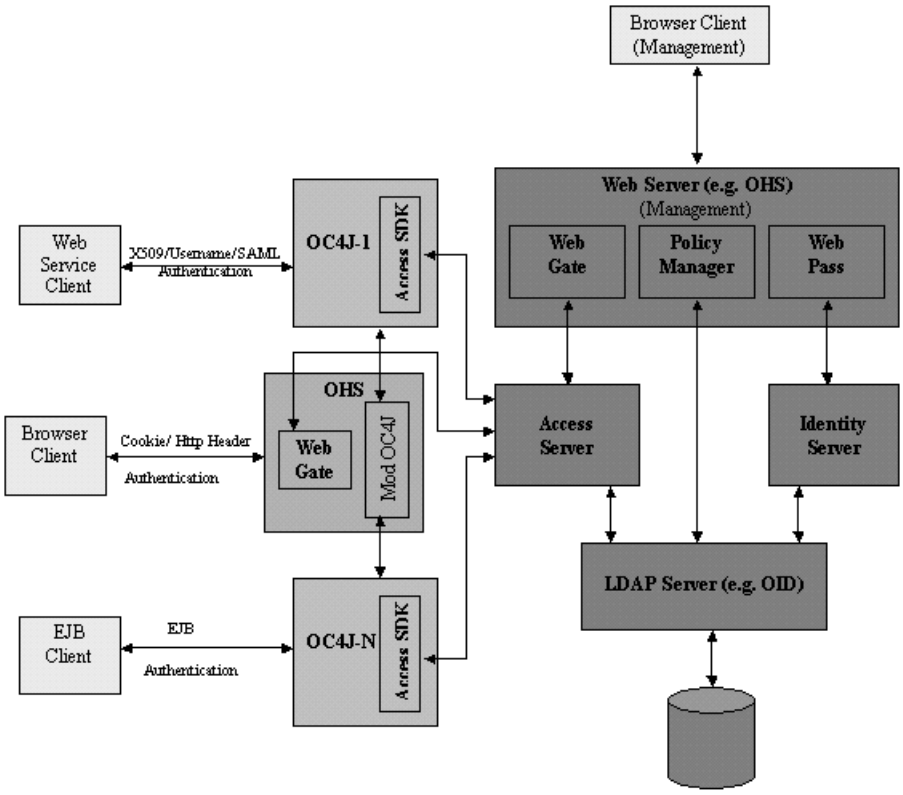
   Using Oracle Access Manager requires Oracle HTTP Server; you cannot use standalone OC4J.
2. Oracle Access Manager WebGate installed on this Oracle HTTP Server.
3. Additional OC4J instances as needed. Typically, when using Oracle Access Manager SSO, multiple OC4J instances are used in the topology, so the Oracle HTTP Server instance must be configured to route and maintain multiple OC4J instances.
4. Access Manager SDK, one for each OC4J instance, on the same system as OC4J. The Access Manager SDK is installed independently and is required by OC4J at runtime to communicate with Access Server. Each OC4J instance communicates with an Access Manager SDK instance, which has been configured to communicate with an Access Server instance.

The next section, "Oracle Access Manager Architecture", shows how the Oracle Access Manager components fit with key components of the Oracle Application Server middle-tier infrastructure.

Oracle Access Manager Architecture

Figure 11–1 shows the Oracle Access Manager architecture.

Figure 11–1 Oracle Access Manager Architecture



Top-Level Summary of Configuration Stages for Oracle Access Manager

There are three stages of configuration steps to use OC4J applications with Oracle Access Manager:

1. One-time configurations for the Oracle Access Manager Policy Manager installation. This includes setting up authentication schemes and configuring an Oracle Access Manager resource type. Refer to "Configuring Oracle Access Manager" on page 11-8.
2. Configurations for each OC4J instance. This includes configuring each OC4J instance with an Access Manager SDK installation. Refer to "Configuring OC4J with the Access Manager SDK" on page 11-14.
3. Configurations for your application. This includes `web.xml` settings, deployment-time settings, `orion-application.xml` settings (pre-deployment or post-deployment), and JAAS login module settings. Refer to "Configuring the Application" on page 11-17.

> **Note:** Also ensure that the LDAP server you use has the accounts you will need, as discussed in "Creating Required Accounts in the LDAP Server" on page 11-16.

Running the Policy Manager

Several of the configuration steps documented later in this chapter involve running the Policy 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 Access Manager Concepts

This section provides background on some Oracle Access Manager concepts that will be relevant later in this chapter:

- About Oracle Access Manager Resource Types
- About Oracle Access Manager Authentication
- About Using HTTP Header Variables for Authentication
- About the Oracle Access Manager Single Sign-On Cookie

About Oracle Access Manager Resource Types

In Oracle Access Manager, 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 Access Manager 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 Manager SDK, and the SDK expects some resource type and resource operation to be specified. For this reason, when you configure the Oracle Access Manager login module, you must configure a custom Oracle Access Manager 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 Access Manager System Administration Guide* for information about Oracle Access Manager resource types

About Oracle Access Manager Authentication

In order to validate any user, Oracle Access Manager must be configured with an authentication scheme. An authentication scheme consists of plug-ins.

OC4J support for Oracle Access Manager uses the `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 Oracle Access Manager:

- Use of the Oracle Access Manager SSO cookie, `ObSSOCookie`, discussed further in the next section, "About the Oracle Access Manager 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 11-7

See Also:

- *Oracle Access Manager System Administration Guide* for information about Oracle Access Manager plug-ins

About the Oracle Access Manager Single Sign-On Cookie

Oracle Access Manager 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, Oracle Access Manager uses the cookie for subsequent requests instead of prompting the user to supply authentication credentials.

OC4J uses the `ObSSOCookie` to connect to the Access Server and retrieve user roles.

Note: `ObSSOCookie` is a session cookie by default, but can be made persistent.

See Also:

- *Oracle Access Manager System Administration Guide* for information about the `ObSSOCookie`

About Using HTTP Header Variables for Authentication

Oracle Access Manager 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 Oracle Access Manager `ObSSOCookie` as discussed in the preceding section.)

To use this mode, you must configure the Oracle Access Manager login module with this user name and password (as discussed in "Configure the Oracle Access Manager Login Module" on page 11-18) for OC4J to use in accessing the 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 Oracle Access Manager and other applications. For example, if Oracle Access Manager and other applications combine to use 1K in the HTTP header, you would have 3K for your data.

Configuring Oracle Access Manager

This section discusses one-time configurations to your Oracle Access Manager installation:

1. Configure Oracle Access Manager Form-Based Authentication
2. Configure Oracle Access Manager Basic Authentication
3. Configure the Resource Type
4. Protect the Action URL

Configure Oracle Access Manager 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 Oracle Access Manager Basic Authentication" on page 11-10.)

The steps here are to create and protect a login page for form-based authentication in Oracle Access Manager, 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 Policy 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 Access Manager System Administration Guide*, particularly the related appendix

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 Policy Manager and related 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 Policy Manager in the next step.

The variable for the user name (`usernamevar` here) must match what you specify in the Oracle Access Manager `credential_mapping` plug-in. The variable for the password (`passwordvar` here) must match what you specify in the Oracle Access Manager `validate_password` plug-in.

Define Form-Based Authentication in Policy Manager

This step uses the Policy Manager to define form-based authentication. Navigate as follows in Policy 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 Oracle Access Manager Basic Authentication", following shortly.

Configure the credential_mapping Plug-In for Form-Based Authentication

Next, you must configure the Oracle Access Manager `credential_mapping` plug-in for form-based authentication in Policy 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 Policy Manager, shown in previous steps.

This also corresponds to the value of the `coreid.name.attribute` option in the Oracle Access Manager login module configuration in OC4J.

See Also:

- *Oracle Access Manager System Administration Guide* for more information about the `credential_mapping` plug-in

Configure the validate_password Plug-In for Form-Based Authentication

Now configure the Oracle Access Manager `validate_password` plug-in for form-based authentication in Policy 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 Policy Manager (shown in previous steps).

This also corresponds to the value of the `coreid.password.attribute` option in the Oracle Access Manager login module configuration.

Configure Oracle Access Manager Basic Authentication

You must configure the Oracle Access Manager 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 11-12.

The Oracle Access Manager login module will use this URL to communicate to the Access Server through the Access Manager SDK.

- The action URL for the form page, noted in "Create a Login Form" on page 11-8 and "Define Form-Based Authentication in Policy Manager" on page 11-9. 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 Oracle Access Manager Form-Based Authentication" on page 11-8.)

These steps define basic authentication, without a password, to protect the resource.

1. Define Basic Authentication in Policy Manager
2. Configure the credential_mapping Plug-In for Basic Authentication

Define Basic Authentication in Policy Manager

This step uses the Policy Manager to configure basic authentication. Navigate as follows in Policy 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 Oracle Access Manager `credential_mapping` plug-in for basic authentication in Policy 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 Oracle Access Manager login module configuration.

See Also:

- *Oracle Access Manager System Administration Guide* for more information about the `credential_mapping` plug-in

Configure the Resource Type

In Oracle Access Manager, 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 Access Manager 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 Policy 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 Oracle Access Manager 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 Access Manager `ObSSOCookie` or the user name, using APIs of the Access Manager 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 Oracle Access Manager Resource Types" on page 11-6

Configure the Name and Operation of the Resource Type

To configure the name and operation of the resource type in Policy 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 Oracle Access Manager 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 Policy Manager, navigate as follows:

Policy 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 Policy Manager" on page 11-11.

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 Oracle Access Manager 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 an Oracle Access Manager "return action" that allows Oracle Access Manager to return the appropriate roles to OC4J for the user after successful authentication.

To set up the return action in Oracle Access Manager, navigate as follows:

Policy 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 Access Manager for use in returning all the roles of an authenticated user.

Protect the Action URL

Using Policy Manager, protect the action URL you specified in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8. 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 11-12.

- This configuration must be under a no-password authentication scheme. Use the basic authentication scheme that you configured in "Configure Oracle Access Manager Basic Authentication" on page 11-10.
- Use "HTTP" as the resource type.

- Specify the action URL (`/coreid/access/test.html` in the examples).

  See Also: For information about protecting resources:

  - *Oracle Access Manager System Administration Guide*

Configuring OC4J with the Access Manager 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 Manager SDK to Each OC4J Instance
3. Configure the Access Manager SDK Library Path for Each OC4J Instance

---

Note: Your middle-tier and OC4J installation can be on the same system as Oracle Access Manager, but would typically not be.

See Also:

- *Oracle Access Manager Installation Guide* for information about installing AccessGate/WebGate

Create OC4J Instances as Needed

Typically, when using Oracle Access Manager 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 Manager SDK to Each OC4J Instance

You will need Oracle Access Manager SDK, one installation for each OC4J instance, on the same system as OC4J. The Access Manager SDK is required by OC4J at runtime to communicate with Access Server. OC4J must be given the Access Manager 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 Oracle Access Manager as the security provider. Also note the following:

1. Create a separate Access Manager SDK installation for each OC4J instance, on the same system as OC4J. You can have multiple Access Manager SDK installations on the same system.

2. Configure each Access Manager SDK to work with the appropriate Access Server. From *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 Oracle Access Manager file jobaccess.jar from the Access Manager 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 Access Manager Developer Guide* for information about installing the Access Manager SDK
- *Oracle Access Manager System Administration Guide* for information about the configureAccessGate utility

Configure the Access Manager 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 Manager 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
```

This is shown in more detail in the next section, "Configuring opmn.xml for Oracle Access Manager".

See Also:

- *Oracle Process Manager and Notification Server Administrator's Guide* for information about OPMN and the opmn.xml file

Configuring opmn.xml for Oracle Access Manager

Where OC4J is managed by OPMN, add settings to opmn.xml for Oracle HTTP Server and OC4J, as follows, when you use Oracle Access Manager:

1. For OC4J, under the process types "home", "OC4J_SOA", and any other OC4J instance where applications will be deployed that use Oracle Access Manager, do the following:
   a. Set the LD_ASSUME_KERNEL environment variable to the value "2.4.19".
   b. Set the LD_LIBRARY_PATH environment variable to point to the AccessServerSDK library path.
   c. Add the AccessServerSDK library path to java.library.path as a start parameter.

   Then restart the OC4J instances.
2. For Oracle HTTP Server, under the process type "HTTP_Server", set LD_ASSUME_KERNEL to "2.4.19", then restart the Oracle HTTP Server instance.

Following is an opmn.xml example for the OC4J home instance. Repeat these settings for the OC4J_SOA instance and any other OC4J instances as appropriate:

```
<ias-component id="OC4J">
   <process-type id="home" module-id="OC4J" status="enabled">
      <environment>
         <variable id="LD_ASSUME_KERNEL" value="2.4.19"/>
         <variable id="LD_LIBRARY_PATH"
                value="/your_asdk_home/AccessServerSDK/oblix/lib" append="true"/>
      </environment>
      <module-data>
         <category id="start-parameters">
            <data id="java-options" value="-server ...
                  -Djava.library.path=/your_asdk_home/AccessServerSDK/oblix/lib
                  ... />
         </category>
         ...
      </module-data>
      ...
   </process-type>
   ...
</ias-component>
```

And here is an example for Oracle HTTP Server:

```
<ias-component id="HTTP_Server">
   <process-type id="HTTP_Server" module-id="OHS">
      <environment>
         <variable id="LD_ASSUME_KERNEL" value="2.4.19" />
      </environment>
      ...
   </process-type>
   ...
</ias-component>
```

Creating Required Accounts in the LDAP Server

In the LDAP directory server that you use (such as Oracle Internet Directory), the following accounts are required by OC4J and Application Server Control 10.1.3.x implementations:

- `oc4jadmin` user
- `oc4j-administrators` role, with member `oc4jadmin`
- `oc4j-app-administrators` role
- `ascontrol_admin` (administrative role for all SOA controls, including Application Server Control), with member `oc4jadmin`
- `ascontrol_appadmin` (Application Server Control required role)
- `ascontrol_monitor` (Application Server Control required role)

If you use Oracle Internet Directory, these accounts are created automatically when you associate the OC4J instance with the Oracle Internet Directory instance, as described in "Associate Oracle Internet Directory with OC4J" on page 8-5. ("Required Accounts Created in Oracle Internet Directory" on page 8-7 is a subsection of this.)

If you use an external LDAP provider, you must create accounts manually, as described in "Creating the Administrative User and Roles and Granting RMI Permission" on page 10-9.

See Also:

- "Predefined Accounts" on page 4-11

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 Oracle Access Manager SSO in orion-application.xml
4. Protect the Application URLs in Oracle Access Manager
5. Configure the Oracle Access Manager 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 Oracle Access Manager configuration in "Protect the Application URLs in Oracle Access Manager" on page 11-18.

Settings for Application Deployment

In Oracle Application Server 10.1.3.x implementations, Application Server Control does not yet support Oracle Access Manager 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 Oracle Access Manager SSO in orion-application.xml

To use Oracle Access Manager 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 Oracle Access Manager

Use Policy 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 11-17. Use the following navigation:

Policy 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 Oracle Access Manager Form-Based Authentication" on page 11-8.

See Also: For information about protecting resources:

- *Oracle Access Manager System Administration Guide*

Configure the Oracle Access Manager Login Module

For a Web application, the OC4J implementation to support Oracle Access Manager 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 11–1 following describes the available options, followed by an example. Additional examples of specific scenarios and their configurations are shown later in this chapter.

Note: By convention, as with other custom login modules, the `<jazn>` setting `provider="XML"` is used with the Oracle Access Manager login module.

See Also:

- Table 9–5, " Login Module Control Flags" on page 9-17 for information about `<control-flag>` settings

```
<application>
   <name>yourappname</name>
   <login-modules>
      <login-module>
         <class>
            oracle.security.jazn.login.module.coreid.CoreIDLoginModule
         </class>
         <control-flag>required</control-flag>
```

```
        <options>
           ...
        </options>
      </login-module>
   </login-modules>
</application>
```

Table 11–1 Oracle Access Manager 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 Policy Manager. **See Also**: "About Oracle Access Manager Resource Types" on page 11-6 and "Configure the Name and Operation of the Resource Type" on page 11-12
coreid.resource.operation	Required	Name of the resource operation associated with the resource type specified in `coreid.resource.type`, as defined through Policy Manager. **See Also**: "Configure the Name and Operation of the Resource Type" on page 11-12
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 Policy Manager. **See Also**: "Configure and Protect the URL of the Configured Resource Type" on page 11-12
coreid.name.attribute	Required	Variable for the user name for authentication, as defined in the `credential_mapping` plug-in. **See Also**: "About Oracle Access Manager Authentication" on page 11-6 and "Configure the credential_mapping Plug-In for Form-Based Authentication" on page 11-10
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 11-10

Table 11–1 (Cont.) Oracle Access Manager Login Module Options

Option Name	Required/Optional	Option Value
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 Oracle Access Manager Access Server. **See Also**: "About Using HTTP Header Variables for Authentication" on page 11-7 and "Web Application Using HTTP Header Variables through Oracle Access Manager" on page 11-26
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 Access Server.

Note: The values of `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name` are determined during one-time Oracle Access Manager configuration, as described in "Configure the Resource Type" on page 11-12, and are the same for any application using the same installation of Oracle Access Manager. Each application must have appropriate settings for these property values in its configuration for the Oracle Access Manager login module, which you can accomplish using Application Server Control or the OracleAS JAAS Provider Admintool.

The following sample corresponds to the example that runs throughout "Configure Oracle Access Manager Form-Based Authentication" on page 11-8, "Configure Oracle Access Manager Basic Authentication" on page 11-10, and "Configure the Resource Type" on page 11-12:

```
<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 Oracle Access Manager 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 11-8. 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 LDAP repository that you use).
3. After the user is authenticated, the Oracle Access Manager 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 Manager SDK configured on OC4J.

Granting Permissions to Oracle Access Manager Principals

You must grant any necessary permissions to any Oracle Access Manager principals that require privileges in your application. For an EJB application, this includes granting `RMIPermission` "login" for EJB access.

This section covers the following topics:

- Granting RMI Permission to an Oracle Access Manager Principal
- Granting Required Permissions to Additional Oracle Access Manager Principals

- Confirming Configured Realm Names for Oracle Access Manager Principals

Granting RMI Permission to an Oracle Access Manager Principal

When using Oracle Access Manager for an EJB application, it is necessary to grant RMI permission "login" to an Oracle Access Manager principal for EJB access.

The following example uses the OracleAS JAAS Provider Admintool to accomplish this, assuming the principal name `orcladmin`:

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.CoreIDPrincipal \
       orcladmin 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.CoreIDPrincipal</class>
               <name>orcladmin</name>
            </principal>
         </principals>
      </grantee>
      ...
      <permissions>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

Important: Also refer to "Confirming Configured Realm Names for Oracle Access Manager Principals" on page 11-24.

Granting Required Permissions to Additional Oracle Access Manager Principals

When using Oracle Access Manager, authentication occurs at the Oracle Access Manager end, but JAAS authorization occurs at the OC4J end. (Other levels of authorization may occur at the Oracle Access Manager end.) For JAAS authorization in your application to be successful, the appropriate permissions must be granted to any Oracle Access Manager principals that are populated into your application subjects after authentication, and these grants must be stored in the `system-jazn-data.xml` file.

For this discussion, assume a principal `BPMSystemAdmin` requires the `ServerPermission` "server". The following example uses the OracleAS JAAS Provider Admintool to accomplish this:

```
% java -jar jazn.jar -grantperm oracle.security.jazn.realm.CoreIDPrincipal \
       BPMSystemAdmin com.collaxa.security.ServerPermission server
```

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.CoreIDPrincipal</class>
               <name>BPMSystemAdmin</name>
            </principal>
         </principals>
      </grantee>
      ...
      <permissions>
         <permission>
            <class>com.collaxa.security.ServerPermission</class>
            <name>server</name>
            <actions>all</actions>
         </permission>
         ...
      </permissions>
      ...
   </grant>
   ...
</jazn-policy>
```

Important:

- Ensure that the permission class is in the classpath before you attempt to grant the permission.
- Also refer to "Confirming Configured Realm Names for Oracle Access Manager Principals" on page 11-24.

Here is a sample configuration for the `BPMSystemAdmin` role:

```
<role>
    <name>BPMSystemAdmin</name>
    <guid>3E9D3A5037A311DBBFA2B1BC62ED9FBC</guid>
    <members>
        <member>
            <type>user</type>
            <name>bpeladmin</name>
        </member>
        <member>
            <type>user</type>
            <name>oc4jadmin</name>
        </member>
    </members>
</role>
```

Notes:

- The use of OC4J JAAS mode is supported for an application that uses Oracle Access Manager, in case that is required by your application in checking authorizations with respect to the permissions you have granted. Refer to "Introduction to JAAS Mode" on page 5-5 and "Configuring and Using JAAS Mode" on page 5-18 to gain an understanding of when and how to use this mode.
- For authorization to work properly, also confirm that role mapping is set up appropriately, to correctly map deployment roles to J2EE logical roles. Refer to "Mapping Security Roles" on page 6-10 for additional information.

Confirming Configured Realm Names for Oracle Access Manager Principals

In permissions configuration for Oracle Access Manager principals, each configured principal name must exactly match the principal name, including any realm name, as it comes over from Oracle Access Manager when the principal is populated into a subject.

For example, if `BPMSystemAdmin` is in the `abc` realm in Oracle Access Manager, then the principal name in `system-jazn-data.xml` must be exactly as follows:

```
<grantee>
   <principals>
      <principal>
         <class>oracle.security.jazn.realm.CoreIDPrincipal</class>
         <name>abc/BPMSystemAdmin</name>
      </principal>
   </principals>
</grantee>
```

Important:

- When using Oracle Access Manager, do not attempt to use the Admintool `addrealm` option, which would result in incorrect realm information in the configuration. (This command is intended for only the file-based provider.)
- If you encounter any difficulty, ensure that only the appropriate realm information is included in any principal names for permission grants in `system-jazn-data.xml`.

Considerations for Oracle Application Server SOA Applications

This section discusses special considerations when you use Oracle Access Manager to protect Oracle Application Server SOA applications such as Application Server Control and OWSM. The following topics are covered:

- Configure Logout for Oracle Application Server SOA Applications
- Troubleshooting Login to Oracle Application Server SOA Applications

Configure Logout for Oracle Application Server SOA Applications

For logout to work properly for Oracle Application Server SOA applications protected by Oracle Access Manager, complete the following steps (assuming Oracle HTTP Server is the Web server for Policy Manager):

1. Create a shared logout page for all the applications. Assuming the logout page is `logout.html`, you can accomplish this by copying `logout.html` to the Oracle HTTP Server `Apache/Apache/htdocs` directory.
2. Configure SSO logout to use the logout page "`/logout.html`". This registers this URL as the logout URL with Policy Manager. To accomplish this, navigate as follows in Policy Manager:

   Access System Console > Access System Configuration > AccessGate Configuration > WebGate Configuration

   Set `LogOutURLs` to: `/logout.html`
3. Make sure the logout page is not protected by WebGate, or is protected using the "none" authentication scheme.
4. Restart the Oracle HTTP Server instance that Policy Manager uses.
5. In the `<jazn>` element of the OC4J `jazn.xml` file, set the property `custom.sso.url.logout` to point to the logout page URL, such as:

   ```
   <jazn ... >
      <property name="custom.sso.url.logout" value="/logout.html" />
      ...
   </jazn>
   ```
6. Restart the OC4J instance.

Also ensure that the login page and logout page are in the same cookie domain, or that the cookies set during login and logout map to a shared domain.

Troubleshooting Login to Oracle Application Server SOA Applications

If you try to log in to an Oracle Application Server SOA application (for example, using `http://www.example.com:7778/ccore/index.html` for OWSM), and the login hangs on the form login page after you enter your credentials, one possible cause is that there is a time-synch mismatch between the server running the SOA application (such as OWSM) and the server running Oracle Access Manager. In this case, WebGate will fail to successfully create a session for the user. If you experience this, the administrator should confirm that both systems are synchronized.

Oracle Access Manager Examples for J2EE Applications

This section discusses the following Oracle Access Manager usages for Web applications and EJBs:

- Web Application Using HTTP Header Variables through Oracle Access Manager
- Web Application Using the Oracle Access Manager ObSSOCookie
- EJB Application Using Oracle Access Manager

See Also:

- "Oracle Access Manager Support and Examples for Web Services" on page 11-29

Web Application Using HTTP Header Variables through Oracle Access Manager

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 Oracle Access Manager 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 Name and Password in Policy Manager
2. Configure HTTP Header Variables for the Oracle Access Manager Login Module
3. Secure the Web Application That Uses HTTP Headers

See Also:

- "About Using HTTP Header Variables for Authentication" on page 11-7

Configure Name and Password in Policy Manager

Use Policy Manager to enable the `credential_mapping` and `validate_password` plug-ins.

See Also:

- "Configure the credential_mapping Plug-In for Form-Based Authentication" on page 11-10 and "Configure the validate_password Plug-In for Form-Based Authentication" on page 11-10
- *Oracle Access Manager System Administration Guide* for information about using HTTP header variables

Configure HTTP Header Variables for the Oracle Access Manager Login Module

Include option settings for `coreid.name.header` and (as appropriate) `coreid.password.header` in the Oracle Access Manager login module configuration in `system-jazn-data.xml`. In the following example, password authentication is used. Assume the desired HTTP header variables 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 That Uses HTTP Headers

Define appropriate security constraints in your standard Web application configuration, and set `auth-method="COREIDSSO"` in `orion-application.xml` as shown in "Configure Oracle Access Manager SSO in orion-application.xml" on page 11-17.

Web Application Using the Oracle Access Manager ObSSOCookie

When no HTTP header variables are provided for a secure Web application, the Oracle Access Manager `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 Oracle Access Manager Login Module
2. Secure the Web Application That Uses ObSSOCookie

Configure User Name and Password for the Oracle Access Manager Login Module

Include option settings for `coreid.name.attribute` and (as appropriate) `coreid.password.attribute` in the Oracle Access Manager login module configuration in `system-jazn-data.xml`. In the following example, password authentication is used. Assume 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 That Uses ObSSOCookie

Define appropriate security constraints in your standard Web application configuration, and set `auth-method="COREIDSSO"` in `orion-application.xml` as shown in "Configure Oracle Access Manager SSO in orion-application.xml" on page 11-17.

EJB Application Using Oracle Access Manager

For EJB authentication, OC4J gets the user name and password from the EJB context and passes them to the Oracle Access Manager login module. The same user name and password are used to authenticate against Oracle Access Manager.

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 Oracle Access Manager login module, as discussed in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8.

The client must send the user name and password for authenticating itself before it can access the EJB.

Configure the Oracle Access Manager login module. Assume Oracle Access Manager 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 Oracle Access Manager `ObSSOCookie` is sent instead of the user name and password for authentication.

See Also:

- "Configure the Resource Type" on page 11-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`

Oracle Access Manager Support and Examples for Web Services

Web services can use Oracle Access Manager for authenticating Web service clients. With respect to Oracle Access Manager, 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 Oracle Access Manager.
- X.509 token: OC4J uses the CN value of the X.509 entry to authenticate against Oracle Access Manager.
- SAML token: OC4J uses the subject name to authenticate against Oracle Access Manager.

The following usages are shown below:

- Web Service with Username Token Authentication for Oracle Access Manager
- Web Service with X.509 Token Authentication for Oracle Access Manager
- Web Service with SAML Token Authentication for Oracle Access Manager

Note: In the current release there is no direct support for a scenario where the Oracle Access Manager `ObSSOCookie` is sent instead of the user name and password for authentication.

See Also:

- "Oracle Access Manager Examples for J2EE Applications" on page 11-25
- *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 Oracle Access Manager

A username token client uses the user name and password for authentication. You must configure variables for the user name and password through the Oracle Access Manager `credential_mapping` and `validate_password` plug-ins, with corresponding settings for the `coreid.name.attribute` and `coreid.password.attribute` options in the Oracle Access Manager login module

configuration, as discussed in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8.

Configure the login module as follows, assuming these settings:

- `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 11-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`

Web Service with X.509 Token Authentication for Oracle Access Manager

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 Oracle Access Manager `credential_mapping` plug-in, with a corresponding setting for the `coreid.name.attribute` option in the Oracle Access Manager login module configuration, as discussed in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8.

Do not configure the Oracle Access Manager `validate_password` plug-in or set the login module `coreid.password.attribute` option when X.509 token authentication is used.

Configure the login module as follows, assuming these settings:

- `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 11-12 for information about `coreid.resource.type`, `coreid.resource.operation`, and `coreid.resource.name`

Web Service with SAML Token Authentication for Oracle Access Manager

For a SAML client, OC4J determines the subject name, and you must configure a variable name for SAML subject authentication through the Oracle Access Manager `credential_mapping` plug-in. This `credential_mapping` setting must be reflected in the setting of the `coreid.name.attribute` option in the Oracle Access Manager login module configuration, as discussed in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8. OC4J passes the subject name and `credential_mapping` variable name to Oracle Access Manager for authentication.

Do not configure the Oracle Access Manager `validate_password` plug-in or set the login module `coreid.password.attribute` option when SAML authentication is used.

Configure the login module as shown below, assuming these settings:

- `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 `CoreIDLoginModule` 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 11-12 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 Access Manager Setup

Table 11–2 provides some troubleshooting tips for your Oracle Access Manager setup and configuration.

Table 11–2 Oracle Access Manager Troubleshooting

Problem	Cause/Solution
The application is configured to use Oracle Access Manager 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 Oracle Access Manager SSO application, it throws a Class Not Found exception.	Confirm you copied the Oracle Access Manager file `jobaccess.jar` from the Access Manager SDK to the OC4J path, as described in "Configure the Access Manager SDK to Each OC4J Instance" on page 11-14.
When you try to access the Oracle Access Manager SSO application, it gives an internal server error.	Confirm that the Access Manager SDK installed on the OC4J server is configured to use the appropriate Access Server, as discussed in "Configure the Access Manager SDK to Each OC4J Instance" on page 11-14. Also confirm that OC4J is running.
When you try to access the Oracle Access Manager SSO application, it does not appear in the login page.	Confirm you have enabled your authentication scheme with proper settings, using Policy Manager, as discussed in "Configure Oracle Access Manager Form-Based Authentication" on page 11-8.

Table 11–2 (Cont.) Oracle Access Manager Troubleshooting

Problem	Cause/Solution
When you try to access the Oracle Access Manager SSO application, the login page keeps coming back.	Confirm that the form-based authentication scheme is enabled, that the form variable names (for user and password) in your login page are the same as you configured in the Oracle Access Manager 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 Oracle Access Manager Form-Based Authentication" on page 11-8.
You have configured the application to use Oracle Access Manager, but you always get an "unauthorized" or "unauthenticated" error.	Confirm that the Oracle Access Manager login module is correctly configured for this application in `system-jazn-data.xml`. Refer to "Configure the Oracle Access Manager Login Module" on page 11-18.
You have configured the application to use Oracle Access Manager, but you get an internal server error.	Confirm that the LDAP server (Oracle Internet Directory, for example) that is configured with the Oracle Access Manager Identity Server is running and accessible.
You have configured the application to use Oracle Access Manager 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 11-8.

See Also:

- "Troubleshooting Login to Oracle Application Server SOA Applications" on page 11-25

12

User and Role API Framework

OC4J 10.1.3.x implementations provide a new pluggable identity management API framework for accessing user and role information from disparate identity management repositories. To avoid confusion with the pluggable identity management framework discussed in Chapter 13, which is independent, we will simply refer to the APIs discussed here as the "user and role APIs".

These APIs include functionality to replace the deprecated `UserManager`, `User`, and `Group` classes of the `com.evermind.security` package.

This chapter covers the following topics:

Overview of User and Role (Identity Management) API Framework

The user and role API framework allows applications to access identity information (users and roles) in a uniform and portable manner regardless of the particular underlying identity repository. The underlying repository could be an LDAP directory server such as Oracle Internet Directory, Active Directory (from Microsoft), or Sun Java System Directory Server (from Sun Microsystems), or could be a database, flat file, or some other custom repository.

This API framework provides a convenient way to access repositories programmatically in a portable way, freeing the application developer from the potentially difficult task of accounting for the intricacies of particular identity sources. The framework allows an application to work against different repositories seamlessly. An application can switch between various identity repositories without any code changes being required.

Supported operations include creating, updating, or deleting users and roles, or searching users and roles for attributes or information of interest. For example, you may want to search for the e-mail addresses of all users in a certain role. These APIs are *not* for authentication or authorization functions.

You can use a basic usage model (without OC4J integration) or a usage model with OC4J integration that allows your code to be portable.

User and Role API Features to Replace UserManager, User, Group

The user and role APIs include features to replace functionality of the deprecated `com.evermind.security.UserManager`, `User`, and `Group` classes, as follows:

- User and role API features to create, modify, or retrieve users and roles in the identity repository replace `UserManager` functionality to create and retrieve users and roles.
- User and role API features to retrieve or modify user roles (such as assigning a user to a role) replace `User` functionality to retrieve or modify user roles. (Authentication features of the `User` class are replaced by `DBTableOraDataSourceLoginModule`, discussed in "DBTableOraDataSourceLoginModule" on page 9-5.)
- User and role API features to retrieve a role replace `Group` functionality to retrieve a role. (Features of the `Group` class to grant permissions to a role are replaced by functionality of the OracleAS JAAS Provider policy management APIs, discussed in "OracleAS JAAS Provider APIs for Granting or Revoking Permissions" on page 5-7.)

User and Role API Framework and Providers

The user and role APIs are based on a framework and provider model similar to JNDI.

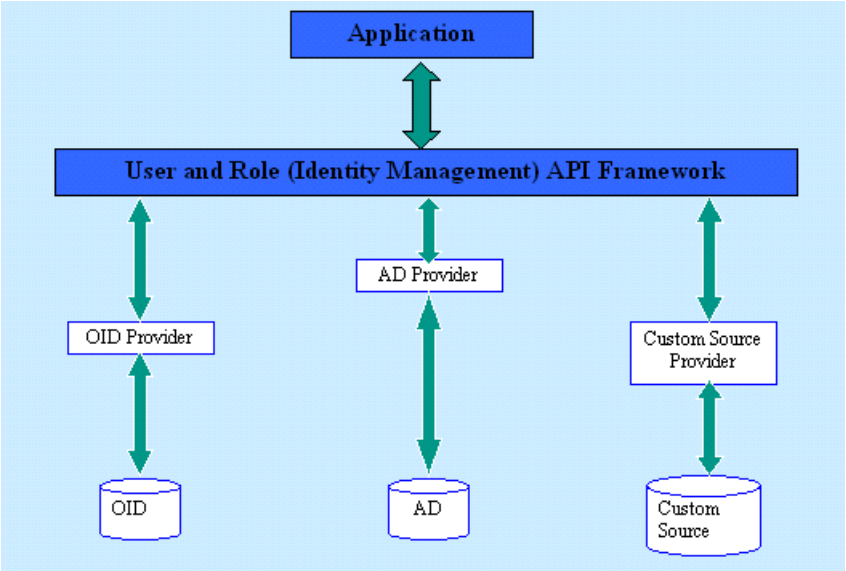
The framework specifies a generic mechanism for accessing identity information, and consists only of Java interfaces. It does not provide implementation details.

The "providers" (included with OC4J) each implement the framework interfaces, with specific implementation classes for accessing information from particular repositories—for example, to read identity information from Active Directory. Each type of identity repository has a corresponding provider.

The application developer creates code based on the generic framework. Later, during a configuration step, the application is plugged in to the appropriate provider for the desired identity repository. Subsequently, the application can be updated to use a different repository by simply reconfiguring it to use the corresponding provider. No changes are required to the application code.

OC4J comes with providers for Oracle Internet Directory, Active Directory, Sun Java System Directory Server (formerly iPlanet), Novell eDirectory, and OpenLDAP (an open source LDAP directory). Figure 12–1 following depicts the framework with respect to a few particular providers.

Figure 12–1 User and Role API Framework Model



Summary of User and Role Interfaces and Classes

This section summarizes interfaces and classes of the user and role API package, `oracle.security.idm`.

See Also:

- *Oracle Containers for J2EE User and Role Java API Reference* (Javadoc)

User and Role Interface Descriptions

This section summarizes interfaces in the `oracle.security.idm` package.

There are the following interfaces for identity repositories:

- `IdentityStore`: An `IdentityStore` instance represents a handle to an identity repository. Methods are specified for the following functions:
  - Get a user manager or role manager.
  - Search for users, roles, or their profiles.
  - Get a search filter or the list of searchable attributes for the identity repository.
- `IdentityStoreFactory`: An `IdentityStoreFactory` instance represents the underlying identity repository, and includes a method to get an `IdentityStore` instance, taking as input a hashtable consisting of properties specific to the provider that are required in order to create the instance.

There are the following interfaces for user entries in the identity repository:

- `User`: A `User` instance represents a user in the identity store. This is a subinterface of `Identity`, and specifies a method to get a user profile. It is also a superinterface of `UserProfile`.
- `UserProfile`: A `UserProfile` instance represents detailed information about a user and includes constants for commonly accessed properties such as name, title, employee number, manager, postal address, e-mail address, phone number, fax number, wireless number, and many more. This is a subinterface of `User`. It

specifies methods to get and set any of these common properties, as well as the more general methods `getProperty()`, `getProperties()`, `setProperty()`, and `setProperties()`. The `setProperty()` and `setProperties()` methods take instances of the `oracle.security.idm.ModProperty` class (described below).

- `UserManager`: A `UserManager` instance is used to manage the user population within the repository, including the execution of operations involving users. This includes creating, authenticating, or dropping a user.

There are the following interfaces for roles in the identity repository:

- `Role`: A `Role` instance represents a role in the identity store. This is a subinterface of `Identity`, and specifies a method to get a role profile. It is also a superinterface of `RoleProfile`.
- `RoleProfile`: A `RoleProfile` instance represents detailed information about a role. This is a subinterface of `Role`, and specifies methods to add, remove, or get owners of the role; get all the grantees that are directly or indirectly granted the role; and determine whether this is an application role or an enterprise role.
- `RoleManager`: A `RoleManager` instance is used to manage the role population within the repository, including the execution of operations involving roles. This includes creating a role, dropping a role, granting a role to a specified principal, or revoking a role from a specified principal.

User and Role Class Descriptions

This section summarizes a key class in the `oracle.security.idm` package.

- `IdentityStoreFactoryBuilder`: Use an instance of this class to build an identity store factory. It includes the overloaded `getIdentityStoreFactory()` method to get an `IdentityStoreFactory` instance.

User and Role API Usage Models

This section supplies step-by-step instructions and samples for using the basic API framework and OC4J integration features, covering the following topics:

- Step by Step: Basic Usage Model
- Step by Step: OC4J Integration Usage Model
- Permission Requirements for the OC4J Integration Feature
- User and Role Properties File

Step by Step: Basic Usage Model

This section describes steps to use the basic API framework. Steps 1 and 2 are primarily related to configuration, determining the provider to be used and its configuration. The code in these two steps would change for a different identity repository and provider. The operations performed on the repository in step 3 are generic in nature. Changing to a different repository would not affect this code.

See Also:

- "Example: Basic User and Role API Framework" on page 12-8 for the complete sample code
- "Step by Step: OC4J Integration Usage Model" on page 12-6 for features that allow you to make your code portable

1. Obtain an `IdentityStoreFactory` instance. This factory instance will represent the identity repository and is created using a `getIdentityStoreFactory()` call on an `IdentityStoreFactoryBuilder` instance. This call accepts the name of the provider to be used for connecting to the particular identity repository. It also accepts any configuration information required by the provider.

   For example, assume the application must connect to Oracle Internet Directory. The following is the provider name for Oracle Internet Directory:

   ```
   oracle.security.idm.providers.oid.OIDIdentityStoreFactory
   ```

   An LDAP provider, such as Oracle Internet Directory, will require configuration information including the LDAP URL, security principal, and credentials. For example:

   ```
   IdentityStoreFactoryBuilder builder =
                              new IdentityStoreFactoryBuilder();
   IdentityStoreFactory oidFactory = null;

   Hashtable factEnv = new Hashtable();

   // creating the factory instance
   // set the configuration information
   factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_PRINCIPAL, "cn=orcladmin");
   factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS, "welcome1");
   factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
            "ldap://ilinabc10.us.oracle.com:3060/");
   factEnv.put(OIDIdentityStoreFactory.ST_LOGGING,"false");
   factEnv.put(OIDIdentityStoreFactory.ST_LOG_LEVEL,
           java.util.logging.Level.ALL);
   oidFactory =  builder.getIdentityStoreFactory(
           "oracle.security.idm.providers.oid.OIDIdentityStoreFactory", factEnv);
   ```

2. Obtain an `IdentityStore` instance to carry out operations on the repository. This is obtained using a `getIdentityStoreInstance()` call on the `IdentityStoreFactory` instance. This call can accept configuration information required for creation of the identity store instance. For example, for Oracle Internet Directory you must specify the subscriber or realm name upon which the operations are to be performed, as shown in the following sample:

   ```
   Hashtable storeEnv = new Hashtable();

   // creating the store instance
   storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                                "dc=us,dc=oracle,dc=com");
   oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
   ```

3. Using the `IdentityStore` instance, perform any operations of interest on the identity repository, such as searching, updating, creating, or deleting entries. For example, the following code will search for all users whose name begins with "`john`":

   ```
   // search filter for users whose name begins with "john"
   ```

```
SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
        UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);

// Add the wildcard character
sf.setValue("john"+sf.getWildCardChar());

// generate the search parameter instance and set the search filter
SearchParameters params = new SearchParameters();
params.setFilter(sf);

// Searching for users
// search on the IdentityStore instance

SearchResponse resp = oidStore.searchUsers(params);
System.out.println("Searched users are:");

// Iterate on the search results
while (resp.hasNext()
{
  User usr = (User) resp.next();
  System.out.println("Name: "+usr.getName());
}
```

Step by Step: OC4J Integration Usage Model

As explained in the preceding section, "Step by Step: Basic Usage Model", the configuration-related code is tied to the identity repository and its associated provider, and thus is subject to change whenever the application changes to use a different identity repository.

To make your code portable, you can use the OC4J integration feature of the API framework. This feature uses security provider information present in the OC4J login module of the application, so that the application itself need not specify any configuration for the repository and provider. The application code becomes generic, and the application can change to a different identity source simply by changing the security provider information in the login module configuration.

Important: The OC4J integration feature is a security-sensitive operation and requires the application to have necessary permissions. See the next section, "Permission Requirements for the OC4J Integration Feature", for `java2.policy` requirements.

See Also:

- "Example: OC4J Integration with User and Role API Framework" on page 12-9 for the complete sample code

Use the OC4J integration feature as follows:

1. The user and role framework requires a path to a user/role properties file (typically called `userrole.properties` by convention), specified as a Java system property, as follows:

```
System.setProperty("oracle.userrole.properties",
                   "/home/jdoe/userrole.properties");
```

The properties file contains settings required by the framework to access OC4J login module information. The properties file format is shown in "User and Role Properties File" on page 12-8.

2. Creating the `IdentityStoreFactory` instance is a privileged operation and must be carried out within an `AccessController.doPrivileged()` block, as follows:

```
IdentityStoreFactory factory = null;

try
{
  factory = (IdentityStoreFactory) AccessController.doPrivileged(
                new PrivilegedExceptionAction()
                {
                  public Object run() throws IMException
                  {
                    IdentityStoreFactoryBuilder builder =
                                new IdentityStoreFactoryBuilder();
                    return builder.getIdentityStoreFactory();
                  }
                });

}catch (PrivilegedActionException e)
{
  e.getException().printStackTrace(out);
}
catch (Exception e)
{
  e.printStackTrace(out);
}
```

3. Obtaining the `IdentityStore` instance is convenient:

```
IdentityStore store = factory.getIdentityStoreInstance();
```

4. Perform operations on the identity store as shown in the previous section,"Step by Step: Basic Usage Model".

Permission Requirements for the OC4J Integration Feature

Because the OC4J integration feature is a security-sensitive operation, the application code must have certain permissions. In particular, the `getIdentityStoreFactory()` method of the `IdentityStoreFactoryBuilder` class makes API calls that require certain permissions.

In the OC4J `java2.policy` file, grant the following permissions to the application codebase:

```
grant codebase "file:${oracle.home}/application_code_base"
{
    permission oracle.security.jazn.JAZNPermission "*";
};
```

Be aware that in order to use Java 2 policies, you must specifically enable a security manager, as discussed in "Specifying a Java 2 Security Manager and Policy File" on page 5-1.

User and Role Properties File

To use the OC4J integration feature, the API framework must know the path to the user/role properties file, expressed as a Java system property. The properties file contains the properties required by the framework to access required OC4J login module information. The file can have any name you choose, but must have the following file format:

```
# This line should not be changed.
configurationsourceclass=oracle.security.idm.util.OC4JConfigurationSource

# This property specifies the JMX Mbean URL for the OC4J container in which the
# application is deployed.
# For OPMN-managed OC4J, uncomment the URL that follows; comment out all others.
# format: service:jmx:rmi:///opmn://opmnhost[:opmnport]/oc4jInstance

jmxserviceurl=service:jmx:rmi:///opmn://localhost:6008/home

# For standalone OC4J, uncomment the URL that follows; comment out all others.
# format: service:jmx:rmi:///opmn://oc4jhost:rmiport/oc4jContextRoot
#jmxserviceurl=service:jmx:rmi://localhost:23791/oc4j/
```

See Also:

- *Oracle Containers for J2EE Developer's Guide* for information about setting the JMX Service URI for an OPMN-managed or standalone OC4J instance

Example: Basic User and Role API Framework

This section supplies the complete example corresponding to the steps discussed in "Step by Step: Basic Usage Model" on page 12-4.

```
import oracle.security.idm.*;
import oracle.security.idm.providers.oid.*;
import java.util.*;
import java.io.*;

public class BasicSampleOID
{
  public static void main(String args[])
  {
    IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();
    IdentityStoreFactory oidFactory = null;
    IdentityStore oidStore = null;

    try
    {

      Hashtable factEnv = new Hashtable();
      Hashtable storeEnv = new Hashtable();

      // creating the factory instance
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_PRINCIPAL,
                  "cn=user,....");
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
                   "password");
      factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
                  "ldap://johnmc.us.oracle.com:3060/");
      factEnv.put(OIDIdentityStoreFactory.ST_LOGGING,"false");
      factEnv.put(OIDIdentityStoreFactory.ST_LOG_LEVEL,
```

```
                java.util.logging.Level.ALL);

      oidFactory =  builder.getIdentityStoreFactory(
                "oracle.security.idm.providers.oid.OIDIdentityStoreFactory",
                factEnv);

      // creating the store instance
      storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                  "dc=us,dc=oracle,dc=com");
      oidStore = oidFactory.getIdentityStoreInstance(storeEnv);

      // search filter (cn=a*)
      SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
                   UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
      sf.setValue("john"+sf.getWildCardChar());

      SearchParameters params = new SearchParameters();
      params.setFilter(sf);

      // Searching for users
      SearchResponse resp = oidStore.searchUsers(params);
      System.out.println("Searched users are:");
      while (resp.hasNext())
      {
        User usr = (User) resp.next();
        System.out.println("Name: "+usr.getName());
      }

    }catch (IMException e)
    {
      e.printStackTrace();
    }
  }
}
```

Example: OC4J Integration with User and Role API Framework

This section supplies the complete example corresponding to the steps discussed in "Step by Step: OC4J Integration Usage Model" on page 12-6.

```
import java.io.*;
import java.util.*;
import java.security.AccessController;
import java.security.PrivilegedExceptionAction;
import java.security.PrivilegedActionException;

// Packages for Servlets
import javax.servlet.*;
import javax.servlet.http.*;
import oracle.security.idm.*;

public class UserSearch extends HttpServlet
{
    private String USERROLEPROPFILE = "UserRolePropFile";
    IdentityStore store = null;

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }
```

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
        throws ServletException, IOException {
    String name = "";
    String searchType = "usersearch";
    try
    {
        name = request.getParameter("name");
        searchType = request.getParameter("searchtype");
    } catch (Exception e)
    {
        e.printStackTrace();
    }

    String filter = (name != null)? name: "";

    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    out.println("<html>");
    out.println("<head><title>UserSearch</title></head>");
    out.println("<body>");
    out.println(
        "<label for=\"fld\"><b>User name begining with</b></label>");
    out.println(
      "<form action=\"usersearch\">"+
      "<P><select name=\"searchtype\" size=\"1\">"+
      "<option value=\"usersearch\">Search Users</option>"+
      "<option value=\"rolesearch\">Search Roles</option>"+
      "<option value=\"membershipsearch\">Search Membership details"+
      "<option value=\"membersearch\">Searchs Members of role"+
      "</option></select></P>"+
      "<input name=\"name\" id=\"fld\" value=\""
      + filter +
      "\"type=\"text\"/><input type=\"SUBMIT\" value=\"Search\"/></form>");
    out.println("<br><br><br>");

    // Create the IdentityStore instance required for searching
    configureAPI(out);

    // Carries out the actual search using IdentityStore instance obtained
    // above
    doSearch(out, searchType, name);

    out.println("</body></html>");
    out.close();
}

public void configureAPI(PrintWriter out)
{
  IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();

  // Set the following system property to specify the location of
  // "userrole.properties" file. This file is used by user-role apis for
  // reading the configuration from OC4J

  // Get the file location from the servlet init parameters
  System.setProperty("oracle.userrole.properties",
            getServletConfig().getInitParameter(USERROLEPROPFILE));

  IdentityStoreFactory factory = null;
```

```
  try
  {
    factory = (IdentityStoreFactory) AccessController.doPrivileged(
             new PrivilegedExceptionAction()
             {
               public Object run() throws IMException
               {
                 IdentityStoreFactoryBuilder builder =
                             new IdentityStoreFactoryBuilder();
                 return builder.getIdentityStoreFactory();
               }
             });
    store = factory.getIdentityStoreInstance();
  }catch (PrivilegedActionException e)
  {
    e.getException().printStackTrace(out);
  }
  catch (Exception e)
  {
    e.printStackTrace(out);
  }
}

public void doSearch(PrintWriter out, String searchType, String name)
{
    System.out.println("Inside doSearch");

    if (name == null) return;
    if (searchType.equals("usersearch"))
      searchUsers(out, name);
    else if (searchType.equals("rolesearch"))
      searchRoles(out, name);
    else if (searchType.equals("membershipsearch"))
      searchMembership(out, name);
    else if (searchType.equals("membersearch"))
      searchMembers(out, name);
}

public void searchMembers(PrintWriter out, String name)
{
    System.out.println("Inside searchMembers");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        Role rle = store.searchRole(IdentityStore.SEARCH_BY_NAME, name);
        out.println("Members of role \""+rle.getName()+
                    "\" are:<br>");
        SearchResponse resp =
         rle.getRoleProfile().getGrantees(null, false);
        while (resp.hasNext()) {
            Identity idy = resp.next();
            out.println("Unique name: " + idy.getUniqueName() + "<br>");
        }

    } catch (IMException e) {
        e.printStackTrace(out);
    }
}
```

```
public void searchMembership(PrintWriter out, String name)
{
    System.out.println("Inside searchMembership");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        User usr = store.searchUser(name);
        out.println("Membership details for user \""+usr.getName()+
                    "\" are:<br>");
        SearchResponse resp =
         store.getRoleManager().getGrantedRoles(usr.getPrincipal(), false);
        while (resp.hasNext()) {
            Identity idy = resp.next();
            out.println("Unique name: " + idy.getUniqueName() + "<br>");
        }

    } catch (IMException e) {
        e.printStackTrace(out);
    }
}

public void searchRoles(PrintWriter out, String name)
{
    System.out.println("Inside searchRoles");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        SimpleSearchFilter sf = store.getSimpleSearchFilter(
                RoleProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
        sf.setValue(name + sf.getWildCardChar());

        SearchParameters params = new SearchParameters();
        params.setFilter(sf);

        // Searching for users
        SearchResponse resp = store.searchRoles(0, params);
        out.println("Searched roles are:<br>");
        while (resp.hasNext()) {
            Identity idy = resp.next();
            out.println("Unique name: " + idy.getUniqueName() + "<br>");
        }

    } catch (IMException e) {
        e.printStackTrace(out);
    }
}

public void searchUsers(PrintWriter out, String name)
{
    System.out.println("Inside searchUsers");
    if (name == null) return;
    out.println("Results: <br>");

    try {
        SimpleSearchFilter sf = store.getSimpleSearchFilter(
                UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
        sf.setValue(name + sf.getWildCardChar());
```

```
            SearchParameters params = new SearchParameters();
            params.setFilter(sf);

            // Searching for users
            SearchResponse resp = store.searchUsers(params);
            out.println("Searched users are:<br>");
            while (resp.hasNext()) {
                Identity idy = resp.next();
                out.println("Unique name: " + idy.getUniqueName() + "<br>");
            }

        } catch (IMException e) {
            e.printStackTrace(out);
        }
    }
}
```

13

Pluggable Identity Management Framework

In addition to support for specific security providers documented earlier in this manual, OC4J includes a framework to more generally support heterogeneous third-party identity management systems for use by Web-based applications.

This chapter documents this identity management framework, covering the following topics:

- Overview of OracleAS JAAS Provider Identity Management Framework
- Identity Management Framework Programmatic Interfaces
- Identity Management Framework Configuration
- Summary of How to Use the Identity Management Framework
- Sample Use Case: Using a Header-Based Identity Token

Overview of OracleAS JAAS Provider Identity Management Framework

This section introduces the OC4J identity management framework, for use by Web-based applications, with the following discussion:

- Need for a Pluggable Identity Management Framework
- How the Identity Management Framework Works
- Overview of Identity Management Framework Programmatic Implementation
- Overview of Identity Management Framework Configuration
- Use of the Identity Management Framework by OC4J Java Single Sign-On

---

Notes:

- Note that by convention, the `<jazn>` setting `provider="XML"` is used with the pluggable identity management framework.
- If you use any identity repository other than the file-based provider or Oracle Internet Directory, you must define an administrative user account and administrator roles, grant the roles to the user, and grant necessary permissions to the roles, as discussed in "Creating the Administrative User and Roles and Granting RMI Permission" on page 10-9.

Need for a Pluggable Identity Management Framework

As discussed earlier in this document, Oracle Application Server provides security infrastructure such as Oracle Identity Management and Oracle Access Manager, which both include identity repositories. There is also support, as noted previously, for certain external LDAP providers (Active Directory and Sun Java System Directory Server).

However, in earlier releases there was no general framework to support other third-party identity management and security systems. The OC4J 10.1.3.1 implementation adds such a framework, allowing integration of heterogeneous third-party systems into OC4J, and hence allowing any J2EE application to interoperate with these third-party systems.

How the Identity Management Framework Works

This section introduces the components of the identity management framework, and summarizes how they work together. (Integration of third-party identity management systems into OC4J is based on standard JAAS login modules.)

In the general model of the identity management framework, the components work together as follows (as shown in Figure 13–1 below):

1. A *token collector* takes appropriate action to collect user credentials from the HTTP request. The token collector interface enables you to plug in a mechanism to collect the credentials. For form-based authentication, for example, a token collector would redirect to the login page for input of the user name and password. A token collector can be used to collect various types of credentials—including user name and password, or HTTP cookie—using basic, form-based, or custom authentication methods. An X.509 or SAML token can also be used, as long as it is received as part of the HTTP request object.
2. The token collector creates an appropriate *identity token* (depending on a specified token type) that corresponds to the user credentials, and passes it to OC4J. In the identity management framework, the type of token is typically an HTTP cookie or HTTP header that contains the user identity. If you do not want to use a cookie or header, then the HTTP request object itself can be used, and you are responsible for an implementation that would know how to parse a request object to retrieve the identity. (For example, the identity could be obtained by parsing and interpreting query parameters.)
3. A *token asserter* receives the identity token from OC4J and validates the identity, either by authenticating it against the third-party identity management system, or by validating the token in some other way. The token asserter interface enables you to plug in an assertion mechanism for the identity token. (For example: With OC4J Java SSO, which uses the identity management framework, the identity token is encoded into a cookie, and a symmetric key is used to validate the information in the cookie.)

   With respect to the identity management framework, *assertion* refers to the ability to interpret the identity token, validate its contents, and establish the identity corresponding to the token. This does not necessarily require a password or other credentials, as the token typically comes from a trusted source or is submitted with a key.

   Note that a token asserter can accept an identity authenticated by any single sign-on system, allowing the single sign-on system to collect credentials from a custom security provider and pass the identity to OC4J. The token asserter then verifies the identity and establishes the identity within the OC4J container.

4. Once the identity has been validated, the token asserter returns user information to OC4J through an *identity callback handler*, which it constructs. At this point, the identity passed in the callback handler is a trusted identity. The callback handler may be constructed simply by passing in an identity string, or an identity string and request object, for example.
5. OC4J passes the identity callback handler with user information to the login module configured for the application. The login module constructs appropriate callback types to handle user information, and passes them to the identity callback handler through the callback handler `handle(Callback[])` method (standard login module functionality). The login module collects principals for the user, such as for roles the user belongs to, by retrieving information from the third-party identity management system as necessary. The login module then sends a populated subject to OC4J.

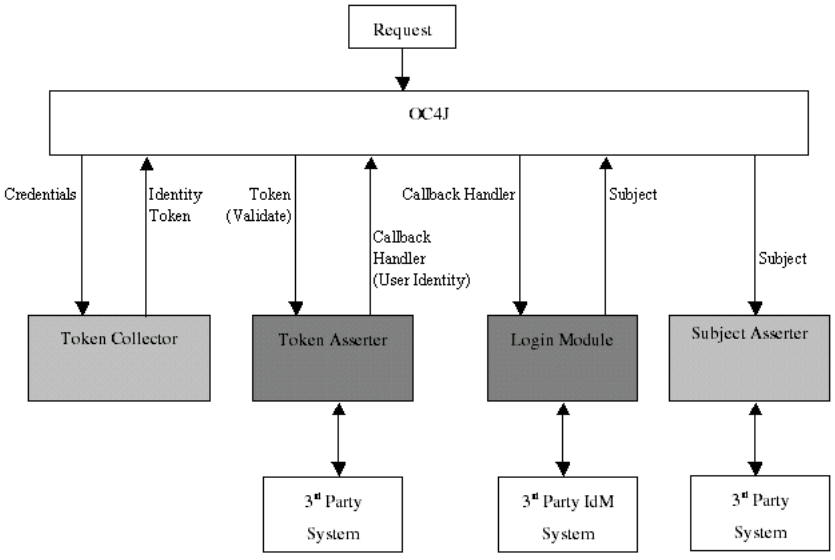
   Note that because the identity has already been validated by the token asserter, there is no need for the login module to perform authentication again. Also note that a custom login module for the identity management framework must be able to handle an identity callback handler.

   Alternative: Instead of implementing and configuring a login module for the application, you can provide a token asserter that creates the subject and implements the `getSubject()` method to return the subject (populated with user roles) directly to OC4J, without a login module. If you choose this option, there is an identity management framework property (`idm.subject.loginmodule.disabled`) that you must set accordingly.
6. A *subject asserter* can optionally validate the subject propagated by the login module. For example, you may want to sign and verify subject principals to protect the authenticity of the principals.

Oracle supplies Java interfaces for these components, which you can implement as appropriate to integrate a third-party identity management system with OC4J.

An administrator must configure OC4J to reference the implementation classes to use, and configure the login module.

Figure 13–1 Identity Management Framework Data Flow



Note: The token asserter and login module may access different backend identity systems. For example, the token asserter may access an identity management system in order to validate the identity, while the login module may access a separate identity store, such as a database, containing additional application-specific information about the user, such as user groups or roles. This additional information can be included in the subject that is populated for the user. (An *identity store* is a repository, such as a directory or database, that contains user information.)

See Also:

- "JAAS Authentication: Login Modules" on page 2-13
- "Identity Management Framework Programmatic Interfaces" on page 13-5
- "Identity Management Framework Configuration" on page 13-13

Overview of Identity Management Framework Programmatic Implementation

Use the following programmatic steps to use a third-party identity management system through the Oracle framework:

1. Provide a token collector class that implements the token collector interface. One option is to extend the Oracle implementation.
2. Choose the appropriate identity token class that Oracle supplies, for an HTTP cookie identity token, HTTP header identity token, or HTTP request identity token.
3. Provide a token asserter class that implements the token asserter interface.
4. Provide an identity callback handler class that implements the identity callback handler interface, or use the Oracle implementation.
5. Implement a login module (unless you choose the alternative of having your token asserter populate the subject and implement the `getSubject()` method). A login module must be able to appropriately process an identity callback handler instance.
6. Optionally provide a subject asserter class that implements the subject asserter interface.

See Also:

- "Identity Management Framework Programmatic Interfaces" on page 13-5

Overview of Identity Management Framework Configuration

The Oracle identity management framework requires the following configuration, discussed in detail later in this chapter:

1. An administrator configures `jazn.xml` to set identity management properties, such as to indicate the token collector implementation class, token asserter implementation class, and token type (such as HTTP header or cookie). Each property is set in a `<property>` subelement of the `<jazn>` element.

2. An administrator configures the appropriate login module (or optionally uses the default login module, `RealmLoginModule`). Configuration is stored under the `<jazn-loginconfig>` element in `system-jazn-data.xml`.
3. For any application to use the framework, the application assembler specifies the authentication method `CUSTOM_AUTH` in the `orion-application.xml` file packaged with the application.

See Also:

- "Identity Management Framework Configuration" on page 13-13

Use of the Identity Management Framework by OC4J Java Single Sign-On

The OC4J 10.1.3.1 implementation packages an alternative Java single sign-on implementation, Java SSO, that uses the identity management framework. Java SSO, discussed fully in Chapter 14, "OC4J Java Single Sign-On", is an SSO implementation that decouples OC4J from whatever identity system you want to use. It does not have particular infrastructure requirements such as Oracle Identity Management (required for Oracle Single Sign-On) or Oracle Access Manager (required for Oracle Access Manager SSO).

Java SSO includes a token collector implementation to collect cookie credentials and return an identity token to OC4J, and a token asserter implementation to verify the identity in the token and pass the identity back to OC4J in a callback handler.

Identity Management Framework Programmatic Interfaces

This section discusses interfaces, APIs, and Oracle implementations to integrate a third-party identity management system with OC4J.

Methods discussed here are typically called by the identity management framework in OC4J.

- Identity Token Interface and Oracle Implementations
- Token Collector Interface and Oracle Implementation
- Token Asserter Interface
- Identity Callback Handler Interface
- Subject Asserter Interface
- Packaging Your Identity Management Framework Implementation Classes

Notes:

- The identity management framework is driven by your configuration of the implementation classes you choose to use or provide.
- For any class implementations you supply yourself, we recommend that you package them in a single JAR file that you deploy as an OC4J shared library. (Refer to "Tasks to Share a Library" on page 6-14.)

See Also:

- *Oracle Containers for J2EE Security Java API Reference* for Javadoc that includes the identity management framework APIs

Identity Token Interface and Oracle Implementations

Oracle defines the following identity token interface:

`oracle.security.jazn.token.IdentityToken`

An identity token object contains user credentials. It is returned by the token collector and passed to the token asserter.

The `IdentityToken` interface specifies the following methods:

- `void setTokenType(String tokenType)`

  This method specifies the token type: `HTTP_COOKIE`, `HTTP_HEADER`, or `HTTP_REQUEST`.

- `String getTokenType()`

  This method returns the token type.

OC4J supplies the following `IdentityToken` implementations:

- `oracle.security.jazn.token.HttpCookieIdentityToken`

  Instances of this class can be constructed by the token collector by providing a `Map` instance that contains the name of the cookie as a key to the cookie value:

  ```
  HttpCookieIdentityToken(java.util.Map Cookies)
  ```

  It includes the following method to retrieve identity information:

  ```
  Map getCookies()
  ```

- `oracle.security.jazn.token.HttpHeaderIdentityToken`

  Instances of this class can be constructed by the token collector by providing a `Map` instance that contains the name of the header as a key to the header value:

  ```
  HttpHeaderIdentityToken(java.util.Map headerValues)
  ```

  It includes the following method to retrieve identity information:

  ```
  Map getHeaderValues()
  ```

- `oracle.security.jazn.token.HttpRequestIdentityToken`

  This class has the following constructor:

  ```
  HttpRequestIdentityToken(HttpServletRequest request)
  ```

  It includes the following method to retrieve identity information:

  ```
  HttpServletRequest getRequest()
  ```

Token Collector Interface and Oracle Implementation

Oracle supplies the following token collector interface:

`oracle.security.jazn.collector.TokenCollector`

Use an implementation of this interface to receive HTTP-based authentication credentials and create an identity token. The implementation class should include the following functionality:

1. Collect the credentials from the user request.
2. Create an identity token for the user.
3. Pass the token to OC4J.

A token collector takes the following input:

- Desired token type (so it knows what type to create)
- HTTP request object
- List of cookie or header names (as applicable, if the token type is cookie or header)
- Any user-defined properties, as applicable

If the token supplied by the user is not valid, the token is missing, or the user cannot be properly asserted (among other possible scenarios), then OC4J calls the token collector `fail()` method and passes it the reason for failure. The token collector can then take action as appropriate. The `fail()` method must be implemented to perform appropriate failure actions, such as redirecting the user back to a login page, for example.

The `TokenCollector` interface specifies the following methods:

- ```
  IdentityToken getToken(String tokenType,
                         HttpServletRequest request,
                         List names,
                         Properties props)
  ```

  This method returns an identity token as an Oracle `IdentityToken` instance (discussed in the preceding section, "Identity Token Interface and Oracle Implementations"). It takes as input the token type (`HTTP_COOKIE`, `HTTP_HEADER`, or `HTTP_REQUEST`); the current request object; a list of names of cookies or headers, from the HTTP request, that will constitute the token; and any custom properties that may be configured.

  The list of names is not applicable when using `HTTP_REQUEST`, so would be null in that case. When using `HTTP_COOKIE` or `HTTP_HEADER`, the list corresponds to cookie names or header names configured through the properties `idm.token.collector.cookie.#` or `idm.token.collector.header.#`, respectively. The `#` is replaced by numbers (1, 2, ..., n), as discussed in "Configuring Identity Management Framework Properties" on page 13-13.

  The list of properties would be specific to the particular token collector implementation and would be used by the token collector as applicable. By convention, any such property names must start with "`custom.`" (including the period). For example, Java SSO properties include `custom.sso.url.login` and `custom.sso.key.alias`.

- ```
  void fail(HttpServletRequest request,
            HttpServletResponse response,
            int reason) throws CollectorException
  ```

  This method would be called in a variety of failure modes, including:

  – The required token is not found in the HTTP request object during execution of the `getToken()` method.

  – The identity is not asserted successfully (such as if the supplied token is not valid).

  – The identity is successfully established but the user attempts to access resources without proper permission (authorization failure).

The integer `reason` is a code specified by OC4J that indicates why the failure occurred, and is one of the following values (defined in the class `oracle.security.jazn.collector.IdmErrorConstants`):

- `REASON_CHALLENGE_USER`: The user is not authenticated, but can be prompted for authentication (401 error in the browser). This is typically for basic authentication, where the browser will bring up the authentication window again. The value of this constant is 1.
- `REASON_INVALID_USER`: The user is not authenticated, for example due to invalid credentials (401 error). The value of this constant is 1.
- `REASON_UNAUTHORIZED`: The user is authenticated, but not authorized to access the protected resource (403 error). The value of this constant is 2.
- `REASON_PRECLUDED_ACCESS`: The resource is protected by a security constraint that contains no roles (403 error). The value of this constant is 3.

Oracle supplies the following `TokenCollector` implementation:

`oracle.security.jazn.collector.oc4j.TokenCollectorImpl`

This implementation is an abstract class that supports the use of an HTTP cookie or HTTP header for the identity token. If either of these is what you want to use, then you can extend `TokenCollectorImpl` and add appropriate error handling in the `fail()` method (such as redirecting the user to the login page, for example).

The `getToken()` method of `TokenCollectorImpl` returns the appropriate token type according to your configuration—either `HttpCookieIdentityToken` or `HttpHeaderIdentityToken`.

To use the HTTP request object itself instead of a cookie or header, you must implement a custom token collector.

A token collector uses the constructor of the appropriate identity token class to create the token. The `HttpCookieIdentityToken`, `HttpHeaderIdentityToken`, and `HttpRequestIdentityToken` constructors are documented in "Identity Token Interface and Oracle Implementations" on page 13-6.

Important: You must provide your own implementation of the `fail()` method, which is an abstract method in `TokenCollectorImpl`.

Token Asserter Interface

Oracle supplies the following token asserter interface:

`oracle.security.jazn.asserter.TokenAsserter`

Implement this interface to accept an identity token passed by OC4J and return an identity callback handler to OC4J.

Once OC4J successfully receives an identity token from the token collector, it passes the token to the token asserter, which must validate the identity for that token.

If the identity is validated successfully, then the user is considered authenticated and the identity is returned to OC4J as an identity callback handler.

A token asserter can also optionally get all the roles or groups for the identity and populate a subject for it, or this can be left to a login module.

The `TokenAsserter` interface specifies the following method:

- `IdentityCallbackHandler assertIdentity(String tokenType, IdentityToken token, Properties props) throws AsserterException`

  Upon successful validation, this method returns the asserted identity in an instance of `IdentityCallbackHandler` (discussed in the next section, "Identity Callback Handler Interface"). The token type is `HTTP_COOKIE`, `HTTP_HEADER`, or `HTTP_REQUEST`. The token is an instance of `IdentityToken` (discussed earlier). The properties would be the same as described for the `getToken()` method in "Token Collector Interface and Oracle Implementation" on page 13-6.

If you extend `TokenCollectorImpl` for the token collector, your token asserter must use the applicable identity token method to get identity information from the token. These methods in `HttpCookieIdentityToken`, `HttpHeaderIdentityToken`, and `HttpRequestIdentityToken` were noted earlier.

Note: You can avoid using a login module by implementing a token asserter that accesses the identity management system, populates the subject, and implements a `getSubject()` method. If you choose this alternative, the identity management property `idm.subject.loginmodule.disabled` must be set accordingly.

See Also:

- "Identity Token Interface and Oracle Implementations" on page 13-6 for information about the `IdentityToken` interface, `HttpCookieIdentityToken` implementation, `HttpHeaderIdentityToken` implementation, `HttpRequestIdentityToken` implementation, and related methods and constructors

Identity Callback Handler Interface

Use an identity callback handler in conjunction with a token asserter. Oracle supplies the following identity callback handler interface:

`oracle.security.jazn.callback.IdentityCallbackHandler`

An identity callback handler is constructed by the token asserter and used to pass the asserted identity to OC4J. This is the user identity that will also be passed, typically to a login module, so that a subject corresponding to the identity can be populated with principals as appropriate.

The `IdentityCallbackHandler` interface specifies the following methods:

- `void setIdentity(String identity) throws AsserterException`

  This method takes a string to specify the user identity.

- `String getIdentity()`

  This method returns a string with the user identity.

- `Subject getSubject()`

  If a subject was populated by the token asserter, then the identity management framework uses this method to retrieve it. Otherwise (if it is the login module that populates the subject), this method returns `null`.

There is also the following standard callback handler method:

- `void handle(Callback[] callbacks)`

  This method takes an array of one or more `javax.security.auth.callback.Callback` instances in order to handle (such as retrieve or display) information in the provided callbacks.

Oracle supplies the following identity callback handler implementation:

`oracle.security.jazn.callback.IdentityCallbackHandlerImpl`

Oracle Callback Implementations

Oracle provides the following callback implementations in package `oracle.security.jazn.callback`, for use with the identity management framework:

- `IdentityCallback`
- `HttpRequestCallback`

Identity Callback

Oracle supplies an identity callback class for use with the identity management framework:

`oracle.security.jazn.callback.IdentityCallback`

You can use an `IdentityCallback` instance to obtain the identity itself, as well as the authentication state of the identity (whether it has been authenticated) and the authentication method used to authenticate it.

The constructor takes no parameters:

`IdentityCallback()`

You can then use the identity callback handler `handle()` method to set an identity callback into the callback handler.

```
// In token asserter:
IdentityCallbackHandler ich = new IdentityCallbackHandlerImpl("identity");
...

// In login module:
IdentityCallback icb = new IdentityCallback();
Callback[] callback = {icb};
ich.handle(callback);
```

The `IdentityCallback` class has the following methods:

- `String getIdentity()`

  Retrieves the identity from the identity callback, as a string.

- `boolean isIdentityAsserted()`

  Returns `true` if the identity has been asserted, or `false` if it has not.

- `String getAuthenticationType()`

  Indicates the authentication method (such as form-based or basic) used to authenticate the identity.

HTTP Request Callback

Oracle provides an HTTP request callback class for identity management framework implementations that do not use HTTP cookies or HTTP headers as the token type:

`oracle.security.jazn.callback.HttpRequestCallback`

Using an HTTP request callback instance, the login module can obtain the credentials from the request object, authenticate the user, and populate a subject for the user.

The constructor takes no parameters:

`HttpRequestCallback()`

You can then use the identity callback handler `handle()` method to set an HTTP request callback into the callback handler.

```
// In token asserter:
IdentityCallbackHandler ich = new IdentityCallbackHandlerImpl("identity");
...

// In login module:
HttpRequestCallback httpreqcb = new HttpRequestCallback();
Callback[] callback = {httpreqcb};
ich.handle(callback);
```

The `HttpRequestCallback` class has the following methods:

- `void setHttpRequest(HttpServletRequest request)`

  Use this method to set the HTTP request object, presumably containing the user credentials.

- `HttpServletRequest getHttpRequest()`

  This method returns the HTTP request object.

Login Module Requirements

A typical implementation of the identity management framework includes a custom login module. After an identity is successfully asserted by the token asserter, the token asserter passes the identity to OC4J as an identity callback handler, which OC4J then passes to the login module.

In typical JAAS usage, a login module executes two key functions: authenticating the user, and populating a subject for the user. Within the identity management framework, however, the login module need not handle authentication. Its key function is to access the identity management system as necessary in order to get information about roles for the user and to populate the subject.

A login module used with the identity management framework must be able to handle an `IdentityCallbackHandler` instance that is used to pass the user name. The login module must accomplish the following:

1. Receive an `IdentityCallbackHandler` instance from OC4J.
2. Construct appropriate callbacks for handling user information. If you use cookie or header tokens, this can include the Oracle callback `oracle.security.jazn.callback.IdentityCallback` as well as standard callbacks such as `javax.security.auth.callback.NameCallback` and `PasswordCallback`. If you use HTTP request tokens, you would use an Oracle callback `oracle.security.jazn.callback.HttpRequestCallback`.

3. Pass callbacks to the `IdentityCallbackHandler` instance through its `handle(Callback[])` method.
4. Get the identity through the appropriate callback.
5. Go to the identity system to find out the roles (or groups) for the identity.
6. Populate a subject to send to OC4J.

Notes:

- It is not required to use a custom login module with the identity management framework. For the file-based provider or Oracle Identity Management, `RealmLoginModule` will suffice. For supported external LDAP providers (Active Directory and Sun Java System Directory Server), `LDAPLoginModule` will suffice.
- You can avoid using any login module if you instead implement a token asserter that accesses the identity management system, populates the subject itself, and implements the `getSubject()` method. If you choose this alternative, the identity management framework property `idm.subject.loginmodule.disabled` must be set accordingly.

See Also:

- "JAAS Authentication: Login Modules" on page 2-13

Subject Asserter Interface

You can optionally implement a subject asserter to validate the subject that was populated and returned by the token asserter or login module. OC4J will invoke a subject asserter if one is configured. In a typical use case, the subject has been signed, the signature has been added to the subject as a principal, and the subject asserter examines and validates the signature.

Oracle supplies the following subject asserter interface:

`oracle.security.jazn.asserter.SubjectAsserter`

The `SubjectAsserter` interface specifies the following method:

- `boolean assertSubject(Subject subject) throws AsserterException`

  Use this method to validate the subject; it returns true if successful.

Packaging Your Identity Management Framework Implementation Classes

Implementation classes you create for the identity management framework are not part of your application and are not packaged and deployed with your application. Package them into a single JAR file, and add the JAR file to the OC4J class path. You can do this as a shared library that can be imported by any application that needs to use it, as documented in "Tasks to Share a Library" on page 6-14.

If you implement a login module as part of your identity management framework implementation, you can include that in the same library or as a separate library.

Note: The shared library feature is the preferred way to make a JAR file available to OC4J applications, but alternatively you can put the JAR file into the following directory:

ORACLE_HOME/j2ee/home/lib/ext

(Note that this makes it globally available.)

Identity Management Framework Configuration

This section documents configuration for the identity management framework, covering the following topics:

- Configuring Identity Management Framework Properties
- Configuring the Identity Management Framework Login Module
- Configuring an Application to Use the Identity Management Framework
- Considerations for Multiple OC4J Instances

An administrator configures the framework properties and login module, while an application assembler sets the authentication method to configure the application to use the framework.

Note: By default in any single-instance OC4J installation, Java SSO, which is an implementation of the identity management framework, is preconfigured as shown in "Default Java SSO Property Settings for Single-Instance OC4J Installations" on page 14-14.

Configuring Identity Management Framework Properties

An administrator configures `jazn.xml` to set properties for the identity management framework, as appropriate for the class implementations being used. Table 13–1 describes the identity management framework properties.

Table 13–1 Identity Management Framework Properties

Property	Description
idm.authentication.name	This can be anything that specifies or qualifies how the identity management framework is being used. It is merely for reference and is not used by the framework. (For example, for OC4J Java SSO this is set to "`JavaSSO`" by convention.)
idm.token.type	Type of identity token to use: `HTTP_COOKIE`, `HTTP_HEADER`, or `HTTP_REQUEST`.
idm.token.collector.class	Fully qualified name of the class that implements the token collector interface.
idm.token.asserter.class	Fully qualified name of the class that implements the token asserter interface.
idm.subject.asserter.class	Fully qualified name of the class that implements the subject asserter interface (if applicable).
idm.subject.loginmodule.disabled	A boolean that specifies whether to invoke the login module (`false` by default, so the login module is enabled). If no login module is invoked, then the token asserter must populate the subject.

Table 13–1 (Cont.) Identity Management Framework Properties

Property	Description
idm.token.collector.cookie.#	Name(s) of cookie(s) containing identity information. You can specify one or more, replacing "#" with numbers, such as `idm.token.collector.cookie.1`. (It is typical to have just one.)
idm.token.collector.header.#	Name(s) of HTTP header(s) containing identity information. As with cookies, you can specify one or more, replacing "#" with numbers. (It is typical to have two, for the user name and password.)

Set these properties in `<property>` subelements of the `<jazn>` element, as in the following example:

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com">
   ...
   <!-- properties to configure the 3rd party IDM framework -->
   <property name="idm.authentication.name" value="JavaSSO" />
   <property name="idm.token.asserter.class"
             value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
   <property name="idm.token.collector.class"
             value="oracle.security.jazn.sso.SSOCookieTokenCollector" />
   <property name="idm.token.type" value="HTTP_COOKIE" />
   <property name="idm.token.collector.cookie.1" value="ORA_OC4J_SSO"/>
   ...
</jazn>
```

Important:

- By default, the identity management framework is configured to use Java SSO (discussed in Chapter 14, "OC4J Java Single Sign-On").
- If you associate an OC4J instance with an Oracle Internet Directory instance, the `<jazn>` element configuration in the `jazn.xml` file of the OC4J `home` instance is rewritten and any previous settings are lost.
- For an installation of the OC4J 10.1.3.1 patch over an existing 10.1.3.0.0 installation (as opposed to a fresh 10.1.3.1 installation), any default configurations are not in place—the properties are not referenced in `jazn.xml`. In this case, you should manually add the above configuration to `jazn.xml`.

Configuring the Identity Management Framework Login Module

Assuming you will use a custom login module (as is typical), it must be configured. You can configure a custom login module either during or after application deployment through Application Server Control, as discussed in "Configuring the Custom Security Provider in Application Server Control" on page 9-15. (The OracleAS JAAS Provider Admintool also has options for login module configuration, as noted in "Using Admintool to Configure Login Modules and Grant RMI Permission" on page 9-19.)

The configuration is stored in the `system-jazn-data.xml` file. The following example is for a custom login module `CustomLM` used with an application `myapp`:

```
<jazn-loginconfig>
```

```
   <application>
      <name>myapp</name>
      <login-modules>
         <login-module>
            <class>mypackage.CustomLM</class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
</jazn-loginconfig>
```

See Also:

- Chapter 9, "Login Modules"

Configuring an Application to Use the Identity Management Framework

For any application that will use the identity management framework, the application assembler must specify the authentication method `CUSTOM_AUTH` in the `<jazn-web-app>` element of the `orion-application.xml` file. Here is an example:

```
<jazn provider="XML" ... >
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH" />
   ...
</jazn>
```

This triggers usage of the identity management framework according to configuration of the framework properties in `jazn.xml`, and of the login module in `system-jazn-data.xml` (as applicable).

Important: If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. A `CUSTOM_AUTH` setting for the identity management framework would be lost and would have to be redone.

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

Notes:

- The `<jazn-web-app>` element is also supported in the `orion-web.xml` file, as a subelement of `<orion-web-app>`, for a particular Web application. A setting there overrides the `orion-application.xml` setting for that Web application.
- An authentication method setting in `orion-application.xml` (or `orion-web.xml`) overrides any authentication method setting in `web.xml`.

Considerations for Multiple OC4J Instances

For an installation type with multiple OC4J instances, any configuration for the identity management framework must be duplicated across instances. This includes identity management framework property settings in `jazn.xml`, and the login module configuration (as applicable) in `system-jazn-data.xml`. Take the following steps, as appropriate:

1. Repeat the same security provider configuration in each OC4J instance.
2. Use OC4J group functionality to coordinate `system-jazn-data.xml` updates. This updates each `system-jazn-data.xml` file across a group of OC4J instances. This would include user settings if you are using the file-based security provider. "Cluster MBean Browser Features and the J2EEServerGroup MBean" on page 7-18 discusses how to coordinate settings in each `system-jazn-data.xml` file across OC4J instances. There are also operations for maintaining login module configuration across instances (for example, `setLoginModule`).
3. As necessary, go to each OC4J instance and repeat configuration, through Application Server Control or manually as applicable. Generally speaking, for property settings in the `jazn.xml` file, such as for identity management framework settings, the only option is to manually configure `jazn.xml` in each OC4J instance.

See Also:

- For additional information about OC4J group features, the topic "Group OC4J Instances Page" in the Application Server Control online help

Summary of How to Use the Identity Management Framework

This section summarizes previous discussion of the key duties involved in using the identity management framework.

1. The integrator (the developer who is integrating some third-party identity management system into OC4J) implements classes as appropriate for identity management framework components: token collector, identity token, token asserter, identity callback handler, and subject asserter.
2. The integrator develops a custom login module as desired.
3. The integrator packages the implementation classes into a JAR file, and packages the login module into the same JAR file or a separate JAR file.
4. The integrator or administrator deploys each JAR file to the target system as a shared library.
5. The application is enabled to use the identity management framework. This can be accomplished by the assembler in setting `auth-method="CUSTOM_AUTH"` before deployment, in the `<jazn-web-app>` element of the `orion-application.xml` file packaged with the application.
6. The application deployer deploys the application, using Application Server Control. This includes configuring any custom login module (or it can be configured later by an administrator) and importing the shared library comprising the implementation classes.
7. On the target system, the administrator configures identity management framework properties in the `jazn.xml` file.

Sample Use Case: Using a Header-Based Identity Token

This sample assumes that the user is being authenticated against some custom identity store. Subsequently, a custom HTTP header ("`Acme-Custom-Auth`") containing the identity of the logged-in user is added to the request. Sample components function as follows:

- The token collector implementation extracts the header value.
- The token asserter implementation verifies the identity from the header, then asserts this identity through identity callback handler functionality using the `IdentityCallbackHandlerImpl` class that Oracle supplies.

The corresponding configuration is in `jazn.xml`, as shown.

Sample Token Collector: CollectorImpl.java

This section shows the code for the sample token collector implementation.

```
package com.acme.idm;

import java.io.IOException;

import java.util.List;
import java.util.Map;
import java.util.Properties;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import oracle.security.jazn.collector.CollectorException;
import oracle.security.jazn.collector.oc4j.TokenCollectorImpl;
import oracle.security.jazn.token.HttpHeaderIdentityToken;
import oracle.security.jazn.token.IdentityToken;
import oracle.security.jazn.token.TokenNotFoundException;
import oracle.security.jazn.collector.IdmErrorConstants;


public class CollectorImpl extends TokenCollectorImpl {
    public CollectorImpl() {
    }

    public IdentityToken getToken(String tokenType,
                                  HttpServletRequest request,
                                  List tokenNames,
                                  Properties properties)
                                            throws CollectorException,

TokenNotFoundException {
        if (null == tokenType || 0 == tokenType.length() ||
            !IdentityToken.HTTP_HEADER.equalsIgnoreCase(tokenType)) {
            throw new CollectorException("invalid token type" + tokenType);
        }

        HttpHeaderIdentityToken identityToken =
            (HttpHeaderIdentityToken)super.getToken(tokenType,
                                                    request,
                                                    tokenNames,
                                                    properties);
        Map m = identityToken.getHeaderValues();
```

```
        if (m == null || m.size() == 0) {
            throw new TokenNotFoundException("no HTTP Header token was found");
        }

        return identityToken;
    }

    public void fail(HttpServletRequest httpServletRequest,
                     HttpServletResponse httpServletResponse, int reason) {
        try {
            switch (reason) {
                case IdmErrorConstants.REASON_INVALID_USER:
              httpServletResponse.sendError(HttpServletResponse.SC_UNAUTHORIZED);
                case IdmErrorConstants.REASON_UNAUTHORIZED:
              httpServletResponse.sendError(HttpServletResponse.SC_FORBIDDEN);
            }
        } catch (Exception e) {
            System.err.println("failed to send response " + e);
            e.printStackTrace(System.err);
        }
    }
}
```

Sample Token Asserter: TokenAsserterImpl.java

This section shows the code for the sample token asserter implementation.

The assumption is that for all unauthenticated users, the Oracle HTTP Server or other Web server in front of OC4J will send "`Acme-Custom-Auth`" set to "`ANONYMOUS`". The Acme implementation requires that the user be authenticated. (A more production-quality implementation would very likely not pass a clear user name in the header, because there would be no way to validate where the header was generated.)

```
package com.acme.idm;

import java.util.Map;
import java.util.Properties;

import oracle.security.jazn.asserter.AsserterException;
import oracle.security.jazn.asserter.TokenAsserter;
import oracle.security.jazn.callback.IdentityCallbackHandler;
import oracle.security.jazn.callback.IdentityCallbackHandlerImpl;
import oracle.security.jazn.token.IdentityToken;
import oracle.security.jazn.token.HttpHeaderIdentityToken;

public class TokenAsserterImpl implements TokenAsserter {
    private static final String HEADER_NAME = "Acme-Custom-Auth";
    private static final String AUTH_TYPE = "CUSTOM_HTTP_HEADER";
    public TokenAsserterImpl() {
    }

    public IdentityCallbackHandler assertIdentity(String tokenType,
                                                  IdentityToken identityToken,
                                                  Properties properties)
                                                  throws AsserterException {
        if (tokenType != null &&
            tokenType.length() > 0 &&
            IdentityToken.HTTP_HEADER.equalsIgnoreCase(tokenType)) {
            HttpHeaderIdentityToken token =
                                          (HttpHeaderIdentityToken) identityToken;
```

```
            Map m = token.getHeaderValues();
            if (m != null && m.size() > 0) {
                String user = (String) m.get(HEADER_NAME);
                if ("ANONYMOUS".equalsIgnoreCase(user)) {
                    throw new AsserterException
                                    ("anon user - expected authenticated user");
                }
                IdentityCallbackHandler idcb =
                                        new IdentityCallbackHandlerImpl(user);
                idcb.setIdentityAsserted(true);
                idcb.setAuthenticationType(AUTH_TYPE);

                return idcb;
            }
            else {
                throw new AsserterException
                                ("not a valid token - no identity to assert");
            }
        }
    }
}
```

Sample Configuration: jazn.xml

This section shows the configuration in the `jazn.xml` file for this sample.

```
<?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" >
        <property name="idm.token.asserter.class"
                  value="com.acme.idm.TokenAsserterImpl" />
        <property name="idm.token.collector.class"
                  value="com.acme.idm.CollectorImpl" />
        <property name="idm.token.type" value="HTTP_HEADER" />
        <property name="idm.token.collector.header.1" value="Acme-Custom-Auth" />
        <property name="idm.authentication.name" value="Acme-IDM" />
</jazn>
```

14

OC4J Java Single Sign-On

The OC4J 10.1.3.1 implementation packages an alternative Java single sign-on solution that does not rely on additional required infrastructure like other Oracle Application Server single sign-on products. This Java SSO, based on the OracleAS JAAS Provider identity management framework discussed in Chapter 13, can be used across Web applications in any of the following deployment scenarios:

- Web applications are deployed in the same application EAR file.
- Web applications are deployed in different application EAR files in the same OC4J instance.
- Web applications are deployed in different application EAR files in different OC4J instances, where the Web applications share a common security domain and cookie domain.
- A single application EAR file, including Web applications, is deployed into multiple OC4J instances within an OC4J cluster.

This chapter documents OC4J Java SSO, covering the following topics:

Notes:

- If you use any identity store other than the file-based provider or Oracle Internet Directory, you must define an administrative user account and administrative roles, grant the roles to the user, and grant necessary permissions to the roles, as discussed in "Creating the Administrative User and Roles and Granting RMI Permission" on page 10-9.
- This upgrades and replaces the functionality of the "Lightweight J2EE Single Sign-On" feature documented for Oracle Application Server 10*g* Release 2 (10.1.2). That feature is desupported.

Overview of OC4J Java SSO

This section provides an overview of Java SSO, including the following:

Need for an OC4J Container-Level Java Single Sign-On Solution

In Oracle Application Server, to use Oracle Single Sign-On you must have a separate Oracle Application Server installation with Oracle Identity Management, including Oracle Internet Directory, Oracle Single Sign-On, and Oracle Database. Alternatively, to use the Oracle Access Manager single sign-on solution, you must have a full Oracle Access Manager installation. These single sign-on solutions are closely coupled with the required infrastructure. While robust, neither may be viable for smaller deployment scenarios, and neither can be used with standalone OC4J by itself.

As an alternative, the OC4J 10.1.3.1 implementation supplies Java SSO, which is packaged with OC4J itself and decouples OC4J from whatever identity server you want to use. This allows applications deployed in the same OC4J instance or cluster to have a single point of authentication and the ability to share user identities.

OC4J Java SSO can be used with any security provider supported by OracleAS JAAS Provider (summarized in "Introducing the OracleAS JAAS Provider and Security Providers" on page 3-1). Furthermore, any login module (including custom ones) that conforms to certain requirements can be used in conjunction with Java SSO. In particular, the login modules `RealmLoginModule`, `LDAPLoginModule`, and `DBTableOraDataSourceLoginModule` provided with OC4J support the identity assertion feature of the identity management framework and can be used with Java SSO and partner applications. (An instance of `IdentityCallback`, discussed in "Oracle Callback Implementations" on page 13-10, is used to determine the identity to assert.)

Notes: Be aware of the following limitations in the 10.1.3.1 Java SSO implementation:

- There is no access control or authorization functionality. It is for authentication and identity assertion only.
- It is limited to Web applications sharing a common cookie domain.
- The `UserManager` interface (deprecated for general use anyway) is not supported.

How Java SSO Works

Java SSO includes a dedicated Java SSO application (predeployed in OC4J) and an implementation of the OracleAS JAAS Provider identity management framework.

This section discusses the following:

Single Sign-On Interaction and Logical Flow

There are three basic scenarios to consider in discussing how OC4J Java SSO works with a particular application:

- A user with no identity token tries to access the application, in which case login is required. As discussed in Chapter 13, "Pluggable Identity Management Framework", an identity token corresponds to a user and user credentials. For Java SSO, the token is an HTTP cookie.
- A user with an identity token tries to access the application, in which case the token must be validated and the identity asserted.
- An authenticated user tries to access the application, where a subject and identity token are both available and no further authentication or assertion is required.

Typical interaction with Java SSO is as follows:

1. A client tries to access protected content (requests a protected URL) from an application, `App1` for example.
2. `App1` requires SSO authentication; the client is prompted for credentials by Java SSO.
3. The client provides credentials.
4. Java SSO delegates authentication to the security provider.
5. Upon successful authentication, the security subsystem establishes the identity.
6. The SSO provider (the Java SSO implementation of the identity management framework) encodes an identity token—the Java SSO cookie—corresponding to the authenticated user, and returns the cookie to the client.
7. The client presents the cookie to access `App1`.
8. `App1` determines the client identity from the cookie and allows the client to access the protected content.
9. The client reuses the cookie to try to access protected content of another application, `App2` for example.
10. `App2` determines the client identity from the cookie and allows the client to access the protected content.

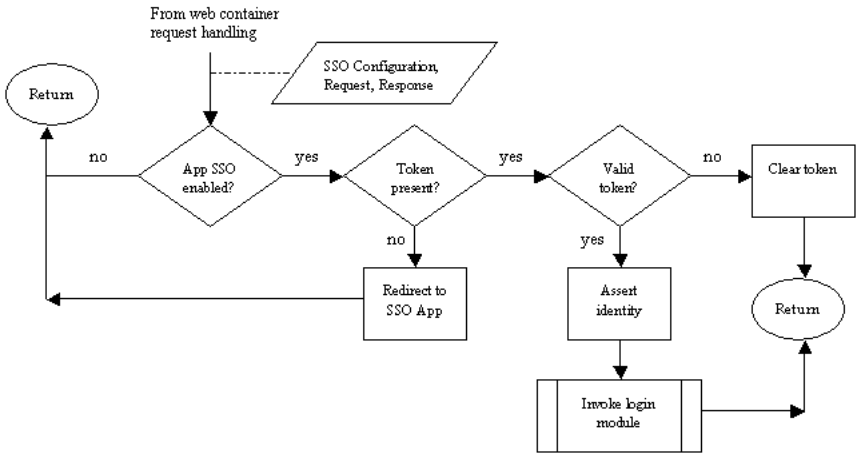
When using Java SSO, a user session is active until one of the following occurs:

- The application calls the Java SSO logout API (discussed in "Java SSO Logout API" on page 14-18).
- The Java SSO cookie becomes invalid.

The set of applications that will share usage of Java SSO are referred to as *partner applications*. These may be customer applications, or may be Oracle Application Server console applications such as Application Server Control, Oracle BPEL Process Manager, or Oracle Web Services Manager. A partner application is one of the applications in a *security domain* that delegates authentication through Java SSO. In a security domain, multiple applications share common identity stores, algorithms, and keys used in the authentication and validation process.

Figure 14–1 following presents a flow chart for Java SSO logic.

Figure 14–1 Java SSO Internal Logic Flow



Java SSO Runtime Operations

The Java SSO application, `javasso`, is deployed in all OC4J instances as a built-in system application. The application consists of the login servlet `SSOLogin` and the logout servlet `SSOLogout`.

When you start OC4J, `javasso` is typically disabled (except in an Oracle Application Server "Basic Install"). An administrator can start `javasso` through Application Server Control.

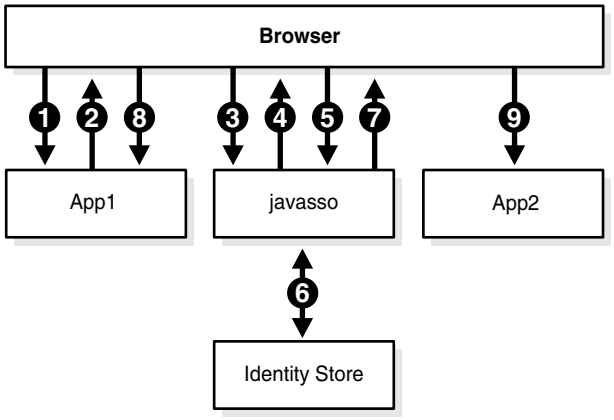
Once `javasso` is enabled, attempts to access Java SSO partner applications result in a process such as the following, as depicted in Figure 14–2 below.

> **Note:** Assume `App1` and `App2` below are partner applications configured to use Java SSO for single sign-on.

1. A user, through the browser, attempts to access a protected URL for application `App1`.
2. There is a redirect to `javasso` (more specifically, the `SSOLogin` servlet) through the browser.
3. The `App1` URL is passed to `javasso`.
4. Through `javasso` functionality, the user is prompted with the Java SSO login page.
5. The user supplies the user name and password.
6. Through `javasso` functionality, the user is authenticated according to information in the identity store. After the user is successfully authenticated, the `SSOLogin` servlet maps the authenticated identity to the Java SSO cookie (identity token), then uses a symmetric key cipher to sign and encrypt the contents of the cookie.
7. There is a redirect to `App1` through the browser, and the secured cookie is sent back to the client.
8. The user accesses the content of `App1`.

9. The user attempts to access a protected URL for application `App2`, and is able to access the content of `App2` without authentication.

Figure 14–2 Java SSO Runtime Operations



Java SSO Implementation of the Identity Management Framework

Java SSO is an implementation of the identity management framework documented in Chapter 13, "Pluggable Identity Management Framework". The Java SSO implementation uses a cookie for credentials.

Java SSO is installed with OC4J and includes the following:

- There is a cookie, `ORA_OC4J_SSO`, for user credentials.
- A token collector implementation class, `SSOCookieTokenCollector`, collects credentials from a cookie (passed through the HTTP request), constructs a cookie identity token, and returns the cookie identity token to OC4J. Content of the token includes the user identity, issuing authority, token lifetime, and a digital signature to ensure content integrity.
- A token asserter implementation class, `SSOCookieTokenAsserter`, asserts the user identity—receives the cookie token from OC4J, validates it, determines the identity in the cookie, and passes the asserted identity back to OC4J in an identity callback handler.

  To decode the cookie token, the token asserter retrieves a shared key (shared between all partner applications and the Java SSO application) and decrypts the contents of the cookie. The token asserter validates the data by verifying the signature. (If the cookie has expired, an error is thrown, typically resulting in the user being redirected to the login page.)

> **Note:** In the Java SSO implementation, the cookie used as the identity token is a session cookie; it is not persistent.

Java SSO Deployment Scenarios

Of particular significance, consider these Java SSO deployment scenarios:

- Single instance of OC4J

  The `javasso` application, customer applications, and Application Server Control all run in the same OC4J instance.

- SOA installation with two OC4J instances, using the file-based provider

  The Oracle BPEL Process Manager (the Business Process Execution Language engine) and OWSM (Oracle Web Services Manager) run on the `OC4J_SOA` instance. The `javasso` application, customer applications, and Application Server Control run on the `OC4J_Home` instance.

  The `system-jazn-data.xml` file (file-based provider) must be synchronized between the two instances, as discussed in "Considerations with the File-Based Provider and Two OC4J Instances" on page 14-16.

- SOA installation with two OC4J instances, using Oracle Internet Directory

  The Oracle BPEL Process Manager and OWSM run on the `OC4J_SOA` instance. The `javasso` application, customer applications, and Application Server Control run on the `OC4J_Home` instance.

  A single Oracle Internet Directory instance is used as the security provider.

- High-availability installation with multiple OC4J instances, using Oracle Internet Directory

  The `javasso` application and all customer applications are available on all instances.

  A single Oracle Internet Directory instance is used as the security provider.

Notes:

- Partner applications (either Oracle Application Server console applications or customer applications) can be split between OC4J instances however desired, but scenarios such as the above are typical.
- Customers using Oracle Internet Directory within a full Oracle Identity Management environment would typically use Oracle Single Sign-On instead of Java SSO, but there may be situations where Java SSO with Oracle Internet Directory is useful.

Summary of Java SSO Configuration

Beyond standard configuration for a user's J2EE application, the following configuration steps are required. These are covered in detail in "Java SSO Setup and Configuration" on page 14-7.

1. The `javasso` application must be started, through Application Server Control.
2. In any target OC4J instance, the `jazn.xml` file must have the correct property settings for Java SSO and its implementation of the identity management framework. You can set properties through Application Server Control, which can also generate the symmetric key. Settings are indicated in `<property>` subelements of the `<jazn>` element. Configuration includes the following (default settings may be adequate):
   - Java SSO login URL and logout URL
   - Symmetric key used to encrypt the cookie token
   - DNS domain to which the Java SSO cookie is restricted (required in advanced installations, across multiple OC4J instances and hosts)

(Be sure the cookie domain is set properly, as discussed in the troubleshooting section, "Returned to Java SSO Login Page Despite Correct Credentials" on page 14-20.)

3. The partner applications using Java SSO must all be configured to use the same security provider and same identity store as the `javasso` application. (In particular, when using the file-based provider, use the same repository file. For a scenario with multiple OC4J instances, see "Considerations with the File-Based Provider and Two OC4J Instances" on page 14-16.) By default, `javasso` is configured to use the file-based provider, but you can change this using the Application Server Control Console.

   For partner applications that are customer applications, you can configure the security provider during deployment. For Application Server Control itself, there are special steps in Application Server Control Console to change the security provider. For other Oracle Application Server console applications that you want to include, if changes are necessary they can be handled similarly to changing the security provider for `javasso`.

4. Each partner application must be enabled to use Java SSO. This can also be configured through Application Server Control . Being enabled to use Java SSO is indicated by a setting of `auth-method="CUSTOM_AUTH"` in the `<jazn-web-app>` element, a subelement of `<jazn>` in `orion-application.xml`.

> **Note:** Java SSO supports any provider supported by OracleAS JAAS Provider.

About the Java SSO Login Page and Error Page

This section discusses localization and customization of the Java SSO login page and error pages.

Localization Support for the Java SSO Login Page and Error Pages

The Java SSO login page and error pages shipped with OC4J support localization. Content displayed by these pages is localized according to the browser settings.

Customizing the Login Page or Error Page

We do not directly support the use of a custom login page or custom error pages with Java SSO, but you can customize the deployed `login.jsp`, `loginerror.jsp`, and `error.jsp` files and do one of the following:

- Replace the existing files in the deployment directory:

  ORACLE_HOME`/j2ee/home/applications/javasso/javasso-web/WEB-INF`

- Repackage and redeploy the `javasso.ear` file with the customized files.

Java SSO Setup and Configuration

This section discusses Java SSO configuration in OC4J, covering the following topics:

- Configuring Java SSO through Application Server Control
- Java SSO Configuration Properties
- Configuration for Enabling Partner Applications for Java SSO

- Configuration for Special Scenarios

We recommend that you configure Java SSO and partner applications through Application Server Control, which is covered in the first section. The second and third sections show the configuration parameters and properties that are set as a result (and can also be set manually as necessary). The final section is for special considerations and scenarios.

Configuring Java SSO through Application Server Control

In the Application Server Control Console, use the Java SSO Configuration page. You can navigate to this page as follows:

- In an Oracle Application Server clustered environment, from the Cluster Topology page, choose **Java SSO Configuration**.
- In an OC4J standalone environment:
  1. From the OC4J Home page, select the **Administration** tab.
  2. From the Administration page, under "Properties", go to the SSO Configuration task.

The rest of this section covers the following topics:

- Start the javasso Application
- Set Java SSO Properties and Generate the Symmetric Key
- Configure the Security Provider for the javasso Application
- Configure the Security Provider for Partner Applications
- Enable Partner Applications to Use Java SSO

Start the javasso Application

In an Oracle Application Server clustered environment, from the Cluster Topology page, you can start the `javasso` application for any OC4J instance (as necessary):

1. Under Members (which lists the application server and OC4J instances), choose the **Expand All** link to see all the applications under each instance.
2. Select the `javasso` application for the desired instance.
3. Choose **Start**.

In an OC4J standalone environment, you can start the `javasso` application as follows:

1. From the OC4J Home page, choose **Applications**.
2. From the Applications page, select the `javasso` application.
3. Choose **Start**.

Set Java SSO Properties and Generate the Symmetric Key

From the Java SSO Configuration page, select the **Instances and Properties** tab to set Java SSO properties. This also enables you to generate a new symmetric key.

From the Instances and Properties page, you can specify the following:

- The key type. Choose `AES_128_CBC`, `AES_192_CBC`, `AES_256_CBC`, or `DES_EDE_CBC`. This information is coded into the `custom.sso.key.alias` Java SSO property, discussed in "Java SSO Configuration Properties" on page 14-12.

- The URL for the Java SSO `SSOLogin` servlet (corresponding to the `custom.sso.url.login` property). If you have multiple hosts, use the fully qualified domain name.
- The URL for the Java SSO `SSOLogout` servlet (corresponding to the `custom.sso.url.logout` property). If you have multiple hosts, use the fully qualified domain name.
- The session timeout in seconds (corresponding to the `custom.sso.session.timeout` property). Note that this is a hard timeout, not an inactivity timeout. The session will timeout after this amount of time no matter what.
- The number of login attempts to allow (corresponding to the `custom.sso.login.attempts` property).

If Java SSO has not yet been properly configured, use **Configure Java SSO**, after setting any of the above parameters as appropriate, to configure it. This will automatically generate a new symmetric key.

To reconfigure Java SSO after it has been properly configured, there is a checkbox to specify whether a new symmetric key should be generated when you apply the reconfiguration. You should generate a new key in either of the following circumstances:

- If an error condition has arisen where different OC4J instances may be using different keys. **All instances must use the same key.**
- If you want a new key just to be cautious from a security standpoint. (We recommend that you do this on a regular basis.)

Choose **Apply** (which appears in place of **Configure Java SSO**), after setting any of the above parameters as appropriate, to apply the reconfiguration and generate the key (as applicable).

Settings are reflected in `<property>` subelements under the `<jazn>` element in the `jazn.xml` file. In a cluster, these settings and the key apply across all OC4J instances.

Important:

- When using the 10.1.3.1 patch set (as opposed to a fresh 10.1.3.1 installation), default `jazn.xml` settings shown in "Default Java SSO Property Settings for Single-Instance OC4J Installations" on page 14-14 will not be present. Add these settings manually before configuring Java SSO.
- By default, the `jazn.xml` file has a dummy value for the key (in the `custom.sso.key.alias` setting) until you configure Java SSO.
- All applications configured to use the same Java SSO should share a single key. (This is a potential issue only in a scenario with multiple OC4J instances.)
- You must restart OC4J instances for these changes to take effect. You will be notified and prompted to do so.
- If you will be using Oracle Identity Management, make Java SSO settings *after* associating the OC4J instance with the Oracle Internet Directory instance. When you do the association, the `<jazn>` element configuration in the `jazn.xml` file of the OC4J `home` instance is rewritten and any previous settings are lost.

Configure the Security Provider for the javasso Application

The `javasso` application that is used, and each partner application using Java SSO, must all be configured to use the same security provider. By default, `javasso` is configured to use the file-based provider.

To change the security provider for `javasso` in a given OC4J instance, go to the OC4J Home page for that instance in the Application Server Control Console. In a standalone environment, there is only one OC4J Home page. In an Oracle Application Server clustered environment, choose the desired OC4J instance in the Cluster Topology page.

1. From the OC4J Home page, select the **Administration** tab.
2. From the Administration page, under "Security", go to the Security Providers task.
3. From the Security Providers page, go to the Edit task for `javasso`.
4. From the Security Provider page for `javasso`, choose **Change Security Provider**.

Notes:

- You must restart the application for a change in security provider to take effect.
- Where there are multiple OC4J instances and not all `javasso` applications have the same security provider, there is a **Change Security Provider** option relating to the `javasso` application for each OC4J instance on the Java SSO Instances and Properties page in the Application Server Control Console. This simplifies the process of configuring the `javasso` application you want to use.

Each type of security provider necessitates its own set of configuration tasks, documented in the following locations:

- "Changing to the File-Based Provider after Deployment" on page 7-3
- "Changing to Oracle Identity Management after Deployment" on page 8-14
- "Changing to a Custom Security Provider after Deployment" on page 9-18
- "Changing to an External LDAP Provider after Deployment" on page 10-5

Configure the Security Provider for Partner Applications

As noted above, all partner applications and the `javasso` application must be configured to use the same security provider. For customer applications that are partner applications, configure the appropriate security provider during deployment. Each type of security provider necessitates its own set of configuration tasks, documented in the following locations:

- "Configuring the File-Based Provider during Application Deployment" on page 7-3
- "Specifying Oracle Identity Management during Deployment" on page 8-13
- "Specifying and Configuring a Custom Security Provider during Deployment" on page 9-15
- "Specifying and Configuring an External LDAP Provider during Deployment" on page 10-3

If you want any Oracle Application Server console applications to share in using Java SSO (Application Server Control, for example), you must also ensure that the console applications use the same security provider.

For Application Server Control, you can change the security provider through the following steps:

1. In the OC4J Home page for the instance where Application Server Control is running, choose **Administration**.
2. In the Administration page, under "Security", go to the Security Providers task.
3. In the Security Providers page, choose **Application Server Control Security**.
4. In the initial Setup page, choose **Security Provider**.
5. In the Security Provider page, choose **Change Security Provider**.
6. In the Change Security Provider page, specify whether to use the `system-jazn-data.xml` file-based provider, an application-specific `jazn-data.xml` file-based provider, or Oracle Identity Management (if an Oracle Internet Directory instance had previously been associated with the OC4J instance, as described in "Associating Oracle Internet Directory with OC4J" on page 8-6).
7. Choose **OK**. You must then restart Application Server Control for the change to take effect.

For other console applications, steps for changing the security provider would be similar to those for `javasso` in the preceding section, "Configure the Security Provider for the javasso Application". You must then restart the application for a change in security provider to take effect.

Enable Partner Applications to Use Java SSO

From the Java SSO Configuration page, select the **Participating Applications** tab to enable applications to use Java SSO.

From the Participating Applications page:

1. Select **Java SSO Enabled** for each desired application.
2. Choose **Apply** to enable them to use Java SSO.

This results in the setting `auth-method="CUSTOM_AUTH"` in the `<jazn-web-app>` element of the `orion-application.xml` file for each application.

Once you have started the `javasso` application and enabled an application to use Java SSO, invoking the application results in a login prompt through Java SSO.

Important:

- If you will be using Oracle Identity Management as the security provider for an application, configure that before enabling the application for Java SSO. When you switch from the file-based provider to Oracle Identity Management through Application Server Control at any time for any partner application, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any previous setting to enable the application for Java SSO (the `CUSTOM_AUTH` setting) will be lost and must be redone.

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

- You must restart OC4J instances for these changes to take effect. You will be notified and prompted to do so.
- If you have multiple hosts, use the fully qualified domain name when invoking partner applications from your browser.

Java SSO Configuration Properties

In general, we recommend that you configure Java SSO through Application Server Control, as discussed in preceding sections.

This section provides descriptions for all Java SSO configuration properties and shows the default settings (as applicable) upon OC4J installation.

Java SSO Configuration Property Descriptions

Table 14–1 describes Java SSO properties whose settings would be reflected in the `jazn.xml` file, in `<property>` subelements of the `<jazn>` element.

Properties that are not supported through Application Server Control (as indicated by "n/a" in the table) must be set manually.

See Also:

- "Set Java SSO Properties and Generate the Symmetric Key" on page 14-8

Table 14–1 Java SSO Properties

Property	Description	Name in Application Server Control
custom.sso.app.url.default	The default URL for redirection from the `SSOLogin` servlet if none is provided.	n/a

Table 14–1 (Cont.) Java SSO Properties

Property	Description	Name in Application Server Control
custom.sso.cookie.domain	The Java SSO cookie is restricted to this DNS domain. By default, without a domain setting, the cookie is restricted to the host from which the `SSOLogin` URL is accessed. A domain setting is required in advanced installations, across multiple OC4J instances and hosts. (Be sure the cookie domain is set properly, as discussed in the troubleshooting section "Returned to Java SSO Login Page Despite Correct Credentials" on page 14-20.)	n/a
custom.sso.cookie.path	Path restriction for the cookie. Default: `/`	n/a
custom.sso.cookie.secure	If `true`, only HTTPS sites are supported. Default: `false`	n/a
custom.sso.key.alias	Stores the shared symmetric key used to sign the contents of the Java SSO cookie identity token.	n/a
custom.sso.login.attempts	Number of login attempts allowed. Default: 3	Number of Login Attempts
custom.sso.session.timeout	Period of time (in seconds) for which the Java SSO cookie is valid. Note that this is a hard timeout, not an inactivity timeout. The session will timeout after this amount of time no matter what. Default: 7200	Session Timeout
custom.sso.token.asserter.authtypes	Java SSO authentication method. This is the method that will be used to authenticate end users once they are redirected to Java SSO. You can specify a standard J2EE Web application authentication method. Default: `FORM`	n/a
custom.sso.url.login	URL of the `SSOLogin` servlet. If you have multiple hosts, use the fully qualified domain name.	Java SSO Login URL
custom.sso.url.logout	URL of the `SSOLogout` servlet. If you have multiple hosts, use the fully qualified domain name.	Java SSO Logout URL

Table 14–1 (Cont.) Java SSO Properties

Property	Description	Name in Application Server Control
realm.user.manager.lazy.initialization	Optional. This property is used with SSO configured applications and enables the lazy initialization of their configured realm user managers. Valid values are `true` or `false`. If the property is not specified, the default default value is `false`. When set to `true` and the container hosting SSO configuration applications is started while OID is down, accessing one of the SSO configured applications will result in a `500-Internal Server Error`. This is expected and is caused by the initialization failure of the configured realm manager due to the fact that OID is down. When OID comes back online, simply pressing the **Refresh** button in the browser produces the SSO login screen, because the configured realm manager will be reinitialized successfully.	n/a
jazn.policy.ignore.cs	Optional. This property is used with SSO configured applications and enables the lazy initialization of their configured realm user managers. Valid values are `true` or `false`. If the property is not specified, the default default value is `false`. When set to `true`, the JAAS policy provider ignores the codebase in authorization checks/permission computation. This behavior affects the XML and LDAP providers. If codebase authorization checks are not needed, setting this property to `true` delivers a performance improvement especially for the LDAP provider.	n/a

Important: If you associate an OC4J instance with an Oracle Internet Directory instance, the `<jazn>` element configuration in the `jazn.xml` file of the OC4J `home` instance is rewritten and any previous settings are lost.

Default Java SSO Property Settings for Single-Instance OC4J Installations

By default in an installation where there is a single OC4J instance, OC4J is set up to use Java SSO as the identity management framework implementation. This includes settings for Java SSO properties documented in the preceding section, "Java SSO Configuration Property Descriptions", and Java SSO settings for identity management framework properties documented in "Configuring Identity Management Framework Properties" on page 13-13.

Here are the preconfigured settings in `jazn.xml`:

```
<jazn provider="XML" location="./system-jazn-data.xml" default-realm="jazn.com">
   <!-- properties to configure the 3rd party IDM framework -->
   <property name="idm.authentication.name" value="JavaSSO" />
   <property name="idm.token.asserter.class"
             value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
   <property name="idm.token.collector.class"
             value="oracle.security.jazn.sso.SSOCookieTokenCollector" />
   <property name="idm.token.type" value="HTTP_COOKIE" />
   <property name="idm.token.collector.cookie.1" value="ORA_OC4J_SSO" />

   <!-- properties for the out of the box Java SSO -->
   <property name="custom.sso.url.login" value="/jsso/SSOLogin" />
   <property name="custom.sso.url.logout" value="/jsso/SSOLogout" />
   <property name="custom.sso.key.alias" value="ssoSymmetricKey" />
</jazn>
```

Important:

- These settings do not exist by default in some scenarios, as noted in "General Considerations for Multiple OC4J Instances" on page 14-17 and "Considerations When Using the 10.1.3.1 Patch over 10.1.3.0.0" on page 14-18. In these cases, manually add the default configuration to `jazn.xml` before configuring Java SSO.
- By default, there is a dummy value for the key in the `custom.sso.key.alias` setting (as shown above) until you configure Java SSO through Application Server Control, as discussed in "Set Java SSO Properties and Generate the Symmetric Key" on page 14-8. Once you configure Java SSO, the setting will change to something like the following:

  ```
  value="{AES-128}IdG4OPSqGPJ8hZFPn1W4Uw=="
  ```

Configuration for Enabling Partner Applications for Java SSO

Each partner application that is to use Java SSO must be configured to enable Java SSO. We recommend that you do this through Application Server Control, as discussed in "Enable Partner Applications to Use Java SSO" on page 14-11.

Enabling a partner application to use Java SSO is indicated by the authentication method setting "`CUSTOM_AUTH`" in the `<jazn-web-app>` element of the `orion-application.xml` file, as in the following example:

```
<jazn provider="XML" ... >
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH" />
   ...
</jazn>
```

This triggers usage of Java SSO and its implementation of the identity management framework according to configuration in `jazn.xml`.

Important: If you will be using Oracle Identity Management as the security provider for an application, configure that before enabling the application for Java SSO. When you switch from the file-based provider to Oracle Identity Management through Application Server Control at any time for any partner application, the `<jazn>` element in `orion-application.xml` is replaced with the following. Any previous setting to enable the application for Java SSO (the `CUSTOM_AUTH` setting) will be lost and must be redone.

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

Notes:

- Notice that because the OC4J default configuration specifies Java SSO as the identity management framework implementation, enabling a partner application to use the identity management framework (`auth-method="CUSTOM_AUTH"`) enables it to use Java SSO.
- The `<jazn-web-app>` element is also supported in the `orion-web.xml` file, as a subelement of `<orion-web-app>`, for a particular Web application. A setting there overrides the `orion-application.xml` setting for that Web application.
- An authentication method setting in `orion-application.xml` (or `orion-web.xml`) overrides any authentication method setting in `web.xml`.

See Also:

- "Configuring an Application to Use the Identity Management Framework" on page 13-15

Configuration for Special Scenarios

Preceding discussion covered general configuration for Java SSO, but this section covers the following special considerations:

- Considerations with the File-Based Provider and Two OC4J Instances
- General Considerations for Multiple OC4J Instances
- Considerations When Using the 10.1.3.1 Patch over 10.1.3.0.0

Considerations with the File-Based Provider and Two OC4J Instances

This section discusses a use case where you want to employ Java SSO with the file-based provider for single sign-on access across two OC4J instances, such as in a SOA installation. The key issue is synchronizing user accounts and roles within two `system-jazn-data.xml` files across both OC4J instances.

Consider a use case where:

- A customer has created OC4J instances `OC4J_HOME` and `OC4J_SOA` (such as through the installer or manually through `createinstance`).
- They would like to activate Java SSO in `OC4J_HOME` and deploy SOA applications in `OC4J_SOA`.

- They would like to use the file-based provider in both `OC4J_HOME` and `OC4J_SOA`.

By default, each OC4J instance is configured with its own file-based provider, the `system-jazn-data.xml` file. For Java SSO to work correctly across both instances, however, user accounts and user roles must be synchronized across the two `system-jazn-data.xml` files. Using OC4J group functionality, the following steps make this possible:

1. Create an OC4J group for Java SSO—`JSSO_GROUP`, for example. Refer to "OC4J Basic Group Features" on page 7-17.
2. Start the Java SSO application in `OC4J_HOME` after making sure it is configured to use the file-based provider, `system-jazn-data.xml`, which is its default setting. (Note that the Java SSO application should be started in only one OC4J instance in this sample use case.)
3. Deploy the J2EE applications that are to use Java SSO to the `OC4J_SOA` instance; configure each application to use the file-based provider; and enable Java SSO for each application.
4. Add `OC4J_HOME` and `OC4J_SOA` as members of `JSSO_GROUP`. Again refer to "OC4J Basic Group Features".

For ongoing user and role administration for these applications, the customer would use the OC4J `J2EEServerGroup` MBean that corresponds to `JSSO_GROUP`, executing appropriate security provider MBean operations. This MBean will ensure synchronization of user accounts and roles across the `JSSO_GROUP` members. Refer to "Cluster MBean Browser Features and the J2EEServerGroup MBean" on page 7-18.

Note: You can achieve the same result by manually coordinating user configuration between the `system-jazn-data.xml` files on the two instances, although this may be more prone to error.

See Also:

- For additional information about OC4J group features, the topic "Group OC4J Instances Page" in the Application Server Control online help

General Considerations for Multiple OC4J Instances

For an installation type with multiple OC4J instances, the default configurations discussed in "Default Java SSO Property Settings for Single-Instance OC4J Installations" on page 14-14 are not in place—the properties may be referenced in `jazn.xml`, but not set.

Any configuration for Java SSO (and related configuration for the identity management framework) must be duplicated across instances. The primary concern is Java SSO or related identity management framework properties in `jazn.xml` that are not set through Application Server Control, as well as `system-jazn-data.xml` settings (as applicable). Java SSO properties that *are* set through Application Server Control, as well as the shared Java SSO key, are already propagated to all OC4J instances through Application Server Control functionality.

Take the following steps, as appropriate:

1. Complete your Java SSO setup and configuration, as described under "Configuring Java SSO through Application Server Control" on page 14-8.

Remember that Java SSO properties set through Application Server Control apply across instances.

2. Ensure that any other relevant property settings in `jazn.xml`, such as Java SSO properties not supported by Application Server Control, or additional related identity management framework properties, are duplicated across OC4J instances. Generally speaking, the only option for such properties is to manually repeat the configuration in the `jazn.xml` file of each OC4J instance.
3. Use OC4J group functionality to configure `system-jazn-data.xml` across a group. (This may be for the file-based security provider, login module settings, or policy settings, for example.) OC4J groups are discussed in "OC4J Basic Group Features" on page 7-17. Also, "Cluster MBean Browser Features and the J2EEServerGroup MBean" on page 7-18 discusses how to coordinate `system-jazn-data.xml` user settings across OC4J instances. There are also operations for maintaining login module configuration across instances (for example, `setLoginModule`).

See Also:

- For additional information about OC4J group features, the topic "Group OC4J Instances Page" in the Application Server Control online help

Considerations When Using the 10.1.3.1 Patch over 10.1.3.0.0

For an installation of the OC4J 10.1.3.1 patch over an existing 10.1.3.0.0 installation (as opposed to a fresh 10.1.3.1 installation), the default configurations discussed in "Default Java SSO Property Settings for Single-Instance OC4J Installations" on page 14-14 are not in place—the properties are not referenced in `jazn.xml`.

Manually add the default configuration to `jazn.xml`, then complete the appropriate Java SSO configuration described under "Configuring Java SSO through Application Server Control" on page 14-8.

Java SSO APIs

Java SSO supplies the following utility class:

`oracle.security.jazn.sso.util.JSSOUtil`

In the OC4J 10.1.3.1 implementation, this class provides the following API:

- Java SSO Logout API

See Also:

- *Oracle Containers for J2EE Security Java API Reference* for Javadoc that includes the Java SSO APIs

Java SSO Logout API

The `JSSOUtil` utility class that comes with Java SSO provides a static `logout()` method, to be used as the last step in application session logout, once you have completed any application-specific logout preparations and processing:

- `logout(HttpServletResponse response, String targetURL)`

  Supply the HTTP response object that corresponds to the request being processed, and specify the URL where the application is to be redirected after logout. The

application is first directed to the Java SSO `SSOLogout` servlet, which unsets the Java SSO cookie, and then to the specified target URL.

For example:

```
JSSOUtil.logout(response, "http://portal.acme.com");
```

Summary of How to Use Java SSO

This section summarizes the previously discussed key action items for using Java SSO.

For development, using Java SSO is mostly transparent, without special coding required. The one point of integration, which you can optionally use, is the logout API documented in the previous section, "Java SSO Logout API".

For configuration, use the following steps, described in detail in "Java SSO Setup and Configuration" on page 14-7. All steps can be completed through Application Server Control.

1. Start the `javasso` application.
2. Configure Java SSO properties. (Settings are written into `<property>` elements under the `<jazn>` element in `jazn.xml`.)
3. Configure the `javasso` application to use the appropriate security provider.
4. Configure partner applications to use the same security provider as `javasso`. For customer applications, set the security provider during deployment. For any Oracle Application Server console applications (such as Application Server Control) that will share in Java SSO, you can change the security provider appropriately.
5. Enable each partner application to use Java SSO. (This results in the setting `auth-method="CUSTOM_AUTH"` in the application `orion-application.xml` file.) This is the only configuration step required to convert an existing application to be Java SSO-enabled. You can leave `<login-config>` settings in `web.xml` files unchanged (they are overridden).

Important: If you have multiple hosts, use the fully qualified domain name when invoking partner applications from your browser.

Troubleshooting Java SSO

This section discusses the following issues you may encounter:

- Page Not Found
- Returned to Java SSO Login Page Despite Correct Credentials

Page Not Found

If you receive a "Page Not Found" error when attempting to access a partner application that has been enabled for Java SSO, it may be that the `javasso` application is not running. You can test this by trying to access the `SSOLogin` servlet. You can go to the URL directly:

```
http://host:port/jsso/SSOLogin
```

Or you can choose **Test Login URL** from the Application Server Control page where you configure Java SSO properties.

If you are redirected to the *host*:*port* top-level page, then the test is successful.

Instructions for starting the javasso application are in "Start the javasso Application" on page 14-8.

Returned to Java SSO Login Page Despite Correct Credentials

If you provide correct credentials to the Java SSO login page but keep getting returned to that page, consider the following possibilities:

- You are attempting to access a partner application through localhost or an IP address, which you are not allowed to do.
- You are attempting to access a partner application where the domain is not specified exactly as it is for the javasso application. For example, it would not work to access an application as http://sso/myapp/index.jsp when the Java SSO login URL set through Application Server Control (and corresponding to the custom.sso.url.login property) is set as http://sso.mydomain.com/jsso/SSOLogin.
- The cookie domain setting (custom.sso.cookie.domain property) does not match the domain where you try to access a partner application. For example, it would not work to access an application at http://app.mydomain.com/myapp/index.jsp when the cookie domain is set to .sso.mydomain.com. It *would* work, however, if the cookie domain is set to .mydomain.com.

15

SSL Communication with OC4J

OC4J supports Secure Sockets Layer (SSL) communication, as follows:

- In an Oracle Application Server environment (OPMN-managed), OC4J supports SSL communication between Oracle HTTP Server and OC4J 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.)
- In a standalone OC4J environment, or in an OPMN-managed environment without Oracle HTTP Server, OC4J supports SSL communication directly between a client and OC4J, using HTTPS.
- OC4J supports ORMI over SSL, or ORMIS. With this feature, OC4J supports RMI communication over SSL between objects across OC4J server instances.
- OC4J supports TCP over SSL, or TCPS. With this feature, communication between an OC4J data source and the database is secured.

Note that this chapter discusses the standalone OC4J scenario as well as two scenarios in an Oracle Application Server environment—OPMN-managed OC4J with OC4J as the Web listener, and OPMN-managed OC4J with Oracle HTTP Server as the Web listener. In all, the following topics are covered:

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.

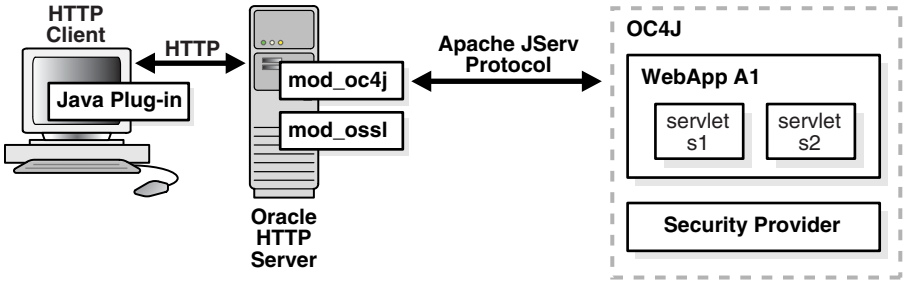
See Also:

- "Transport-Level Security" on page 1-3 (overview)
- *Oracle HTTP Server Administrator's Guide* for related information about using SSL with Oracle HTTP Server
- *Oracle Application Server Administrator's Guide* for information about configuring additional Oracle Application Server components to take advantage of SSL

Integrating the Security Provider with SSL-Enabled Applications

This section describes the responsibilities of Oracle components when an HTTP client request is initiated in an SSL-enabled J2EE environment. Figure 15–1 shows an application running in such an environment.

Figure 15–1 Oracle Component Integration in SSL-Enabled J2EE Environments



Here are the steps of the process:

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 "Introduction to JAAS Mode" on page 5-5 and "Configuring and Using JAAS Mode" on page 5-18 for information about how JAAS mode works.

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 an application. 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.

Notes:

- A keystore or wallet must include a private key, a public key, and a trusted certificate (such as a CA certificate) in order to be used in establishing trust relationships.
- OC4J comes with a default wallet that does not include a trusted certificate. You should create and provision your own wallet instead of using the default wallet.

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 at the OC4J end for 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 (such as Verisign or Thawte; links following shortly).

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, at the Oracle HTTP Server end you must execute the following steps as necessary.

1. Execute steps equivalent to the preceding steps for OC4J, but using a wallet and Oracle Wallet Manager instead of a keystore and the `keytool` utility.
2. As appropriate: **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, depending on the entity.

   b. From Oracle HTTP Server, use Oracle Wallet Manager to import the certificate of the entity.
3. As appropriate: **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 15-15.)

a. Obtain the certificate of the entity in any appropriate way, such as by exporting it from the entity. The exact steps vary, 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.

Using SSL with Standalone OC4J

Standalone OC4J supports SSL communication directly between a client and OC4J, using HTTPS. This section describes how to accomplish this.

Use the following steps:

1. Create a keystore with an RSA private/public key pair using the `keytool` utility. In this example, we generate a keystore to reside in a file named `mykeystore.jks`, which has a password of `123456`, using the RSA key pair generation algorithm:

```
% keytool -genkey -keyalg RSA -keystore mykeystore.jks -storepass 123456
```

In this tool:

- The `keystore` option sets the filename where the keys are stored.
- The `storepass` option sets the password for protecting the keystore. You can optionally omit this from the command line and be prompted for a password instead.

The `keytool` prompts you for additional 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=Redwood Shores, ST=CA, C=US> correct?
  [no]:  yes

Enter key password for <mykey>
        (RETURN if same as keystore password):
```

Always press `RETURN` for the key password. In OC4J 10.1.3.x implementations, 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: The keytool utility supports PKCS12 format wallets as well as JKS format keystores.

See Also:

- For detailed information about the keytool utility:

  http://java.sun.com/j2se/1.5.0/docs/tooldocs/solaris/keytool.html

- For the ISO two-letter country code list:

  http://www.iso.org/iso/english_country_names_and_code_elements

2. If you do not have a secure-web-site.xml file, create one in the following location: *ORACLE_HOME*/j2ee/home/config/secure-web-site.xml (by convention). You can start by copying whatever content you need from default-web-site.xml. This typically includes the following subelements under the <web-site> element:

   - <web-app> (for each Web application you want to secure)
   - <access-log> (for logging; confirm this specifies an appropriate log file)
   - <default-web-app>

   You will also need an <ssl-config> element for your SSL configuration; that is discussed later in this procedure.

3. Update secure-web-site.xml with the following elements:

   a. Update the <web-site> element to add secure="true" and to set the port value to some available port. (For example, port="4443". To use the default of 443, you have to be a super user.) For standalone OC4J, use HTTP protocol, which is the default setting. (The setting protocol="http" in combination with secure="true" results in HTTPS being used.)

   ```
   <web-site port="4443" secure="true" protocol="http"
             display-name="Default OracleAS Containers for J2EE Web Site" >
      ...
   </web-site>
   ```

   (Also remember to change the display-name setting as appropriate.)

   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.

   Note: You can hide the password through password indirection, as described in "Using Password Indirection" on page 6-1.

c. Also see "Optional Steps in secure-web-site.xml" below.

d. Save the changes to `secure-web-site.xml`.

Here is an example:

```
<?xml version="1.0"?>
<web-site display-name="OC4J 10g Secure Web Site" protocol="http"
          port="4443" secure="true">
   <ssl-config keystore="./roadrunner.jks" keystore-password="welcome1" />
   <default-web-app application="default" name="defaultWebApp" root="/" />
   <web-app application="SSLDemos-Project1-WS" name="WebServices"
            load-on-startup="true" root="/SSLDemos-Project1-context-root" />
   <access-log path="../log/default-web-access2.log" split="day" />
</web-site>
```

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 optional steps may be appropriate as well:

1. 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 according to 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.

Important: In standalone OC4J (no Oracle HTTP Server), setting `needs-client-auth="true"` is required in order to use client-cert authentication mode. See "Requesting Client Authentication" on page 15-15 for related information.

2. 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.

   See Also:

   - *Oracle Containers for J2EE Configuration and Administration Guide* for more information about sharing Web applications between Web sites

3. 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 `cookie-domain` 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" />
   ```

4. Specify the cipher suites to use—combinations of cryptographic specifications that define security algorithms and key sizes. (This is a server-side cipher suite setting, as opposed to `HTTPClient` settings discussed in Chapter 16, "Oracle Security for Client Connections".) Use the `cipher-suites` attribute of the `<ssl-config>` element in `secure-web-site.xml`, as in this example:

   ```
   <ssl-config keystore="your_keystore" keystore-password="your_password"
               cipher-suites="SSL_RSA_WITH_RC4_128_SHA,
                              SSL_RSA_WITH_RC4_128_MD5,..." />
   ```

This is a comma-delimited list of cipher suites. If you omit this attribute, the set of cipher suites used is according to those specified as "enabled by default" in the reference documentation at:

http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

Example 15–1 HTTPS Communication with Client Authentication

The following example 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 in standalone OC4J, 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 in OPMN-Managed OC4J without Oracle HTTP Server

OC4J, when managed by OPMN and used as its own Web listener (in other words, without Oracle HTTP Server), supports SSL communication between a client and OC4J using HTTPS (similarly to the standalone OC4J scenario discussed previously). You must also configure OPMN to support HTTPS.

This section describes how to use SSL in this OPMN-managed scenario, involving the following steps:

1. Configure OC4J with SSL (Scenario without Oracle HTTP Server)
2. Configure OPMN to Support HTTPS (Scenario without Oracle HTTP Server)

Configure OC4J with SSL (Scenario without Oracle HTTP Server)

Configuring OC4J with SSL in an OPMN-managed environment without Oracle HTTP Server is largely the same as for standalone OC4J, as covered above in "Using SSL with Standalone OC4J" on page 15-5. Refer there for additional details.

1. Create a keystore.
2. Create `secure-web-site.xml` (by convention). (Copy content from `default-web-site.xml` as appropriate.)
3. Update `secure-web-site.xml` with the following elements:
   a. Update the `<web-site>` element to add `secure="true"`. Typically, in an OPMN-managed environment, the choice of port defers to OPMN, as

indicated by setting `port="0"`. Use `protocol="http"`. (The setting `protocol="http"` in combination with `secure="true"` results in HTTPS being used.)

```
<web-site port="0" secure="true" protocol="http"
          display-name="Default OracleAS Containers for J2EE Web Site" >
   ...
</web-site>
```

(Also remember to change the `display-name` setting as appropriate.)

b. Add an `<ssl-config>` element under `<web-site>` to define the keystore location and password, using the `keystore` and `keystore-password` attributes.

c. Save the changes to `secure-web-site.xml`.

Here is an example:

```
<web-site display-name="OC4J Web Site" protocol="http" 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>
```

4. Ensure that `server.xml` points to `secure-web-site.xml`.
5. Stop and restart OC4J to initialize `secure-web-site.xml`.

Notes:

- 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.
- In an OC4J environment without Oracle HTTP Server, setting `needs-client-auth="true"` is required in order to use client-cert authentication mode. See "Requesting Client Authentication" on page 15-15 for related information.

Configure OPMN to Support HTTPS (Scenario without Oracle HTTP Server)

To use SSL with OC4J in an OPMN-managed environment without Oracle HTTP Server, you must configure OPMN to support HTTPS. 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 HTTPS protocol for the Web site:

```
<ias-component id="OC4J">
```

```
   ...
   <port id="secure-web-site" range="12501-12600" protocol="https"/>
   ...
</ias-component>
```

See Also:

- *Oracle Process Manager and Notification Server Administrator's Guide* for details about OPMN and `opmn.xml`

Using SSL in OPMN-Managed OC4J with Oracle HTTP Server

In an Oracle Application Server environment, where OC4J is managed by OPMN and Oracle HTTP Server is the Web listener, OC4J supports SSL communication between Oracle HTTP Server and OC4J using AJPS (the secure version of Apache JServ Protocol). This section describes how to use SSL in this scenario, involving the following:

1. Configure OC4J with SSL (Scenario with Oracle HTTP Server)
2. Configure AJP over SSL

This discussion concludes with sample configuration files.

Note: In Oracle Application Server 10.1.3.x implementations, SSL is enabled by default for communication between Oracle HTTP Server and the client. No special steps are required. Note that this is unrelated to the discussion in this section for using SSL between OC4J and Oracle HTTP Server.

See Also:

- *Oracle HTTP Server Administrator's Guide* for related information about using SSL with Oracle HTTP Server, including how to customize your configuration to enable client authentication
- *Oracle Application Server Administrator's Guide* for information about configuring additional Oracle Application Server components to take advantage of SSL

Configure OC4J with SSL (Scenario with Oracle HTTP Server)

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 15-5. Refer there for additional details.

1. Create a keystore.
2. Create `secure-web-site.xml` (by convention). Copy content from `default-web-site.xml` as appropriate.
3. Update `secure-web-site.xml` with the following elements:
   a. Update the `<web-site>` element to add `secure="true"`. Typically, in an Oracle Application Server environment, the choice of port defers to OPMN (as indicated by the `port="0"` setting, which is added automatically). Also, use `protocol="ajp13"`, which is the default setting in an Oracle Application Server environment. (The setting `protocol="ajp13"` in combination with `secure="true"` results in AJPS being used.)

```
<web-site port="0" secure="true" protocol="ajp13"
          display-name="Default OracleAS Containers for J2EE Web Site" >
   ...
</web-site>
```

(Also remember to change the `display-name` setting as appropriate.)

b. Add an `<ssl-config>` element under `<web-site>` to define the keystore location and password, using the `keystore` and `keystore-password` attributes.

c. Save the changes to `secure-web-site.xml`.

Here is an example; individual `<web-app>` entries have been excluded from the example for better readability:

```
<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>
```

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.

4. Ensure that `server.xml` points to `secure-web-site.xml`.
5. Stop and restart OC4J to initialize `secure-web-site.xml`.

Important: The OC4J 10.1.3 implementation can support only a single AJP/AJPS Web site. For example, you cannot have a `default-web-site.xml` file that configures an AJP Web site at the same time that you have a `secure-web-site.xml` file that configures an AJPS Web site.

Notes:

- 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.
- In an Oracle Application Server environment, where Oracle HTTP Server is the Web listener, the OC4J `needs-client-auth` attribute in the `<ssl-config>` element is not relevant to SSL authentication from the browser—that would be by arrangement between the client and Oracle HTTP Server. This attribute *is* relevant, however, if you want OC4J to require SSL authentication from Oracle HTTP Server. See "Requesting Client Authentication" on page 15-15 for related information.

Configure AJP over SSL

This section covers the following aspects of using AJP over SSL:

- Configure AJPS between OC4J and Oracle HTTP Server
- Configure OPMN to Support AJPS (Scenario with Oracle HTTP Server)

Configure AJPS between OC4J and Oracle HTTP Server

Configuring AJPS between OC4J and Oracle HTTP Server 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. 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 \
            -storepass password
   ```

   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 can optionally omit `storepass` from the command line and be prompted for a password instead. You will receive a message confirming the certificate file name if the command is successful.
3. Use Oracle Wallet Manager to import the generated certificate into your wallet. Under "Operations", use "Import Trusted Certificate".
4. 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`
   - Regarding steps 1 and 3, *Oracle Application Server Administrator's Guide* for details about managing wallets and certificates

Configure OPMN to Support AJPS (Scenario with Oracle HTTP Server)

In an Oracle Application Server environment, configuration steps are also required for 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

This section discusses SSL authentication of a client to a server, focusing in particular on the OC4J client authentication mode that uses the OC4J `needs-client-auth` attribute.

The discussion considers the following scenarios:

- An end user as a direct client to OC4J (either standalone or OPMN-managed)
- Oracle HTTP Server, for purposes of this discussion, as a client to OC4J in an Oracle Application Server environment
- An end user as a client to Oracle HTTP Server in an Oracle Application Server environment (for which OC4J configuration, including the `needs-client-auth` attribute, is irrelevant)

Overview of OC4J Client Authentication Mode

OC4J supports a client authentication mode in which the OC4J server explicitly requests SSL authentication from the client (or, in an Oracle Application Server environment, from Oracle HTTP Server). A client must identify itself with a digital certificate, which OC4J requests upon connection.

During secure communication with authentication between the client and OC4J, the following functionality is executed:

- All communications between the two are encrypted.
- OC4J is authenticated to the client. A "secret key" is securely exchanged and used for the encryption of the link.
- The client is authenticated to OC4J.

Request client authentication through the `needs-client-auth` attribute of the `<ssl-config>` element in `secure-web-site.xml`, as shown in the following example, and then perform the steps that follow:

```
<web-site ... secure="true" ... >
   ...
   <ssl-config keystore="path_and_file" keystore-password="pwd"
               needs-client-auth="true" />
</web-site>
```

1. A certificate that OC4J trusts is called a *trust point*. Decide which of the certificates in the chain from the client 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 15-5. 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 the client.
5. Provide the certificate for the client initiation of the secure connection.

OC4J accepts or rejects a client entity for secure communication according to the client identity. If the root certificate of the client is recognized, then the client is accepted. The keystore specified in the `<ssl-config>` element in `secure-web-site.xml` must contain the certificates of any clients that are authorized to connect to OC4J.

In the certificate chain from the client, the trust point is the first certificate 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 the client is in the keystore.
- The root CA certificate in the certificate chain from the client 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.

Client Authentication to OC4J

When the OC4J HTTP listener is used—in standalone OC4J, or in OPMN-managed OC4J without Oracle HTTP Server—you can set `needs-client-auth="true"` in the `<ssl-config>` element in `secure-web-site.xml` to request SSL authentication from the client (end user). In fact, this setting is required in order to use client-cert authentication with the OC4J listener.

To provide a certificate, 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 from a Java client.

Refer to the preceding section, "Overview of OC4J Client Authentication Mode", for additional information.

See Also:

- Chapter 16, "Oracle Security for Client Connections"

Oracle HTTP Server Authentication to OC4J in Oracle Application Server

In an Oracle Application Server environment, Oracle HTTP Server, in a manner of speaking, acts as the client to OC4J. For client authentication in this mode, Oracle HTTP Server must have its own certificate, and authenticates itself by sending a certificate and a certificate chain that ends with a root certificate. OC4J can be configured to accept only root certificates from a specified list in establishing a chain of trust back to a client.

For this scenario, consider Oracle HTTP Server to be the client for purposes of the discussion in "Overview of OC4J Client Authentication Mode" on page 15-16.

AJPS (secure Apache JServ Protocol) is used instead of HTTPS for secure communication between Oracle HTTP Server and OC4J.

Client Authentication to Oracle HTTP Server

For requesting SSL authentication from the client (end user) to Oracle HTTP Server in an Oracle Application Server environment, OC4J is not involved and its configuration (including the `needs-client-auth` attribute) is not relevant. This calls for an arrangement between the client and Oracle HTTP Server. Refer to the *Oracle HTTP Server Administrator's Guide* for information about using SSL with Oracle HTTP Server, including how to customize your configuration to enable client authentication.

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.

Troubleshooting and Debugging SSL

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 mykeystore -import -alias cacert -file cacert.cer
  (keytool -keystore mykeystore -import -alias intercert -file inter.cer)
```

If you use an intermediate CA `keytool` utility, then execute these commands:

```
% keytool -keystore mykeystore -genkey -keyalg RSA -alias serverkey
% keytool -keystore mykeystore -certreq -file my.host.com.csr
```

Get the certificate from the Certificate Signing Request (CSR), then execute the following command:

```
% keytool -keystore mykeystore -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: javax.net.debug Property

You can display verbose debug information from the Java Secure Socket Extension (JSSE) implementation for SSL connections, using the `javax.net.debug` property. To get a list of options, start OC4J as follows:

- `-Djavax.net.debug=help` (for a list of options)
- `-Djavax.net.debug=all` (for debug messages with full verbosity)

This includes display of 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

See Also:

- *Oracle Containers for J2EE Services Guide* for general information about using ORMI in OC4J

Configuring ORMIS for Standalone OC4J

ORMIS configuration and related RMI configuration require 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
- Disable ORMI with ORMIS Enabled (Optional)

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 details 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 startup of 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 in the following example:

```
<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.

Disable ORMI with ORMIS Enabled (Optional)

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 15-19 and "Configure rmi.xml for ORMIS" on page 15-19. 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 `<host-access>` or `<ip-access>` subelements (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 `<host-access>` or `<ip-access>` subelements (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 is 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 three choices for where to specify these settings, in order of precedence:

- As JSSE properties:

  ```
  -Djavax.net.ssl.keyStore=keystore_path
  -Djavax.net.ssl.keyStorePassword=keystore_password
  ```

- As properties in `jndi.properties` (ignored if JSSE property settings are used):

  ```
  oc4j.[rmi.]keyStoreLoc=keystore_path
  oc4j.[rmi.]keyStorePass=keystore_password
  ```

- As properties in `ejb_sec.properties` (ignored if JSSE or `jndi.properties` property settings are used):

  ```
  oc4j.[rmi.]keyStoreLoc=keystore_path
  oc4j.[rmi.]keyStorePass=keystore_password
  ```

> **Notes:**
>
> - Either `oc4j.keyStoreLoc` or `oc4j.rmi.keyStoreLoc` is acceptable. Similarly for `keyStorePass`.
> - To use `ejb_sec.properties`, place it in the current directory, from which the client Java VM was launched.

Enabling ORMI Tunneling through HTTPS

The RMI chapter of the *Oracle Containers for J2EE Services Guide* discusses how to configure ORMI tunneling through HTTP.

It is also possible to configure ORMI tunneling through HTTPS for SSL functionality. The basic steps are as follows:

1. Complete your SSL configuration as discussed in "Using SSL with Standalone OC4J" on page 15-5 or "Using SSL in OPMN-Managed OC4J with Oracle HTTP Server" on page 15-11 (as applicable).

   For standalone OC4J, this consists of creating your keystore and configuring the `secure-web-site.xml` file.

   For an Oracle Application Server environment, this consists of creating your keystore, configuring the `secure-web-site.xml` file (with some differences compared to standalone OC4J, as noted), configuring AJP over SSL (as desired),

and configuring OPMN to enable HTTP and use SSL. Note that SSL is enabled by default in Oracle HTTP Server.

2. Configure your client appropriately (parallel to the steps in "Configuring Clients to Use ORMIS" on page 15-22):

   a. For either standalone OC4J or an Oracle Application Server environment, specify the `ormi:https` protocol in setting the `java.naming.provider.url` environment property, which defines the URI of the system and application:

      ```
      java.naming.provider.url=ormi:https://hostname:https_port/appname
      ```

      For standalone OC4J, the *`https_port`* is as specified in `secure-web-site.xml`, as discussed in "Using SSL with Standalone OC4J" on page 15-5. In an Oracle Application Server environment, the *`https_port`* is the Oracle HTTP Server SSL port.

   b. Configure the keystore and password, as discussed in "Specify the Keystore and Password" on page 15-23.

The following client code snippet uses a URL with the `ormi:https` protocol:

```
private static Context getInitialContext() throws NamingException {
    Hashtable env = new Hashtable();
    env.put( Context.INITIAL_CONTEXT_FACTORY,
      "oracle.j2ee.naming.ApplicationClientInitialContextFactory" );
    env.put( Context.SECURITY_PRINCIPAL, "oc4jadmin" );
    env.put( Context.SECURITY_CREDENTIALS, "welcome1" );
    env.put(Context.PROVIDER_URL, "ormi:https://localhost:443/apache-ejb");
    env.put("oc4j.keyStoreLoc",
      "C:/product/iasSOA0622/Apache/Apache/conf/ssl.wlt/default/ewallet.p12");
    env.put("oc4j.keyStorePass", "welcome");

    return new InitialContext( env );
}
```

Configuring a TCPS Data Source

An OC4J managed data source can be configured to use TCP over SSL (TCPS) to secure communication with an Oracle database. Using TCPS has the following requirements:

- A database that has been configured with a trustore (in this case a PKCS12 wallet). Contact a DBA if you require assistance setting up security on the database server.
- A 10.2.0.3 (or higher) JDBC driver (the application server uses the 10.1.0.5 JDBC driver as the default driver). The driver can be configured as a shared library for the container or for a specific application. See the `"Utilizing the OC4J Class-Loading Framework"` in the *Oracle Containers for J2EE Developer's Guide*.
- An Oracle PKI security provider entry in the application server's JRE security file (*`JAVA_HOME`*`/jre/lib/security/java.security`). For example:

  ```
  security.provider.3=oracle.security.pki.OraclePKIProvider
  ```

Note: For more information on JDBC / SSL using TCPS, refer to the following document:
http://www.oracle.com/technology/tech/java/sqlj_jdbc/pdf/wp-oracle-jdbc_thin_ssl.pdf

The managed data source definition must include the following:

- The `oracle.jdbc.driver.OracleDriver` factory class.
- A connection URL that defines the TCPS protocol as shown below:

```
jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps) \
                  (HOST=host)(PORT=2484)) \
                  (CONNECT_DATA=(SERVICE_NAME=orcl)))
```

- A database username/password along with the trustore credentials
- The path to the client wallet on the server. The location can be anywhere on the file system. For example:

```
/home/wallets/client/ewallet.p12
```

The following example demonstrates a managed data source definition that uses TCPS to connect to an Oracle database. The example uses SSL for encryption only and passes in a set of cipher suites.

```
<managed-data-source
  connection-pool-name="ConnPoolTCPS"
  jndi-name="jdbc/sslDS"
  name="jdbc/sslDS"/>

<connection-pool name="ConnPoolTCPS">
  <connection-factory
        factory-class="oracle.jdbc.driver.OracleDriver"
        user="user"
        password="password"
        URL="jdbc:oracle:thin:@(DESCRIPTION=(ADDRESS=(PROTOCOL=tcps) \
             (HOST=host)(PORT=2484)) \
             (CONNECT_DATA=(SERVICE_NAME=orcl)))"
                commit-record-table-name="">
      <property name="truststore"
        value="/path/ewallet.p12"/>
      <property name="truststore-password" value="password"/>
      <property name="truststore-type" value="PKCS12"/>
      <property name="oracle.net.ssl_cipher_suites"
        value="SSL_DH_anon_WITH_3DES_EDE_CBC_SHA, SSL_DH_anon_WITH_RC4_128_MD5, \
        SSL_DH_anon_WITH_DES_CBC_SHA"/>
  </connection-factory>
</connection-pool>
```

16

Oracle Security for Client Connections

This chapter discusses features of Oracle security for clients. These features mainly provide Secure Sockets Layer (SSL) functionality to client HTTP connections in conjunction with functionality of the `HTTPClient` package, and support the use of the standard Java Secure Socket Extension (JSSE). In addition, a section is included that documents the HTTP Client's non-SSL authentication features. This chapter may be of interest for any Java application that is to use SSL, where either OC4J is the Web listener (such as in standalone OC4J), or OC4J is behind Oracle HTTP Server. The following topics are included:

- Using Non-SSL Client Authentication
- HTTPS and Clients
- Overview of Client-Side HTTPS Features
- Supported Default System Properties
- Using HTTPClient with JSSE
- HTTPClient Support for SSL Host Name Verification
- Migrating from Oracle Java SSL to JSSE
- Features for Oracle Java SSL (Deprecated)

Notes:

- In addition to JSSE, Oracle Java SSL is also supported, but is deprecated in the OC4J 10.1.3.1 implementation and will be desupported in future releases. We recommend that you use JSSE. As a step in the Oracle Java SSL deprecation, JSSE is the default SSL implementation for `HTTPClient` in the OC4J 10.1.3.1 implementation. Aside from the last section on Oracle Java SSL, this chapter emphasizes the use of JSSE.
- 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 15, "SSL Communication with OC4J". You can also refer to that chapter for information about "Requesting Client Authentication" on page 15-15.

See Also:

- For general information about JSSE:

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

Using Non-SSL Client Authentication

This chapter primarily focuses on HTTPS for client security. However, there are several HTTP Client APIs that clients can use for non-SSL authentication. The library can be used to create HTTP requests that include basic, digest, and NT LAN Manager (NTLM) authentication information in the request header. This section includes instructions for using the `HTTPClient` package to create these types of requests. For detailed HTTP Client API information, refer to the *Oracle Application Server HTTPClient Java API Reference*.

Basic-Based Client Authentication

The HTTP Client can be used to create an HTTP request to access a server resource that requires basic authentication. The following example demonstrates creating a request and using the `addBasicAuthorization` method to include the realm, username, and password with the request.

```
URL url = new URL("http://localhost:1234");

HTTPConnection client = new HTTPConnection(url);

try {
   client.addBasicAuthorization("realm_name", "user", "password");
   HTTPResponse response = client.Get(url.getFile());
   //assertEquals(200, response.getStatusCode());
}
finally {
   client.stop();
}
```

The realm is a server specified string which groups various URLs under a given server together and which is used to select the correct information when a server issues an authentication challenge. The realm must be the empty string (`""`) for schemes which don't use a realm.

Digest-Based Client Authentication

The HTTP Client can be used to create an HTTP request to access a resource that requires digest authentication. 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 following example demonstrates creating a request and using the `addDigestAuthorization` method to include the realm, username, and password with the request.

```
HTTPConnection client = new HTTPConnection(url);

try {
   client.addDigestAuthorization("ProxyAuthDigestSchemeTestServlet",
      validUserName,validPassword);
   HTTPResponse response = client.Get(url.getFile());
   // assertEquals(200, response.getStatusCode());
```

```
}
finally {
   client.stop();
}
```

The realm is a server specified string which groups various URLs under a given server together and which is used to select the correct information when a server issues an authentication challenge. The realm must be the empty string (`""`) for schemes which don't use a realm.

NTLM-Based Client Authentication

NTLM is a proprietary challenge/response authentication protocol used by Microsoft browsers, proxies, and servers. A client using NTLM is able to prove its identity to a server without sending a password. NTLM is a connection-oriented protocol. Once the connection is authenticated, no further credentials are required as long as the connection remains open.

In NTLM, the NT Domain name qualifies the username. The account identifier is `\`. The NT Domain may be specified in HTTP Client by prefixing the username with the NT Domain name followed by a backslash. For example, for the NT Domain `OPERATIONS` and the username `jsmith`, the fully qualified username is `OPERATIONS\jsmith`.

If no NT Domain is given, the default (if any) is assumed. The default NT Domain is set in the HTTP Client using the system property `HTTPClient.ntlm.defaultDomainName`. If the username is given without an NT Domain, and no default NT Domain is defined in HTTP Client, the NTLM-protected server may assume its own default NT Domain.

To connect to an NTLM-protected resource server, add the NTLM credentials to the HTTP Client `AuthorizationInfo` credential store. As with basic and digest authentication, HTTP Client will automatically query the credential store, when challenged by an NTLM server. Credentials may be added to the credential store either by using an HTTP Connection instance:

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.addNtlmAuthentication( myUsername, myPassword );
```

Or, directly using the `AuthorizationInfo` class:

```
AuthorizationInfo.addNtlmAuthentication( myHost, myPort, myUsername, myPassword )
```

The following example demonstrates creating a request and using the `addNtlmAuthentication` method to include the username and password with the request.

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.addNtlmAuthentication( myUsername, myPassword );
HTTPResponse response = conn.Get( "/index.htm" );
int status = response.getStatusCode();
assertEquals( 200, status );
```

Note: A realm, as specified in authentication schemes such as basic authentication, does not apply to NTLM. The NTLM challenge does not have a realm directive. Therefore, all NTLM credentials are assumed to be part of the same empty (`""`) realm within HTTP Client.

How to Connect to an NTLM-Protected Proxy Server

Proxy servers may also use NTLM for client authentication. However, unlike request-oriented authentication, an NTLM client may only authenticate its connection with the proxy, not the resource server.

To connect to an NTLM-protected proxy server, add the NTLM credentials to the HTTP Client `AuthorizationInfo` credential store. As with basic and digest authentication, the HTTP Client automatically queries the credential store when challenged by an NTLM server. Credentials may only be directly added to the credential store using `AuthorizationInfo`. For example:

```
AuthorizationInfo.addNtlmAuthentication( myProxyHost, myProxyPort, myUsername,
   myPassword)
```

The following example demonstrates a client that uses NTML to connect to a protected proxy server:

```
HTTPConnection conn = new HTTPConnection( myHost, myPort );
conn.setCurrentProxy( myProxyHost, myProxyPort );
AuthorizationInfo.addNtlmAuthentication( myProxyHost, myProxyPort, myUsername,
   myPassword, conn.getContext() )
HTTPResponse response = conn.Get( "/index.htm" );
int status = response.getStatusCode();
assertEquals( 200, status );
```

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 a Web application that initiates 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 HTTPS to secure client interactions with a server.

Oracle client HTTPS functionality is based on the `HTTPConnection` class of the `HTTPClient` package, which provides a complete HTTP client library. The `HTTPConnection` class is used to create new connections that use HTTP, with or without SSL.

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.

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/

- *Oracle Application Server HTTPClient Java API Reference* (Javadoc for the `HTTPClient` packages)

Overview of Client-Side HTTPS Features

Oracle client HTTPS extends the `HTTPConnection` class of the `HTTPClient` package to provide SSL functionality, including cipher suite selection, security credential management with Oracle Wallet Manager, support of security-aware applications, and other features that are described in the following sections. Oracle client HTTPS supports HTTP 1.0 and HTTP 1.1 connections between a client and a server.

`HTTPClient` supports two SSL implementations, JSSE and Oracle Java SSL. The latter is deprecated in the OC4J 10.1.3.1 implementation, however, and will be desupported in future releases. We recommend that you use JSSE.

In addition to the functionality included in the `HTTPClient` package, Oracle client HTTPS supports the following:

- Multiple cryptographic algorithms
- Certificate and key management with Oracle Wallet Manager
- Limited support for the `java.net.URL` framework

In addition, the `HTTPClient` package is used to support:

- HTTPS tunneling through proxies
- HTTP proxy authentication

The following sections describe some of the features:

- Supported Keystore Formats
- Accessing Information for Established SSL Connections
- Support for java.net.URL Framework
- SSL Cipher Suites

Supported Keystore Formats

When using JSSE, you can use a PKCS12 or SSO (auto-login) Oracle wallet with the Oracle JSSE implementation (`OraclePKIProvider`), or a JKS-format keystore with the default Sun Microsystems JSSE implementation. (Oracle Java SSL supports only text-format Oracle wallets.)

For either PKCS12 or SSO wallets, credential information is encrypted. The main difference is that with an SSO wallet, you do not have to present a wallet password to open the wallet at time of access.

JKS and PKCS12 are standard formats; SSO wallets are Oracle-proprietary.

See Also:

- *Oracle Application Server Administrator's Guide* (in the chapter for managing wallets and certificates) for details about creating and using PKCS12 and SSO/auto-login wallets

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.

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 client HTTPS features are supported through system properties only.

Features that are supported only through system properties are:

- 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:

- Javadoc for the `java.net.URL` class, at:

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

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.

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

As of the OC4J 10.1.3.1 release, JSSE supports the following cipher suites, listed in default preference order. In this list, "*" indicates the cipher suite is enabled by default, and "**" indicates the cipher suite requires the installation of the JCE Unlimited Strength Jurisdiction Policy Files from Sun Microsystems. Note that with null encryption, SSL is used only for authentication and data integrity purposes.

Note: `HTTPClient` does not provide a way to selectively enable a subset of supported cipher suites.

```
SSL_RSA_WITH_RC4_128_MD5 *
SSL_RSA_WITH_RC4_128_SHA *
TLS_RSA_WITH_AES_128_CBC_SHA *
TLS_DHE_RSA_WITH_AES_128_CBC_SHA *
TLS_DHE_DSS_WITH_AES_128_CBC_SHA *
SSL_RSA_WITH_3DES_EDE_CBC_SHA *
```

```
SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA *
SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA *
SSL_RSA_WITH_DES_CBC_SHA *
SSL_DHE_RSA_WITH_DES_CBC_SHA *
SSL_DHE_DSS_WITH_DES_CBC_SHA *
SSL_RSA_EXPORT_WITH_RC4_40_MD5 *
SSL_RSA_EXPORT_WITH_DES40_CBC_SHA *
SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA *
SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA *
TLS_RSA_WITH_AES_256_CBC_SHA **
TLS_DHE_RSA_WITH_AES_256_CBC_SHA **
TLS_DHE_DSS_WITH_AES_256_CBC_SHA **
SSL_RSA_WITH_NULL_MD5
SSL_RSA_WITH_NULL_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_256_CBC_SHA **
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
```

See Also:

- For general information about JSSE:

  http://java.sun.com/j2se/1.4.2/docs/guide/security/jsse/JSSERefGuide.html

Supported Default System Properties

This section discusses standard Java system properties supported for keystores and truststores. (These properties offer the only way for users of the `java.net.URL` framework to set security credential information.) Oracle client HTTPS recognizes the following properties:

- Property javax.net.ssl.keyStore
- Property javax.net.ssl.keyStorePassword
- Property javax.net.ssl.keyStoreType
- Property javax.net.ssl.trustStore
- Property javax.net.ssl.trustStorePassword
- Property javax.net.ssl.trustStoreType

Note: You can set Java system properties on the JVM command line for standalone OC4J, or in a Java property setting for the OC4J instance in the `opmn.xml` file in an Oracle Application Server environment. Setting system properties is described in the *Oracle Containers for J2EE Configuration and Administration Guide*, in the chapter on OC4J runtime configuration.

Property javax.net.ssl.keyStore

This property specifies the location and name of the keystore file or wallet file to use as the keystore.

Property javax.net.ssl.keyStorePassword

This property can be set to indicate the password that is necessary to open the keystore (keystore file or wallet file). For example:

```
javax.net.ssl.keyStorePassword=welcome1
```

Important: Storing the keystore 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 an SSO wallet, which does not require a password.
- If a password is necessary, do not store it in a clear text file. As an alternative, you can load the property dynamically before the `HTTPConnection` is started by using the `System.setProperty()` method. Unset the property after the handshake is completed.

Property javax.net.ssl.keyStoreType

This property specifies the type of file used for the keystore. With the Oracle JSSE implementation (`OraclePKIProvider`), you can specify `PKCS12` or `SSO`. With the default Sun Microsystems JSSE implementation, you can specify `JKS`.

See Also:

- The SSL overview in the *Oracle Application Server Administrator's Guide* for discussion of the PKCS12 type

Property javax.net.ssl.trustStore

This property is used similarly to `javax.net.ssl.keyStore`, but specifies the location and name of the keystore file or wallet file to use as the truststore (a file that includes the trusted certificate authorities that a client will implicitly accept).

Property javax.net.ssl.trustStorePassword

This property is used similarly to `javax.net.ssl.keyStorePassword`, but specifies the password that is necessary to open the truststore (keystore file or wallet file).

Property javax.net.ssl.trustStoreType

Similarly to `javax.net.ssl.keyStoreType`, this property specifies the type of file used for the truststore—`PKCS12` or `SSO` for the Oracle JSSE implementation, or `JKS` for the default Sun Microsystems JSSE implementation.

Using HTTPClient with JSSE

This section describes Oracle Application Server support for HTTPS client connections using JSSE, covering the following topics:

- Prerequisites for using JSSE
- Configuring HTTPClient to Use JSSE

See Also:

- *Oracle Application Server Administrator's Guide* (in the chapter for managing wallets and certificates) for information about Oracle Wallet Manager and `orapki` features for creating and maintaining wallets
- For complete information on JSSE:

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

Prerequisites for using JSSE

Note the following requirements to use JSSE with Oracle client HTTPS:

- You must be using Sun Microsystems JDK version 1.2 or higher. (JSSE is included as part of JDK 1.4 or higher.)
- The file `oraclepki.jar`, located in directory *`ORACLE_HOME`*`/network/jlib`, must be in your classpath. (The file `jssl-1_1.jar` or `jssl-1_2.jar`, used by Oracle Java SSL, is no longer necessary. These files are incompatible with JSSE and are no longer included with your installation.)

Configuring HTTPClient to Use JSSE

Oracle Application Server supports HTTPS client connections using JSSE. A client can configure `HTTPClient` to use JSSE as the underlying SSL provider as follows:

1. Create a truststore using the Sun Microsystems `keytool`.

   See Also:

   - For details on using the `keytool`:

     `http://java.sun.com/j2se/1.3/docs/tooldocs/win32/keytool.html`

2. Set the truststore property. A client wishing to use JSSE must specify the client truststore location through the `javax.net.ssl.trustStore` property. 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 16–1 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. There must be a call to the `HTTPConnection` class `connect()` method 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. In the Oracle implementation, this method is called implicitly during calls to HTTP methods, such as the `HTTPConnection` class `Get()` method. In addition, an explicit call to the `connect()` method is useful when the calling application requires SSL session information before sending data. Also see "Verifying Additional Connection Information" on page 16-13.

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 16–1 Using JSSE with HTTPClient

```
public void obtainHTTPSConnectionUsingJSSE() throws Exception
{
   // set the truststore to the location of the client's truststore 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 Oracle Java SSL is not specified as preferred or if Oracle Java SSL classes are not found in the application classpath. If no SSL socket factory is specified and Oracle Java SSL is specified as preferred, and Oracle Java SSL classes *are* found in the classpath, then Oracle Java SSL is used by default. Also see "Specifying Oracle Java SSL as the SSL Implementation for HTTPClient" on page 16-16.
- The JSSE SSL implementation is not thread-safe.

HTTPClient Support for SSL Host Name Verification

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.

A *host name verifier* (`javax.net.ssl.HostnameVerifier` implementation) is used by `HTTPClient` to verify that the host name in the SSL certificate matches the host name of the URI used to access the protected server. This helps detect "man-in-the-middle" attacks. `HTTPClient` invokes the `HostnameVerifier` instance immediately after establishing an SSL session, throwing a `javax.net.ssl.SSLPeerUnverifiedException` if a host name mismatch is detected.

Important: You can provide your own `HostnameVerifier` implementation, or use one provided by Oracle (as described below). The implementation must have a no-argument constructor.

See Also:

- Javadoc for `javax.net.ssl.HostnameVerifier`, available through:

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

The section discusses how to enable and use this feature, covering the following topics:

- Enabling Host Name Verification through System Property Setting
- Enabling Host Name Verification Programmatically
- Using the Oracle Standard Host Name Verifier
- Verifying Additional Connection Information

Without either the system property setting or the programmatic setting described below, host name verification will not be performed.

Enabling Host Name Verification through System Property Setting

To enable host name verification without having to alter your code, set the system property `HTTPClient.defaultHostnameVerifier` to the fully-qualified class name of a host name verifier implementation in the classpath.

The setting must specify the name of an appropriate class (a `javax.net.ssl.HostnameVerifier` implementation with a no-argument constructor).

Enabling Host Name Verification Programmatically

You can use the following methods of the `HTTPConnection` class to enable host name verification programmatically, by specifying the `javax.net.ssl.HostnameVerifier` instance to use for verification:

- `static HostnameVerifier setDefaultHostnameVerifier (HostnameVerifier defaultHostnameVerifier)`

  Use this static method to assign the specified `HostnameVerifier` instance as the JVM default. This method returns the previously set default host name verifier, or `null` if host name verification was previously disabled by default.

- `HostnameVerifier setHostnameVerifier (HostnameVerifier hostnameVerifier)`

  Use this instance method to override the default host name verifier and assign the specified `HostnameVerifier` instance for use by the connection. You can also specify `null`, in order to disable host name verification for the connection.

  This method returns the previous host name verifier for the connection, or `null` if host name verification was previously disabled for the connection.

Using the Oracle Standard Host Name Verifier

Oracle supplies the host name verifier implementation `StandardHostnameVerifier`, in the `HTTPClient` package.

`StandardHostnameVerifier` implements standard host name matching rules for site identity checking, providing the following features:

- It verifies the SSL session host name by checking whether the given host name is the same as the common name (CN) of the distinguished name (DN) from the first certificate in the SSL certificate chain. The comparison is not case-sensitive.
- If wildcard matching is enabled, it will recognize and match the given host name with a wildcard CN in the SSL certificate (for example, `*.oracle.com`), if present.

`StandardHostnameVerifier` has the following methods:

- `boolean setRecognizeWildcardCNs(boolean recognizeWildcardCNs)`

  Specify whether to recognize wildcard CNs. This method returns the previously set value.

- `boolean isRecognizeWildcardCNs()`

  This method tells you whether wildcard CNs are recognized.

- `boolean verify(java.lang.String hostname, javax.net.ssl.SSLSession sslSession)`

Call this method to verify that the host name is an acceptable match with the authentication scheme of the server. (This is standard functionality specified in the `javax.net.ssl.HostnameVerifier` interface.)

You can specify `StandardHostnameVerifier` as your host name verifier either programmatically or through the system property setting, as discussed previously. To set it as the default host name verifier, for example, use the following system property setting:

```
HTTPClient.defaultHostnameVerifier=HTTPClient.StandardHostnameVerifier;
```

Verifying Additional Connection Information

Further validation (beyond host name verification) may be performed using any data found in the `javax.net.ssl.SSLSession` object returned from the `HTTPConnection` class `getSSLSession()` method, after the `HTTPConnection` class `connect()` is called.

To perform this validation, establish a connection to the server without transferring any data, as follows:

```
httpsConnection.connect();
```

After the connection is established, the connection information, in this example 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[0].getSubjectDN().getName();
peerCertDN = peerCertDN.toLowerCase();
```

(The user's certificate is first in the array; the root CA's certificate is last.)

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);
}
```

> **Note:** The preferred mechanism is to use the host name verification facility described in the preceding sections.

Migrating from Oracle Java SSL to JSSE

As noted earlier, in the OC4J 10.1.3.1 implementation and beyond, we recommend that you use JSSE rather than Oracle Java SSL. The latter is deprecated in the 10.1.3.1 implementation and will be desupported in future releases. This section discusses how to modify your client code to use JSSE instead of Oracle Java SSL for your SSL functionality.

Code Samples for Migration to JSSE

This section provides code samples to show the steps for creating a new SSL socket factory using JSSE, compared to equivalent code you would have used previously for Oracle Java SSL. The `java.security.Keystore` class and various `javax.net.ssl` classes are used to open the keystore or wallet and apply Oracle-specific security policies for verifying peer certificates.

See Also:

- Javadoc for `Keystore` and the `javax.net.ssl` classes, available through:

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

Here are the steps, with explanations and comparisons:

1. Register the SSL provider. This step is not necessary if providers are set statically in the `jre/lib/security/java.security` properties file for your JDK.

   Old code for Oracle Java SSL: not applicable

   New code for JSSE:

   ```
   Security.insertProviderAt(new oracle.security.ssl.OraclePKIProvider(), 1);
   ```

   The static `insertProviderAt()` method of the `java.security.Security` class adds a new provider, at a specified preference position. The position refers to the preference order in which providers are searched for requested algorithms, with 1 being most preferred. Use `OraclePKIProvider` for enforcement of Oracle security policies. Once you set the provider, you can use standard JSSE classes for your SSL functionality.

2. Load your keystore, wallet, or trusted certificates.

   Old code for Oracle Java SSL:

   ```
   OracleSSLCredential cred = new OracleSSLCredential();
   cred.loadWallet("walletpath", "password");
   ```

   New code for JSSE:

   ```
   KeyStore myWallet = KeyStore.getInstance("keystoretype","OraclePKI");
   FileInputStream istr = new FileInputStream("pathtowallet");
   myWallet.load(istr, password);
   ```

   The static `getInstance()` method of the `Keystore` class creates a keystore object for the specified keystore type from the specified provider. Use `OraclePKI` as the provider.

   In this sample: *`keystoretype`* is `PKCS12` or `SSO`; *`pathtowallet`* is the path and file name of the keystore or wallet file; and *`password`* is a `char[]` array for the password, or `null` if you use an SSO wallet.

   Note that the `OracleSSLCredential` class, used for Oracle Java SSL, is not used for JSSE.

3. Create the SSL socket factory.

   Old code for Oracle Java SSL:

   ```
   OracleSSLSocketFactory socketFactory = new OracleSSLSocketFactoryImpl();
   SocketFactory.setSSLCredentials(cred);
   ```

New code for JSSE:

```
TrustManagerFactory tmf = TrustManagerFactory.getInstance("OracleX509");
tmf.init(trustCerts);
TrustManager[] tmA = tmf.getTrustManagers();

KeyManagerFactory kmf = KeyManagerFactory.getInstance("OracleX509");
kmf.init(trustCerts, password);
KeyManager[] kmA = kmf.getKeyManagers();

SSLContext ctx = SSLContext.getInstance("SSL");
ctx.init(kmA, tmA, null);
SSLSocketFactory factory = ctx.getSocketFactory();
```

Create and set trust managers so the SSL connection can verify the peer certificate chain. Create and set key managers so the SSL connection can access the user certificate and private key. Then use key managers and trust managers to configure the SSL context, which is used to create new SSL socket factories. Once an `SSLSocketFactory` instance is created, you can use it to create SSL sockets.

Additional Changes Relevant for Migration to JSSE

Note the following additional changes between Oracle Java SSL and JSSE that may affect your application:

- There is a different signature for the `createSocket()` method of the SSL socket factory. (This is an `OracleSSLSocketFactory` instance for Oracle Java SSL, and an `SSLSocketFactory` instance for JSSE.) Note, however, that users of `HTTPClient` are not required to call `createSocket()` directly.

  Old signature for Oracle Java SSL:

  ```
  createSocket(Socket sock)
  ```

  New signature for JSSE:

  ```
  createSocket(Socket sock, String host, int port, boolean autoClose)
  ```

  This method creates a new socket layered over an existing socket. Specify the existing socket, server host, server port, and whether to close the underlying socket when the created socket is closed.

- There is a difference in the peer certificate chain returned by the SSL session object.

  For Oracle Java SSL: The `getPeerCertificateChain()` method of `OracleSSLSession` returns a certificate chain with the root CA certificate first and the peer certificate last.

  For JSSE: The `getPeerCertificates()` method (preferred) or `getPeerCertificateChain()` method (maintained for backward compatibility) of `SSLSession` returns a certificate chain with the peer certificate first and the root CA certificate last.

Features for Oracle Java SSL (Deprecated)

Oracle Java SSL is deprecated in the OC4J 10.1.3.1 implementation but not yet desupported. (It will be desupported in future releases.)

For completeness, this section documents features specific to Oracle Java SSL, but we strongly recommend that you migrate to JSSE, as discussed in "Migrating from Oracle Java SSL to JSSE" on page 16-13.

The following topics are covered here:

- Specifying Oracle Java SSL as the SSL Implementation for HTTPClient
- OracleSSLCredential Class for Oracle Java SSL
- Security-Aware Applications Support in Oracle Java SSL
- Using HTTPClient with Oracle Java SSL
- Specifying Cipher Suites for Oracle Java SSL
- SSL Cipher Suites Supported by Oracle Java SSL

Note: Depending on the JDK you use, Oracle Java SSL requires the file `jssl-1_1.jar` or `jssl-1_2.jar` to be in the classpath.

See Also:

- *Oracle Advanced Security Administrator's Guide* for information about Oracle Java SSL

Specifying Oracle Java SSL as the SSL Implementation for HTTPClient

In the OC4J 10.1.3.1 implementation, the JDK-default JSSE implementation is now the default SSL implementation for `HTTPClient`. (This is a step toward deprecating Oracle Java SSL, the previous default SSL implementation for `HTTPClient`.)

You can, however, explicitly specify Oracle Java SSL as the default SSL implementation for `HTTPClient` by completing the following steps:

1. Specify the following system property setting:

   ```
   HTTPClient.preferOracleSSL=true
   ```

2. Ensure that the Oracle Java SSL classes (for example, `oracle.security.ssl.OracleSSLSocketFactory`) are in your classpath.

Important: If an SSL socket factory (`javax.net.ssl.SSLSocketFactory` implementation) is explicitly specified through the `setSSLSocketFactory()` method of `HTTPConnection`, the SSL implementation associated with the specified factory will be used, regardless of the setting of the `HTTPClient.preferOracleSSL` property.

OracleSSLCredential Class for Oracle Java SSL

To support client HTTPS connections for use with Oracle Java SSL, several methods were added to the `HTTPConnection` class that use the Oracle Java SSL class, `OracleSSLCredential`.

For Oracle Java SSL, security credentials are used to authenticate the server and the client to each other. `OracleSSLCredential` is used to load user certificates and trust points 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 `setSSLCredential()` method of the `HTTPConnection` class.

Security-Aware Applications Support in Oracle Java SSL

Oracle client HTTPS uses SSL to provide security-aware applications support. When security-aware applications do not set trust points, 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 trust point level, they are responsible for authenticating individual certificates below the trust point.

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.

Using HTTPClient with Oracle Java SSL

This section shows an application that uses `HTTPClient` and Oracle Java SSL to connect to a Web server, send a `GET` request, and fetch a Web page.

Sample Code (Oracle Java SSL)

This section contains the sample code using `HTTPClient` and Oracle Java SSL.

```
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 Oracle Java SSL

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.

System Property Features with Oracle Java SSL

"Supported Default System Properties" on page 16-7 discusses the Java system properties `keyStore`, `keyStorePassword`, `keyStoreType`, `trustStore`, `trustStorePassword`, and `trustStoreType`. This section discusses features relating to `javax.net.ssl.keyStore` that are specific to Oracle Java SSL:

- 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.
- 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 as this file is read, then the connection fails and an `IOException` is thrown.
- If this property has no setting, the application is responsible for verifying that the certificate chain contains a certificate that can be trusted.

Specifying Cipher Suites for Oracle Java SSL

This section discusses how to specify cipher suites for Oracle Java SSL.

Property Oracle.ssl.defaultCipherSuites

For Oracle Java SSL, `Oracle.ssl.defaultCipherSuites` 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
```

You can set this property before establishing an SSL connection using Oracle Java SSL. The cipher suites that you specify in this property setting are used as the enabled cipher suites for new HTTPS connections.

See Also:

- Table 16–1, " Cipher Suites Supported by Oracle Java SSL" below

Method setSSLEnabledCipherSuites()

For Oracle Java SSL, you can also set cipher suites per connection by using the following method of the `HTTPConnection` class in package `HTTPClient`:

- `boolean setSSLEnabledCipherSuites(String[] cipherSuites)`

This takes a Java string array, with each array element specifying a cipher suite. It returns a boolean indicating whether the current SSL implementation supports this method.

Note: There is no way to specify cipher suites per HTTP connection in JSSE.

See Also:

- *Oracle Application Server HTTPClient Java API Reference* (Javadoc) for additional information

SSL Cipher Suites Supported by Oracle Java SSL

Oracle Java SSL supports the cipher suites listed in Table 16–1. Note that with NULL encryption, SSL is used only for authentication and data-integrity purposes.

Table 16–1 Cipher Suites Supported by Oracle Java SSL

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

17

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 and Constraint 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 8-11 for information relevant when using Oracle Identity Management as the security provider with Oracle Single Sign-On
- "Introduction to JAAS Mode" on page 5-5 and "Configuring and Using JAAS Mode" on page 5-18 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
- Specifying auth-method in orion-application.xml
- Using Basic Authentication Fallback in Digest Authentication Mode
- Using Client-Cert Authentication
- Using Form-Based Authentication

Specifying auth-method in web.xml

To specify a standard authentication method at the Web application level, use the `<login-config>` element and its `<auth-method>` subelement in `web.xml`. For example:

```
<web-app ... >
```

```
   ...
   <login-config>
      <auth-method>BASIC</auth-method>
      ...
   </login-config>
   ...
</web-app>
```

Table 17–1 shows the standard <auth-method> settings in web.xml.

Table 17–1 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.
CLIENT-CERT	The application requires the client to supply its own HTTPS certificate for use with SSL.

Notes:

- For either the file-based provider or Oracle Identity Management, we recommend digest authentication as a more secure solution than basic authentication.
- 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 8-15.
- 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 17-4. (This section also discusses standard configuration for form-based authentication.)
- To use CLIENT-CERT, you must also configure the OracleAS JAAS Provider property mapping.attribute, as described in "Using Client-Cert Authentication" on page 17-5.

See Also:

- "Web Application Standard Authentication Methods" on page 2-2 for a summary of these authentication methods
- The next section, "Specifying auth-method in orion-application.xml", for Oracle-specific authentication methods

Specifying auth-method in orion-application.xml

Use the <jazn-web-app> element and its auth-method attribute in orion-application.xml to specify an Oracle-specific authentication method at the J2EE application level. Supported settings in the OC4J 10.1.3.1 implementation are "SSO" (for Oracle Single Sign-On), "COREIDSSO" (for Oracle Access Manager single

sign-on), and "CUSTOM_AUTH" (to use the identity management framework, including Java SSO). Refer to "Overview of Oracle Application Server Single Sign-On Alternatives" on page 3-6 for more information about these features. The following example is for Oracle Single Sign-On:

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

Notes:

- The authentication method is set in `orion-application.xml` automatically for SSO alternatives you configure through Application Server Control (Oracle Single Sign-On and Java SSO).
- Any standard authentication method should be configured in the `web.xml` file as described in the preceding section, "Specifying auth-method in web.xml", not in any OC4J-specific file. (This differs in some cases from proprietary functionality in earlier releases, although that functionality is still supported for backward compatibility.)
- 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.
- A setting for `auth-method` in `orion-application.xml` (or `orion-web.xml`) overrides any setting in `web.xml`.

See Also:

- "Overview of Oracle Application Server Single Sign-On Alternatives" on page 3-6 for a summary of Oracle-specific authentication methods

Using Basic Authentication Fallback in Digest Authentication Mode

In the OC4J 10.1.3.1 implementation, the default behavior is that the digest authentication module does *not* handle basic authentication, even if the client sends a basic authentication header. (This is different from the default behavior in the 10.1.3.0.0 implementation.) To enable basic authentication fallback in the event that the client sends a basic authentication header, set the `digest.auth.basic.fallback` property to "`true`" in a `<property>` subelement of the `<jazn>` element in `orion-application.xml`, as in the example below. (Note that the logic of this property is the reverse of what it was in the 10.1.3.0.0 implementation.)

```
<jazn provider="XML" location="jazn-data.xml">
   <property name="digest.auth.basic.fallback" value="true" />
   ...
</jazn>
```

To avoid using basic authentication fallback, either do not set this property, or explicitly set it to "`false`".

Important: If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any prior settings within the `<jazn>` element would be lost and would have to be redone.

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

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

OC4J supports the property `oc4j.formauth.redirect` for client-side redirects. If you set this 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.

> **Note:** To use client certificates with Oracle Single Sign-On, follow procedures outlined in the *Oracle Application Server Single Sign-On Administrator's Guide* rather than procedures here.

Configuring OC4J for Client-Cert Authentication

To use client authentication through HTTPS:

1. Configure SSL, as described in Chapter 15, "SSL Communication with OC4J".
2. Set `<auth-method>` to `CLIENT-CERT` in `web.xml`, as described in "Specifying auth-method in web.xml" on page 17-1.
3. Set the OC4J `mapping.attribute` property in the `<jazn>` element of the application `orion-application.xml` file, as necessary.
4. 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 `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="mapping.attribute" value="CN"/>
      ...
   </jazn>
   ...
</orion-application>
```

Important:

- In standalone OC4J (no Oracle HTTP Server), setting `needs-client-auth="true"` in the `<ssl-config>` element in `secure-web-site.xml` is required in order to use client-cert authentication mode. This attribute is discussed in "Optional Steps in secure-web-site.xml" on page 15-7.
- If you switch from the file-based provider to Oracle Identity Management at any time for any application through Application Server Control, the `<jazn>` element in `orion-application.xml` for the application is replaced with the following. Any prior settings within the `<jazn>` element would be lost and would have to be redone.

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

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 `mapping.attribute`, OracleAS JAAS Provider retrieves the appropriate value from the certificate user's 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 `mapping.attribute` determines the search filter. If the attribute setting is "`CN`", for example, the common name is the search filter 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 and Constraint Configuration

This section describes role types and how they are mapped together:

- Configuring J2EE Roles and Security Constraints
- Linking Application Roles to J2EE Roles
- Definition of Deployment Roles and Users

- Specifying a Run-As Security Identity for a Web Application
- OC4J Mapping of J2EE Roles to Deployment Roles

Configuring J2EE Roles and Security Constraints

J2EE includes a feature for portable security roles defined in the standard deployment descriptor `web.xml` for servlets and JavaServer Pages. These J2EE roles are used in defining a set of resource access permissions for an application. For example, assume you configure a J2EE role called `sr_developers` in `web.xml`. This is accomplished using a `<security-role>` element (a subelement of the top-level `<web-app>` element):

```
<web-app>
   ...
   <security-role>
      <role-name>sr_developers</role-name>
   </security-role>
   ...
</web-app>
```

You also define the access permissions for the `sr_developers` role in `web.xml`. A role is tied to capabilities and constraints through additional standard descriptor elements, such as under the `<security-constraint>` element (also a subelement of `<web-app>`):

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

Linking Application Roles to J2EE Roles

Associating a principal or logical role defined in your application code with a J2EE role assigns the defined access permissions of the J2EE role to that principal or application logical role. You can accomplish this in the `web.xml` file so that you do not have to update application code to change role definitions. Use the `<security-role-ref>` element (a subelement of the `<servlet>` element) to link an application role to a J2EE role:

```
<web-app>
   ...
   <servlet>
      ...
      <security-role-ref>
         <role-name>ar_developers</role-name>
         <role-link>sr_developers</role-link>
      </security-role-ref>
      ...
   </servlet>
```

```
   ...
</web-app>
```

The <role-name> element indicates a role in the application code. The <role-link> element specifies that this application role (ar_developers) should be linked to a J2EE role (sr_developers) defined previously in a <security-role> element.

In this example, if a servlet running as a user belonging to sr_developers invokes the HttpServletRequest method isUserInRole("ar_developers"), the method will return true.

> **Note:** 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 Deployment Roles and Users

Deployment roles and users are defined in the security provider that you use. For the file-based provider, for example, deployment users and roles are defined in the system-jazn-data.xml file (or, optionally, a user-supplied jazn-data.xml file).

The following configures a developers deployment role:

```
<jazn-data>
   ...
   <jazn-realm>
      ...
      <realm>
         ...
         <roles>
            ...
            <role>
               <name>developers</name>
               <members>
                  <member>
                     <type>user</type>
                     <name>john</name>
                  </member>
               </members>
            </role>
            ...
         </roles>
         ...
      </realm>
      ...
   </jazn-realm>
   ...
</jazn-data>
```

Specifying a Run-As Security Identity for a Web Application

There are situations where a Web container must allow access to users who are not known by the container. In such situations, the standard Web application descriptor may declare a <run-as> setting to specify the role to be allowed access:

```
<servlet>
   ...
```

```
   <run-as>
      <role-name>sr_developers</role-name>
   </run-as>
   ...
</servlet>
```

The role name must be one of the roles already defined for the Web application. The Web container must propagate the specified identity for any call from a servlet to the EJB layer.

See Also:

- "Run-As Mode and Propagated Identities in Web Applications" on page 2-3

OC4J Mapping of J2EE Roles to Deployment Roles

OC4J enables you to map J2EE roles defined in the `web.xml` file to deployment roles in the security provider. You can accomplish this through Application Server Control during deployment, as described in "Specifying Security Role Mapping through Application Server Control" on page 6-11. 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). In `orion-application.xml`, `<security-role-mapping>` is a subelement of the top-level `<orion-application>` element. Similarly, in `orion-web.xml`, it is a subelement of the top-level `<orion-web-app>` element.

A `<group>` subelement under a `<security-role-mapping>` element corresponds to a role in the security provider.

The following configuration in `orion-application.xml` would map the J2EE role `sr_developers` (defined in `web.xml`) to the deployment role `developers` (defined in `system-jazn-data.xml` for the file-based provider, for example):

```
<orion-application>
   ...
   <security-role-mapping name="sr_developers">
      <group name="developers" />
   </security-role-mapping>
   ...
</orion-appliction>
```

This association permits the `developers` role to access resources that are configured in `web.xml` to be accessible for the `sr_developers` 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_developers`, `john` has access to application resources that are made available to the `sr_developers` role.

18

EJB Security Configuration

This chapter discusses security issues affecting EJBs, covering the following topics:

- Authenticating and Authorizing EJB Applications
- Specifying Credentials in EJB Clients
- Permitting EJB RMI Client Access
- Granting Permissions in the Browser
- Configuring Anonymous EJB Lookup
- Enabling and Configuring Subject Propagation for ORMI

Note that beginning with OC4J 10.1.3.x implementations, the EJB container supports the OracleAS JAAS Provider.

See Also:

- *Oracle Containers for J2EE Enterprise JavaBeans Developer's Guide* for general information about EJBs and information about EJB 3.0 security annotations
- *Oracle Containers for J2EE Services Guide* for information about ORMI
- "Enabling ORMIS for OC4J" on page 15-18 (in this document) for information about ORMIS
- "JAAS Authorization and OracleAS JAAS Provider JAAS Mode" on page 5-4 and "Configuring and Using JAAS Mode" on page 5-18 for information about JAAS mode, which can be used with EJB applications
- Chapter 19, "Common Secure Interoperability Protocol" for information about CSIv2, used in conjunction with EJBs
- The following Web site for OC4J "how-to" examples:

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

Authenticating and Authorizing EJB Applications

You can define security constraints and J2EE roles in the standard EJB deployment descriptor to protect your EJB methods. These J2EE roles can be linked to roles you define in your application, and then mapped to deployment roles in the security provider, as appropriate. (In the case of EJBs, a deployment role may correspond to a role in the backend database that the EJB will access, for example.) The mapping to

deployment roles can be accomplished through Application Server Control during deployment, as described in "Specifying Security Role Mapping through Application Server Control" on page 6-11, and results in appropriate configuration in the OC4J-specific deployment descriptors.

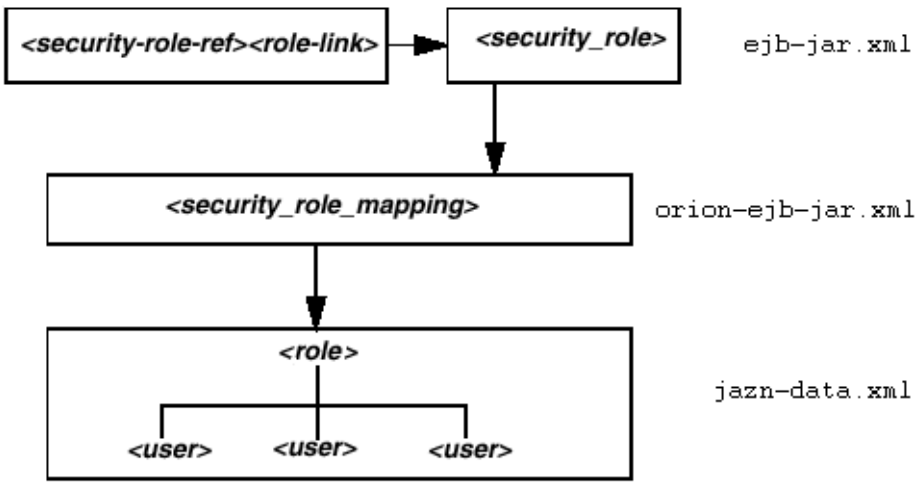
For authentication and authorization, this section focuses on XML configuration within the EJB deployment descriptors. EJB authorization is managed as follows:

- The standard EJB deployment descriptor describes access rules using J2EE logical roles.
- The OC4J-specific deployment descriptor maps the J2EE roles to deployment users and roles (configured in `system-jazn-data.xml`, `jazn-data.xml`, or Oracle Internet Directory, for example).

> **Note:** RMI lookup authentication is integrated with JAAS custom login modules. Refer to Chapter 9, "Login Modules" for information about login modules.

Figure 18–1 provides an overview of EJB role definitions and role mapping (in this case for the file-based provider).

Figure 18–1 End-to-End Security Role Configuration



The steps for EJB authorization are described in the following sections:

- Specifying J2EE Roles and Method Permissions in the EJB Deployment Descriptor
- Specifying Unchecked Security for EJB Methods
- Specifying a Run-As or Caller Security Identity for an EJB
- Mapping J2EE Roles to Deployment Users and Roles
- Configuring Namespace Access
- Specifying a Default Role Mapping for Unidentified Methods

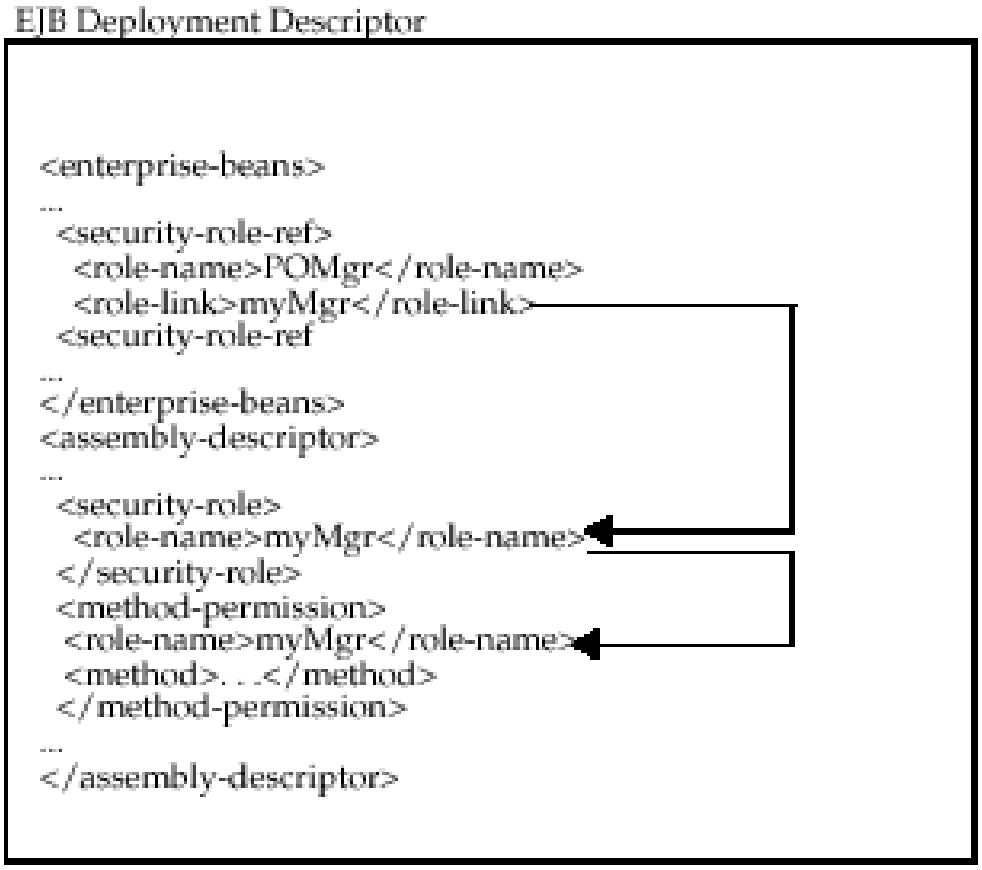
Troubleshooting Tips:

- For an application to access EJBs using RMI, you must grant RMI permission "login" to the appropriate user or role. Refer to "Permitting EJB RMI Client Access" on page 18-10.
- If an application contains an EJB, remote clients must be granted access for read (lookup) and write (bind) operations on the server-side JNDI context of the application, as required. Refer to "Configuring Namespace Access" on page 18-8.

Specifying J2EE Roles and Method Permissions in the EJB Deployment Descriptor

As shown in Figure 18–2 below, you can specify the name of a role (such as `POMgr`) that is defined within your bean implementation, and link this name to the appropriate J2EE user or role (such as `myMgr`) that is defined in the standard EJB deployment descriptor. (The next step, mapping to a deployment role, is reflected in the OC4J-specific deployment descriptor, as discussed in "Mapping J2EE Roles to Deployment Users and Roles" on page 18-6.)

Figure 18–2 Security Role References



The following steps describe this in more detail:

1. Declare the application logical role, such as `POMgr` above, with a `<role-name>` subelement of `<security-role-ref>` in the `<enterprise-beans>` section in

the standard EJB deployment descriptor. (For this example, assume this role has purchase order authority. A caller would have to be mapped into this role, as confirmed by an `isCallerInRole()` call, to work with a purchase order.)

Use a `<role-link>` subelement of `<security-role-ref>` to link the application role to a desired J2EE logical role (which you will define in the next step, also in the standard EJB deployment descriptor). This functionality allows you to use your application in various J2EE environments without changing the bean code. The application role `POMgr` is linked to the J2EE role `myMgr`:

```
<enterprise-beans>
  ...
  <security-role-ref>
   <role-name>POMgr</role-name>
   <role-link>myMgr</role-link>
  </security-role-ref>
  ...
</enterprise-beans>
```

(The J2EE role in the `<role-link>` setting can be the same as a deployment role, or it can be mapped to a deployment role in a later step.)

> **Note:** The `<security-role-ref>` element is required only when you use security context methods within your bean.

2. Define the J2EE role and the EJB methods for which it has permissions in the standard EJB deployment descriptor. In this purchase order example, assume any method executed within a bean `PurchaseOrder` must have authorized itself as `myMgr`, a J2EE role that is declared through a `<role-name>` subelement of `<security-role>`. This is the J2EE role that was linked to the application role `POMgr` in the previous step. Note that `PurchaseOrder` is the name declared in the `<ejb-name>` element, a subelement of the `<session>` or `<entity>` element.

   The following example defines the role `myMgr` and gives it permission to access all methods (as indicated by the "`*`" symbol) of the EJB `PurchaseOrder` bean:

```
<assembly-descriptor>
 ...
 <security-role>
  <description>Role for 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 OC4J maps `POMgr` to `myMgr`.

> **Note:** If you specify 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.

Looking more closely at the <method-permission> element: the <method> subelement 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 take one of the following forms:

- 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>
```

- 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.

- 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>java.lang.String</method-param>
      <method-param>java.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, use the standard EJB deployment descriptor to 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 empty `<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 a Run-As or Caller Security Identity for an EJB

You can specify in the standard EJB deployment descriptor 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 in the `<enterprise-beans>` section. The following example indicates 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 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 J2EE Roles to Deployment Users and Roles

As noted earlier, you can define J2EE roles and related security constraints in the standard EJB deployment descriptor to protect your EJB methods. These J2EE roles can then be mapped to deployment users and roles that are defined in the security

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 6-11.

Mappings are reflected in `<security-role-mapping>` settings in Oracle-specific descriptors, as shown in the discussion that follows.

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

While we recommend that you use Application Server Control for role mapping, the following discussion provides reference information for the resulting configuration in `orion-ejb-jar.xml` when you map the J2EE role `myMgr` to the deployment role `managers`. Any user that can log in as part of the `managers` role is considered to have permissions of the `myMgr` role (which had previously been linked to the `POMgr` application logical role), and can execute the methods of the `PurchaseOrder` bean.

In the standard EJB deployment descriptor:

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

In the OC4J-specific deployment descriptor:

```
<assembly-descriptor>
 ...
 <security-role-mapping name="myMgr">
  <group name="managers" />
 </security-role-mapping>
 ...
</assembly-descriptor>
```

Alternatively, for mapping to a specific user:

```
<security-role-mapping name="myMgr">
 <user name="guest" />
</security-role-mapping>
```

For mapping to a specific user within a specific role:

```
<security-role-mapping name="myMgr">
 <group name="managers" />
 <user name="guest" />
</security-role-mapping>
```

Configuring Namespace Access

If an application contains an EJB, remote clients must be given namespace access to read (look up) and write (bind) objects as required on the server-side JNDI context of the application. "Read" and "write" correspond to the `lookup()` and `bind()` methods of a `javax.naming.Context` object, respectively.

The remote client's user credentials (the JNDI properties passed to the remote client context) should map to one of the roles that are granted access to the JNDI context of the application.

The following example, which would appear in `orion-application.xml`, shows how the namespace access is granted for read operations to roles named `managers` and `developers`.

```
<orion-application ... >
   ...
   <namespace-access>
      <read-access>
         <namespace-resource root="">
            <security-role-mapping name="sr_developer">
               <group name="developers"/>
            </security-role-mapping>
            <security-role-mapping name="myMgr">
               <group name="managers"/>
            </security-role-mapping>
         </namespace-resource>
      </read-access>
   </namespace-access>
   ...
</orion-application>
```

This assumes the indicated role mappings had already been set up elsewhere in `orion-application.xml`.

Specifying a Default Role Mapping for Unidentified 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:

```
<assembly-descriptor>
   ...
   <default-method-access>
      <security-role-mapping name="&lt;default-ejb-caller-role&gt;"
                             impliesAll="true" />
   </default-method-access>
   ...
</assembly-descriptor>
```

The default role is `<default-ejb-caller-role>`, specified 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. A "`true`" setting indicates that no security role checking occurs. A "`false`" setting indicates that the container will check for this default role on these methods.

In the `orion-ejb-jar.xml` file, the `impliesAll` attribute has defaults as follows:

- If <security-role-mapping> is specified in orion-ejb-jar.xml but impliesAll is not set, then this attribute defaults to "false" and the container checks for this default role on these methods.
- If <security-role-mapping> is not specified in orion-ejb-jar.xml, the OC4J EJB layer defaults to a "true" setting for impliesAll and no security role checking occurs for 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> or <group> subelement. 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 in client.jar (standard J2EE client module).
- Servlets or JavaBeans running within the container pass their credentials within the InitialContext, which is created to look up the remote EJBs.

> **Note:** RMI lookup authentication is integrated with JAAS custom login modules. Refer to Chapter 9, "Login Modules" for information about login modules.

Credentials in JNDI Properties

When you access EJBs in a remote container, you must pass valid credentials to the container. Standalone clients define their credentials in the jndi.properties file deployed with the client's code, using the following properties:

```
java.naming.security.principal=username
java.naming.security.credentials=password
```

For example, if you want to access remote EJBs as POMGR/welcome, set the properties as follows. 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");
```

Important: The `jndi.properties` file must be accessible from the classpath.

Credentials in the InitialContext

JavaBeans running within the container pass their credentials within the `javax.naming.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`.

Permitting EJB RMI Client Access

For an application to access EJBs using RMI, you must grant RMI permission "login" to the appropriate user or role. You can accomplish this through the OracleAS JAAS Provider Admintool.

The following example sets this permission for a role (`users`):

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

And this example sets the permission for a user (`JDOE_ENDUSER`):

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

For the file-based provider, you can also grant this permission to a role 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.)

Restart OC4J for changes to take effect.

See Also:

- Appendix C, "OracleAS JAAS Provider Admintool Reference"
- "Policy Configuration in system-jazn-data.xml" on page 5-15

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";
```

Configuring Anonymous EJB Lookup

Anonymous EJB lookup is a mode you may consider, presumably only during 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 generally discourages this practice, 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 Accounts" on page 4-11.
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 "Permitting EJB RMI Client Access" on page 18-10. 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 (lookup) and write (bind) operations on the server-side JNDI context of the application. 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="jndi-user-role">
               <group name="administrators" />
               <group name="users" />
            </security-role-mapping>
         </namespace-resource>
      </read-access>
      <write-access>
         <namespace-resource root="">
            <security-role-mapping name="jndi-user-role">
               <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.

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

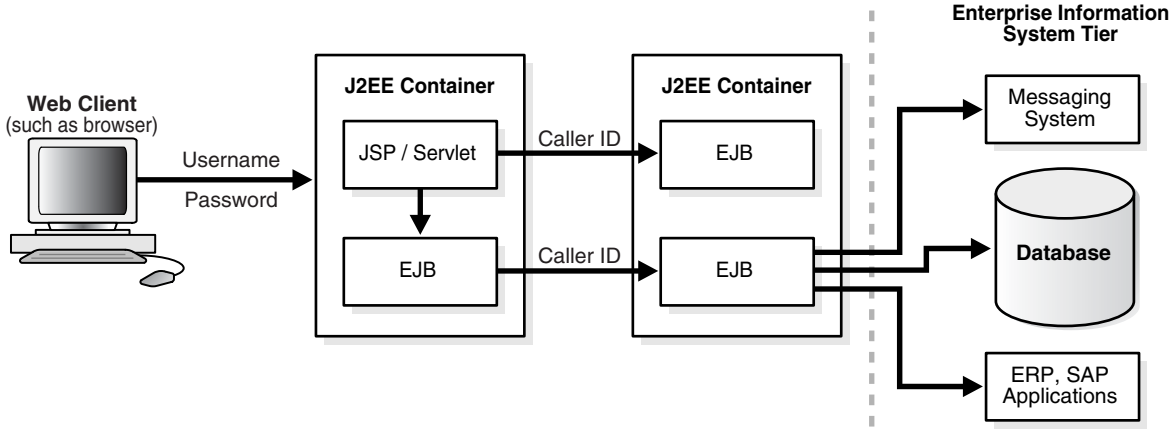
Important:

- 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 15-22), 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 18-15).
- Subject propagation is supported only between OC4J 10.1.3.x instances.

Overview of Subject Propagation in OC4J

OC4J supports subject propagation, as summarized in Figure 18–3. 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 subject (`javax.security.auth.Subject` instance). Similarly, a remote EJB fat client can use this feature in calling to the EJB container.

Figure 18–3 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.

Enabling Subject Propagation for ORMI

The following steps are required in order to use subject propagation:

1. Set the Subject Propagation System Property
2. Enable JAAS Mode
3. Grant RMI Permission for Subject Propagation

Set the Subject Propagation System Property

In OC4J, you can use subject propagation with ORMI if you specifically enable it on the client and server. (It is always enabled for IIOP, in accordance with the CSIv2 specification.) You can accomplish this with the following system property setting on the client and on the server:

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

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

Enable JAAS Mode

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 `<jazn>` element of the `orion-application.xml` file, as discussed in "Introduction to JAAS Mode" on page 5-5.

Grant RMI Permission for Subject Propagation

A propagated subject is accepted by the server only if the `RMIPermission subject.propagation` is granted to the EJB caller. The following example uses the OracleAS JAAS Provider Admintool to grant this permission to the user `oc4jadmin`, so that this user will have its subject propagated to the server:

```
% java -jar jazn.jar -grantperm myrealm -user oc4jadmin \
       com.evermind.server.rmi.RMIPermission subject.propagation
```

You can also grant `subject.propagation` permission to a role. With the following example, any user in the `users` role would have its subject propagated to the server:

```
% java -jar jazn.jar -grantperm myrealm -role users \
       com.evermind.server.rmi.RMIPermission subject.propagation
```

You can restrict subject propagation by specifying the principal names that the server will accept in the subject. The following example again grants `subject.propagation` permission to any user in the `users` role, but the server will filter out all but the `developer` and `manager` principals from propagated subjects. (Note that the filtering is according to principal names, not principal types.)

```
% java -jar jazn.jar -grantperm myrealm -role users \
       com.evermind.server.rmi.RMIPermission subject.propagation developer,manager
```

By default, there is no restriction—all principals in the subject are accepted. But you can also explicitly specify this with the `subject.propagation` parameter `"*"` (including the quotes):

```
% java -jar jazn.jar -grantperm myrealm -user oc4jadmin \
       com.evermind.server.rmi.RMIPermission subject.propagation "*"
```

This is equivalent to the first example above.

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, as documented in "Tasks to Share a Library" on page 6-14.

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 `<host-access>` or `<ip-access>` subelements (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 `<host-access>` or `<ip-access>` subelements (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"/>
```

19

Common Secure Interoperability Protocol

OC4J supports the Common Secure Interoperability Version 2 protocol (CSIv2), an Object Management Group (OMG) standard that can be used in conjunction with EJBs for a secure interoperable wire protocol that supports authorization and identity delegation.

CSIv2 specifies different conformance levels; OC4J complies with the EJB specification, which requires conformance level 0.

There are three files relevant to CSIv2 configuration:

- `internal-settings.xml` (server side)
- `ejb_sec.properties` (client side)
- `orion-ejb-jar.xml`

This chapter is organized as follows:

CSIv2 Security Properties in internal-settings.xml (EJB Server)

Specify server security properties for CSIv2 in the file `internal-settings.xml`, using attribute values within `<sep-property>` elements.

Table 19–1 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 19–1 EJB Server Security Properties

Property	Description
port	IIOP port number (defaults to `5555`).
ssl	A `true` setting indicates IIOP/SSL is supported.
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`.

Table 19–1 (Cont.) EJB Server Security Properties

Property	Description
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	Password for keystore (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	Password for truststore (used only if `ssl` is `true`).

To use the CSIv2 protocol with OC4J, you must both set `ssl` to `true` and specify an IIOP/SSL port (`ssl-port`). Note the following:

- If you do not set `ssl` to `true`, 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 your choice of the following:

- 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 authentication must specify both a keystore and a truststore. The certificate from the keystore is used for client authentication.

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>
```

Notes:

- You cannot update `internal-settings.xml` through Application Server Control.
- Although here the default value of `port` is one less than the default value for `ssl-port`, this relationship is not required.
- If OC4J is started by the Oracle Process Manager and Notification Server (OPMN) in an Oracle Application Server environment, the ports specified in `internal-settings.xml` are overridden. Note that IIOP SSL ports may fail to start if the keystore location, truststore location, or either 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 any setting to enable IIOP through `internal-settings.xml` is overridden. (Refer to the *Oracle Containers for J2EE Services Guide* RMI chapter for information about enabling or disabling IIOP through OPMN.)
- 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.

Here is the DTD for `internal-settings.xml`:

```
<!-- A server extension provider that is to be plugged in to the server. -->
<!ELEMENT server-extension-provider (sep-property*) (#PCDATA)>
<!ATTLIST server-extension-provider name class CDATA #IMPLIED>
<!ELEMENT sep-property (#PCDATA)>
<!ATTLIST sep-property name value CDATA #IMPLIED>
<!-- This file contains internal server configuration settings. -->
<!ELEMENT internal-settings (server-extension-provider*)>
```

CSIv2 Security Properties in ejb_sec.properties (EJB Client)

Any client, whether running inside a server or not, has EJB security properties. Table 19–2 following 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 19–2 EJB Client Security Properties

Property	Description
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.
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.

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 subject and send the user name and password. You must also set `server.trustedhosts` to include your server.

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`. Servers list trusted clients in the `trusted-client` property of the `<sep-property>` element in `internal-settings.xml`, discussed in "CSIv2 Security Properties in internal-settings.xml (EJB Server)" on page 19-1.

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.

OC4J supports both kinds of trust. Configure trust using the `<ior-security-config>` element in `orion-ejb-jar.xml`, discussed in the next section, "CSIv2 Security Properties in orion-ejb-jar.xml".

Notes:

- Server-side authentication takes precedence over a user name and password.
- You can also configure the server to both require SSL client authentication and specify a list of trusted client (or intermediate) hosts that are allowed to insert identity assertions.

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 the bean neither supports nor uses that feature; `supported` means the bean permits the client to use the feature; `required` means 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>
```

20

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
- Understanding Component-Managed Sign-On
- Understanding Container-Managed Sign-On
- Using Declarative Container-Managed Sign-On
- Using Programmatic Container-Managed Sign-On

See Also:

- *Oracle Containers for J2EE Resource Adapter Administrator's Guide* for general information about resource adapters and the J2EE Connector Architecture

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 JAAS subject 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
- Summary of Component-Managed Versus Container-Managed Sign-On

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 subject directly from a J2EE component to a resource adapter (for component-managed sign-on)
- Propagating a 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 subject that differs from the subject 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.

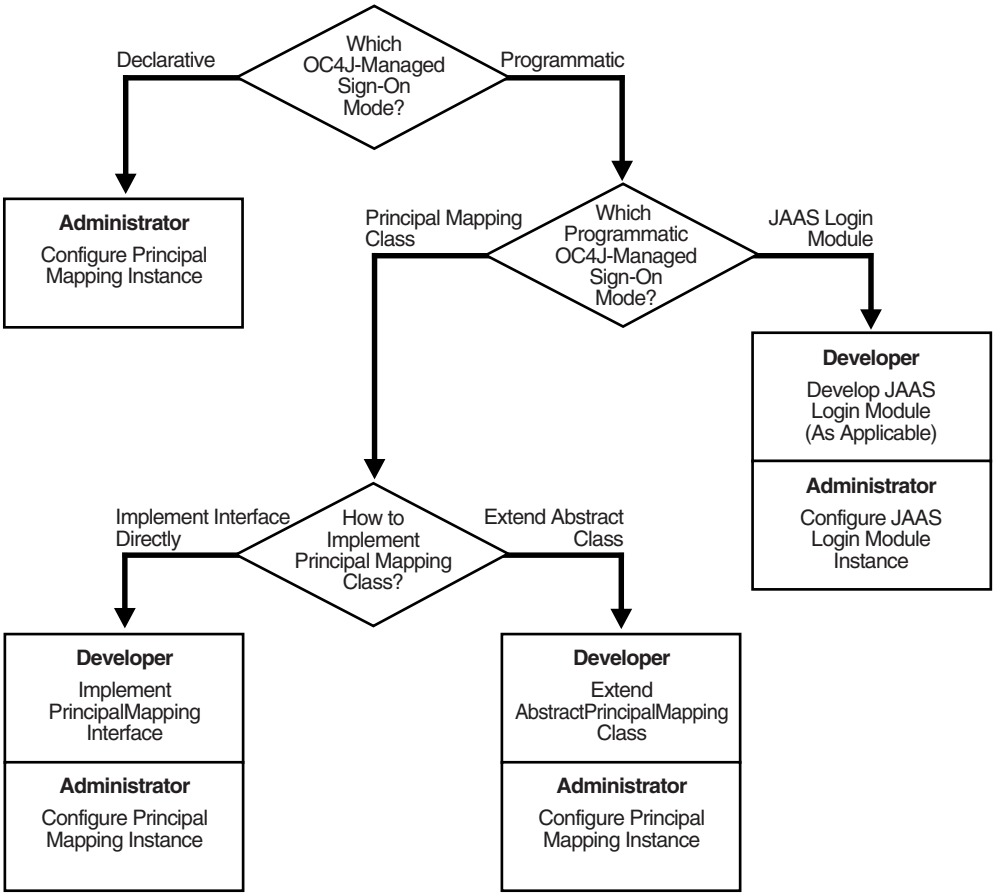
The following list summarizes the options and the type of setup required for component-managed and container-managed sign-on. Bullets at each level represent choices.

- Component-managed sign-on: Requires a`<res-auth>` setting of `Application` in `web.xml` or `ejb-jar.xml`. Setup for sign-on is programmatic. There is no OC4J-specific configuration.
- Container-managed sign-on: Requires a `<res-auth>` setting of `Container` in `web.xml` or `ejb-jar.xml`. 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 20-9); reflected as `use="none"` in the `<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 also described in "Using Declarative Container-Managed Sign-On"); reflected as `use="principal-mapping-entries"` with appropriate subelements under the `<security-config>` element.
- Programmatic, using either a principal mapping class or a JAAS login module:
  * Principal mapping class: Implement the `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 under the `<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 under the `<security-config>` element.

Choices for container-managed sign-on in OC4J are also illustrated in Figure 20–1 that follows.

Figure 20–1 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

See Also:

- *Oracle Containers for J2EE Resource Adapter Administrator's Guide* for additional information about the files and elements discussed here

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 20-9. This section discusses only the syntax for the <security-config> element.

A <security-config> element contains one of the following:

- <principal-mapping-entries> element, specifying user names and passwords explicitly
- <principal-mapping-interface> element, specifying the name of the mapping class
- <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 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

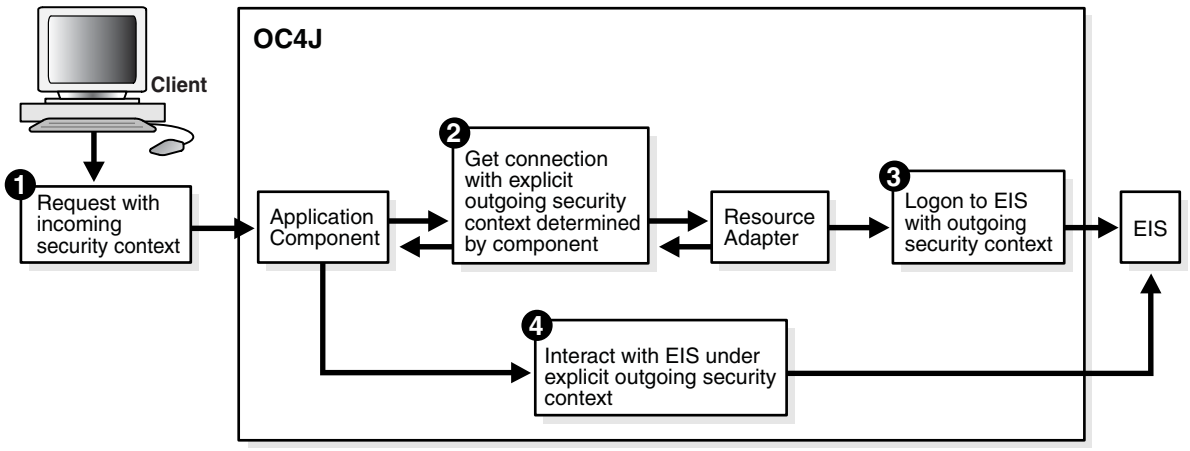
When deploying an application that is to manage its EIS sign-on, use a `<res-auth>` setting of `Application` in the appropriate descriptor file (`web.xml` for a Web component or `ejb-jar.xml` for an EJB component). The application component is then responsible for providing explicit security information 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 20–2 shows the steps in component-managed sign-on, with the text that follows providing further detail.

Figure 20–2 Component-Managed Sign-On



1. The client makes a request, which is associated with an incoming subject (security context) for the initiating principal.
2. As part of servicing the request, the application component maps the incoming subject to an outgoing subject for the resource principal, or hard-codes an outgoing subject, then uses the outgoing subject 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 subject provided by the application component.
4. Once the connection is acquired, the application component can interact with the EIS under the established outgoing subject.

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

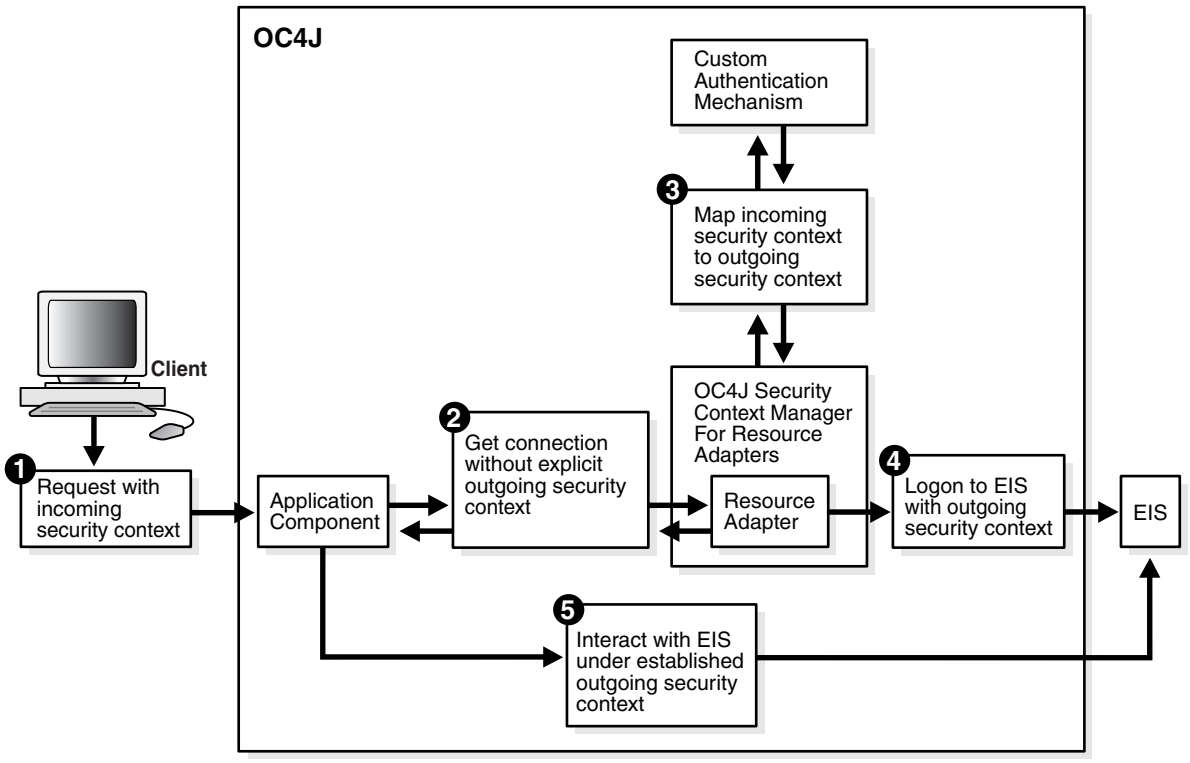
When deploying an application that is to depend on OC4J to manage EIS sign-on, use a `<res-auth>` setting of `Container` in the appropriate descriptor file (`web.xml` for a Web component or `ejb-jar.xml` for an EJB component). OC4J is then responsible for providing security information for the sign-on. 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 20-9. 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 subject and to perform authentication.

Figure 20–3 following illustrates the steps in container-managed sign-on. These steps are detailed following the diagram.

Figure 20–3 Container-Managed Sign-On



1. The client makes a request, which is associated with an incoming subject (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 the figure) maps the incoming subject to the outgoing subject 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 subject provided by OC4J.
5. Once the connection is acquired, the application component can interact with the EIS under the established outgoing subject.

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 you use 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.

Notes: 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.

1. From the **Connection Factories** tab of the appropriate Resource Adapter page, choose the connection factory you want to edit. Connection factories are listed by JNDI location.
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).

      See Also:

      - "Using Password Indirection" on page 6-1

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.
6. Choose **Apply** to apply the changes.

Table 20–1 following summarizes how these settings correspond to XML entities in the `oc4j-ra.xml` file. An example follows the table.

Table 20–1 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.
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 an OracleAS JAAS Provider default realm, as discussed in "Default Realm with the File-Based Provider or Oracle Identity Management" on page 6-4. 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 20–2 following describes how OC4J uses methods of the `PrincipalMapping` interface.

Table 20–2 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 20-15.) The implementation class can use the properties to set either a default user name and password, information for an LDAP connection, or a default mapping.
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 20–3 that follows.

Table 20–3 Method Descriptions for AbstractPrincipalMapping Class

Method Signature	Description
abstract void init (java.util.Properties prop)	The subclass must implement the `init()` method. See Table 20–2, " Method Descriptions for PrincipalMapping Interface" for a description.
void setManagedConnectionFactory (ManagedConnectionFactory mcf)	The subclass need not implement the `setManagedConnectionFactory()` method. See Table 20–2 for a description.
void setAuthenticationMechanisms (java.util.Map authMechanisms)	The subclass need not implement the `setAuthenticationMechanisms()` method. See Table 20–2 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 20–2 for a description.
ManagedConnectionFactory getManagedConnectionFactory()	The `getManagedConnectionFactory()` utility method returns the `ManagedConnectionFactory` instance (for connections to the EIS), which may 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 20–2 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, 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 20–1 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 20-15.

Example 20–1 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, as discussed in "Using Password Indirection" on page 6-1.

Using a JAAS Login Module for an EIS Connection

Alternatively, you can manage sign-on to an EIS programmatically through JAAS.

See Also:

- Chapter 9, "Login Modules"

OC4J furnishes a JAAS pluggable authentication framework that conforms to the Connector Architecture 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 types of JAAS login modules:

- Principal mapping login module
- Credential mapping login module
- Kerberos login module (for caller impersonation)

The 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 (essentially, if there is an anonymous user). The login method of the 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 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 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 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.

21

Configuring Windows Native Authentication

This chapter provides instructions for setting up Windows Native Authentication (WNA) with OC4J to allow transparent authentication for Web application clients. Users must be familiar with Kerberos as well as Active Directory to complete the instructions in this chapter.

The following topics are covered:

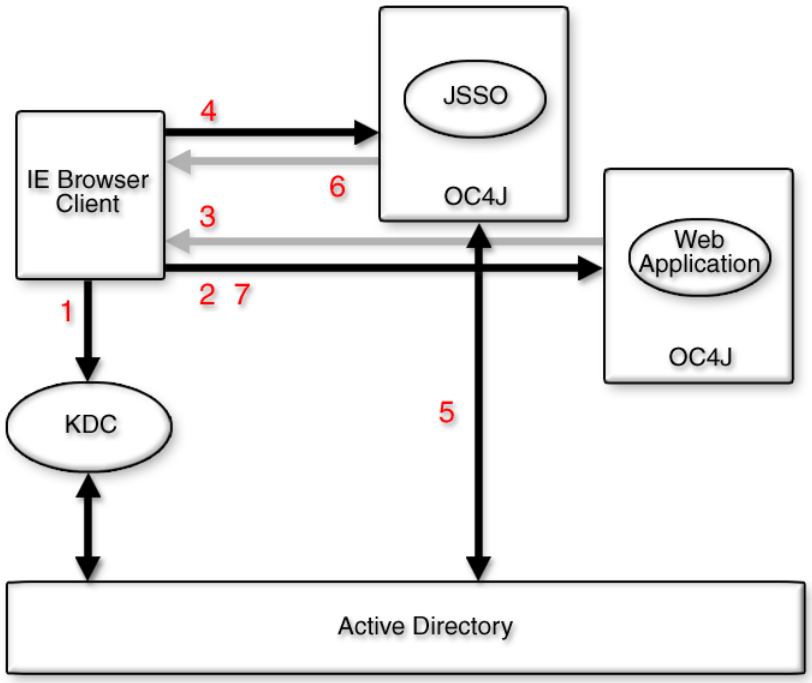
Overview of WNA

WNA allows desktop clients using Microsoft Internet Explorer (or any browser that supports Microsoft's SPNEGO protocol) to automatically authenticate with Web applications using their Windows desktop credentials. OC4J Web applications can be configured to use WNA instead of challenging users for login credentials.

WNA uses a Kerberos session ticket that is generated when logging into the Windows desktop. The ticket is not visible to the user and contains, among other things, login credentials. The ticket is passed through the browser to the web application. The SPNEGO protocol is used to retrieve the Kerberos credentials from the browser. Web applications validate the credentials by checking them against the Key Distribution Center (KDC) server on the Windows domain server. If authentication succeeds, access is automatically granted to the Web applications.

Figure 21–1 below provides a conceptual view of the key components involved when using WNA. The authentication flow is described following the diagram.

Figure 21–1 Conceptual View and Flow of WNA



The authentication flow is as follows:

- (1) The user signs into their domain at the Windows desktop (Windows sign-on), which is configured for Kerberos authentication.
- (2, 3, 4) The user then uses a browser to accesses the Web application, which is configured as a Java Single Sign-On (JSSO) partner application. If the user is not authenticated, they are redirected to JSSO. The JSSO application is configured with WNA authentication and uses the Generic Security Services Java API (GSS-API) token exchange protocol to request the user's Kerberos ticket. OC4J validates the ticket and extracts the user's Security Account Manager (SAM) account information.
- (5, 6, 7) OC4J maps the SAM account name to the user in Active Directory by querying Active Directory to locate the user. Active Directory is then used to compute the user's enterprise groups. The user's identity and their enterprise groups are set as the authenticated subject to complete the authentication step. A cookie is then set for the user and the request is redirected back to the Web application.

Prerequisites for Configuring WNA

The following tasks must be completed before configuring WNA:

- Install OC4J server.
- Verify that the Active Directory server that is being used is accessible and that the server's KDC is operational.
- Create the necessary users and groups in Active Directory. For example `oc4jadmin` and the management groups such as `oc4j-adminstrators` and `oc4j-app-adminstrators`. Refer to "Creating Required Accounts in the LDAP Server" on page 11-16 for a list of all required accounts.

Step 1: Configure the Linux Host System as a Kerberos Client

The Linux host system where JSSO is deployed must be configured as a kerberos client.

To configure the Linux host system as a kerberos client:

1. On the Linux host server, open the `/etc/krb5.conf` file.
2. In the `[libdefaults]` section, edit the value of `default_realm` to a default kerberos realm name for your environment. The name can be a domain name and must be entered in upper case. For example:

```
[libdefaults]
 default_realm = MYDOMAIN.COM
```

3. In the `[realms]` section, replace the realm name with the name defined in step 2 and set its value to the fully qualified domain name of the Active Directory server including the kerberos port number. For example:

```
[realms]
 MYDOMAIN.COM = {
  kdc = name.mydomain.com:8787
 }
```

4. In the `[domain_realm]` section, set the DNS domain name for your JSSO server equal to the default realm. For example:

```
[domain_realm]
.mydomain.com = MYDOMAIN.COM
```

5. Save and close the `krb5.conf` file.
6. Verify that the clocks on both your JSSO server and the Active Directory server are synchronized. This includes the time, date, and time zone settings.

Step 2: Create a User Account in Active Directory

A Web application host user account must be created on the Active Directory server with the same host name where JSSO is deployed.

To create a Web application service host user account:

1. Log in to the Active Directory server as the Administrator.
2. Click **Start** -> **Programs** -> **Administrative Tools** -> **Active Directory Users and Computer**.
3. Select the Users folder.
4. Click **Create User**.
5. Enter the following information:
   - First name: Enter a nickname that can be any readable name for this account.
   - User logon name: Enter the JSSO host's complete DNS name. For example, `mydomain.com`.
6. Click **Next**.
7. Enter a password for this user. Do not select any password expiration settings.
8. Click **Next**. The new user appears in the list of users.

Step 3: Generate and Test the Keytab File for the Linux Host

A keytab file is used by the Web application to map an account name to a service principal name. The keytab file is required and must be generated using the `ktpass` command.

To generate the keytab file for the Linux host:

1. Open a command prompt on the Active Directory server.
2. Issue the following command:

   ```
   C:\> ktpass -princ HTTP/domain@DEFAULT_REALM -pass password
   -mapuser host_user_account -out file_name.keytab
   ```

   Where:

   - The `-princ` (principal) value is `HTTP/` followed by the fully qualified domain name of the Web application service host, followed by the default realm name. This is case sensitive. The Active Directory default realm must be in upper case and the fully qualified domain name of the Web application should be in lower case. For example:

     ```
     -princ HTTP/name.mydomain.com@MYDOMAIN.COM
     ```

   - The `-pass` value must be set to the same password assigned to the Web application host user account created on the Active Directory server.
   - The `-mapuser` value is the JSSO fully qualified domain name (hostname) user created on the Active Directory server.
   - The `-out` value is the name for the output file. For example:

     ```
     MyFile.keytab
     ```

3. Copy the generated keytab file to the OC4J instance where JSSO is deployed and configured on the Linux Host. For example:

   ```
   ORACLE_HOME/j2ee/home/config
   ```

4. Issue the following `kinit` command to test the Kerberos connection between the Linux server and the Active Directory server to ensure that they have both been properly configured. The `kinit` executable for Red Hat Linux is located in the `/usr/kerberos/bin` directory.

   ```
   # /usr/kerberos/bin/kinit -k -t ORACLE_HOME/j2ee/home/config/keytab_file
   HTTP/domain_name
   ```

   The command must include the full path to the keytab file generated in step two and the fully qualified domain name of the of the Web application service host. For example:

   ```
   # /usr/kerberos/bin/kinit -k -t oracle/j2ee/home/config/MyFile.keytab
   HTTP/name.mydomain.com
   ```

   If the command is successful, there should be no output from the command and another command prompt returns. Any errors that are reported in the output must be resolved.

Step 4: Configure the OC4J Instance

The tasks in this section configure an OC4J instance and its deployed Web applications to use WNA. Stop the OC4J instance before completing the tasks in this section. Make sure OC4J is started after completing the tasks in this section.

The following tasks are required to configure an OC4J instance to use WNA:

- Set Security System Properties for OC4J
- Edit the System JAZN Configuration File
- Edit the JAZN Configuration File
- Edit Application Deployment Descriptors

Set Security System Properties for OC4J

The following two system properties must be set to enable WNA for an OC4J instance. The properties are typically set within the `oc4j.cmd`, or `oc4j.sh`, script that is used to start OC4J. The properties can also be set on the command line when directly starting OC4J using `oc4j.jar`.

```
-Djavax.security.auth.useSubjectCredsOnly=false
-Doracle.security.jazn.config =ORACLE_HOME/j2ee/home/config/jazn.xml
```

System Properties for Debugging

Three additional properties can optionally be set to enable debug information for Kerberos authentication. The properties are:

```
-Djava.security.krb5.realm=kerberos-realm-name
-Djava.security.krb5.kdc=kdc_host_name
-Dsun.security.krb5.debug=true
```

Edit the System JAZN Configuration File

Login modules for both Kerberos WNA and Active Directory must be added to the `system-jazn-data.xml` file. The login modules are added within the `<jazn-loginconfig>` element, which contains the login modules that are registered with OC4J.

To add Kerberos WNA and Active Directory login modules:

1. Open *ORACLE_HOME*`/j2ee/home/config/system-jazn-data.xml`.
2. Add the following XML block, which defines a kerberos login module configuration for the `com.sun.security.jgss.accept` application. The `keytab` and `principal` values must be the same as the values supplied in "Step 3: Generate and Test the Keytab File for the Linux Host" on page 4. The `keyTab` value must include the complete path to keytab file and the principal value must be the fully qualified domain name (preceded by `HTTP/`) of the machine that is running the JSSO server.

```
...
<jazn-loginconfig>
   <application>
      <name>com.sun.security.jgss.accept</name>
      <login-modules>
         <login-module>
            <class>com.sun.security.auth.module.Krb5LoginModule</class>
```

```
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>debug</name>
                  <value>true</value>
               </option>
               <option>
                  <name>addAllRoles</name>
                  <value>true</value>
               </option>
               <option>
                 <name>useKeyTab</name>
                 <value>true</value>
               </option>
               <option>
                  <name>keyTab</name>
                  <value>C:/j2ee/home/config/MyFile.keytab</value>
               </option>
               <option>
                  <name>principal</name>
                  <value>HTTP/name.mydomain.com</value>
               </option>
               <option>
                  <name>doNotPrompt</name>
                  <value>true</value>
               </option>
               <option>
                  <name>storeKey</name>
                  <value>true</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
   ...
```

3. Add the following XML block, which configures the Active Directory login module configuration for the JSSO application. If you have already set the instance wide login module to be Active Directory, then you can skip this step.

```
<application>
   <name>javasso</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.connect.pool.prefsize</name>
               <value>10</value>
            </option>
            <option>
               <name>oracle.security.jaas.ldap.provider.connect.pool</name>
               <value>true</value>
            </option>
            <option>
               <name>oracle.security.jaas.ldap.provider.credential</name>
               <value>!welcome1</value>
            </option>
            <option>
```

```
      <name>oracle.security.jaas.ldap.provider.type</name>
      <value>Active Directory</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.connect.pool.maxsize</name>
      <value>25</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.role.searchscope</name>
      <value>subtree</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.user.searchscope</name>
      <value>subtree</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.membership.searchscope</name>
      <value>direct</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.lm.cache_enabled</name>
      <value>true</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.member.attribute</name>
      <value>member</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.connect.pool.initsize</name>
      <value>2</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.connect.pool.timeout</name>
      <value>300000</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.user.object.class</name>
      <value>inetOrgPerson</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.provider.url</name>
      <value>ldap://name.mydomain.com:389</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.role.searchbase</name>
      <value>cn=builtin,dc=dlin,dc=net</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.user.searchbase</name>
      <value>cn=users,dc=dlin,dc=net</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.role.object.class</name>
      <value>group</value>
   </option>
   <option>
      <name>oracle.security.jaas.ldap.role.name.attribute</name>
      <value>cn</value>
   </option>
   <option>
```

```
                <name>oracle.security.jaas.ldap.user.name.attribute</name>
                <value>sAMAccountName</value>
             </option>
             <option>
                <name>oracle.security.jaas.ldap.provider.user</name>
                <value>cn=administrator,cn=users,dc=dlin,dc=net</value>
             </option>
          </options>
       </login-module>
    </login-modules>
</application>
```

4. Add the following XML block, which defines a kerberos login module configuration for the JSSO application. The following values must be changed to match your Active Directory environment:

   - `oracle.security.jaas.ldap.provider.user` – the Active Directory server administrator
   - `oracle.security.jaas.ldap.provider.credential` – the Active Directory server administrator credential
   - `oracle.security.jaas.ldap.user.searchbase` – the Active Directory User Search Base
   - `oracle.security.jaas.ldap.role.searchbase` – the Active Directory Role Search Base
   - `oracle.security.jaas.ldap.provider.url` – the Active Directory server URL

```
...
<jazn-loginconfig>
   <application>
      <name>javasso</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.connect.pool.prefsize</name>
                  <value>10</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.connect.pool</name>
                  <value>true</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.user</name>
                  <value>cn=administrator,cn=users,dc=dlin,dc=net</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.credential</name>
                  <value>!welcome1</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.provider.type</name>
                  <value>Active Directory</value>
               </option>
               <option>
                  <name>oracle.security.jaas.ldap.connect.pool.maxsize</name>
```

```
            <value>25</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.role.searchscope</name>
            <value>subtree</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.user.searchscope</name>
            <value>subtree</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.membership.searchscope</name>
            <value>direct</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.lm.cache_enabled</name>
            <value>true</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.member.attribute</name>
            <value>member</value>
         </option>
         <option>
         <name>oracle.security.jaas.ldap.connect.pool.initsize</name>
            <value>2</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.connect.pool.timeout</name>
            <value>300000</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.user.object.class</name>
            <value>inetOrgPerson</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.provider.url</name>
            <value>ldap://name.mydomain.com:389</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.role.searchbase</name>
            <value>cn=builtin,dc=dlin,dc=net</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.user.searchbase</name>
            <value>cn=users,dc=dlin,dc=net</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.role.object.class</name>
            <value>group</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.role.name.attribute</name>
            <value>cn</value>
         </option>
         <option>
            <name>oracle.security.jaas.ldap.user.name.attribute</name>
            <value>sAMAccountName</value>
         </option>
      </options>
   </login-module>
```

```
      </login-modules>
   </application>
   ...
```

5. Save and close the `system-jazn-data.xml` file.

Edit the JAZN Configuration File

The JAZN configuration file must be edited to register the WNA authentication type. The file contains OC4J's JSSO configuration.

To register the WNA authentication type:

1. Open the *ORACLE_HOME*`/j2ee/home/config/jazn.xml` file.
2. Add the following property within the `<jazn>` element:

```
<jazn>
   ...
   <property name="custom.sso.token.asserter.authtypes"
             value="WINDOWS-KERBEROS-AUTH"/>
   ...
<jazn>
```

3. Save and close the file.

Edit Application Deployment Descriptors

The Oracle application deployment descriptor for the JSSO application as well as any Web applications that uses JSSO must be edited in order to use WNA authentication.

To edit application deployment descriptors:

1. Open the *ORACLE_HOME*`/j2ee/home/application-deployments/javasso/orion-application.xml` file.
2. Comment-out the current `<jazn>` configuration.
3. Add the following `<jazn>` configuration and change the `kerberos-servicename` value to use to use the fully qualified domain name for your JSSO host (preceded by `HTTP@`):

```
<jazn provider="XML">
   <jazn-web-app auth-method="WINDOWS_KERBEROS_AUTH"/>
   <property name="kerberos-servicename" value="HTTP@name.mydomain.com"/>
</jazn>
```

4. Save and close the JSSO application's `orion-application.xml` file.
5. For each Web application that uses the JSSO application, open its `orion-application.xml` file and modify the XML provider's authentication method attribute to use `CUSTOM_AUTH` as follows:

```
<jazn provider="XML">
   ...
   <jazn-web-app auth-method="CUSTOM_AUTH"/>
   ...
</jazn>
```

   The Oracle application deployment descriptor for an application is located in *ORACLE_HOME*`/j2ee/home/application-deployments/`*application_name*`/orion-application.xml`.

Step 5: Configuring a Browser and Testing WNA

The host name where the JSSO application is deployed must be added as a trusted host within a client's Internet Explorer browser before attempting to access a Web application that has been configured to use WNA. If Internet Explorer is running on a computer that is in the JSSO domain, a Kerberos negotiation takes place. If Internet Explorer is running on a computer that is not part of the JSSO domain, then the fallback logic is to use basic authentication and the browser will prompt for a username and password. The username and password is authenticated against the Active Directory configured in the `system-jazn-data.xml` file.

To configure your browser settings and test the WNA feature, perform the following steps:

1. Log in to the Windows domain.
2. Open the Internet Explorer browser.
3. From the browser's Tools menu, select Internet Options. The Internet Options dialog box displays.
4. From the Internet Options dialog box, select the Security tab.
5. Click to highlight the Local intranet icon and click **Sites**. The Local intranet dialog box displays.
6. From the Local intranet dialog box, click **Advanced**.
7. In the text field, enter the URL for the computer that is hosting the JSSO server. For example, `http://dude.mydomain.com`. Do not include the JSSO login page or port number.
8. Click **Add** to add the URL to the list of Web sites for the Local intranet zone.
9. Click **OK** and then click **OK** again to close the Local intranet dialog box.
10. From the Security tab on the Internet Options dialog box, click **Custom Level**. The Security Settings dialog box displays. Make sure the Automatic logon only in Intranet zone option is selected.
11. Click **OK** to close the Security Settings dialog box.
12. Click **OK** to close the Internet Options dialog box.
13. In the browser address field, enter the following URL to navigate to the JSSO server login page: `http://`*`host:port`*`/jsso/SSOLogin`. Replace *`host:port`* with the host and port used to access the JSSO application.

    The Kerberos credentials are transparently passed through the browser to the JSSO server and you are automatically logged into the JSSO server. If you are redirected to the top-level page, then the test is successful.

A

Tips and Troubleshooting for OC4J Security

This appendix discusses best practices for the OC4J security, as well as troubleshooting issues to be aware of, and related tips:

- Best Practices for OC4J Security
- General OC4J Security Tips and Troubleshooting
- Logging

> **Note:** There are also troubleshooting notes and sections elsewhere in this book, particular to the topics of the chapters where they appear.

Best Practices for OC4J Security

This section describes best practices in the following areas:

- JAAS Best Practices
- HTTPS Best Practices

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 Oracle Single Sign-On and Oracle Internet Directory) out of the box. Refer to "Migrating Principals from the principals.xml File" on page 7-16.
- *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 10.1.3.x releases), you will have more time to focus on business logic instead of infrastructure code if you leverage the rich functionality provided by the OracleAS JAAS Provider, Oracle Single Sign-On, and Oracle Internet Directory. Both Oracle 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. (Alternatively, use one of the external LDAP providers that Oracle supports.)
- *Use Oracle Single Sign-On as the authentication mechanism with the OracleAS JAAS Provider.* Various authentication options are available; however, we strongly recommend leveraging the Oracle 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.

  (Alternatively, use Java SSO if your installation does not include Oracle Application Server infrastructure.)
- *Use the OracleAS JAAS Provider declarative features to reduce programming.* Because most of the features in the OracleAS JAAS Provider are controlled declaratively (through configuration), 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.
- *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.
- *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.

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 significant 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 significant 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.
- *Tune the Oracle HTTP ServerSSLSessionCacheTimeout 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.

  See Also:

  - *Oracle HTTP Server Administrator's Guide*

General OC4J Security Tips and Troubleshooting

Be aware of the following issues and how to handle them:

- File jazn.xml Not Found
- Authentication Issues
- 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 4-9

Authentication Issues

If you attempt to log in to a protected application and authentication fails, but you know the user and password are properly configured in the identity repository, confirm by some other means that the identity repository is up and available, and is in the location specified in the `<jazn>` element `location` attribute in `orion-application.xml` or `jazn.xml` (as applicable).

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
```

Then the OracleAS JAAS Provider is probably not specified as the JAAS policy provider.

See Also:

- "Specification of the Oracle Policy Provider" on page 5-16

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 6-6 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 the OracleAS JAAS Provider
- Using Standard JDK Logging with the OracleAS JAAS Provider Admintool

See Also:

- The following location for an overview of standard Java logging:

  `http://java.sun.com/j2se/1.4.2/docs/guide/util/logging/overview.html`

- Javadoc for the `java.util.logging` package:

  `http://java.sun.com/j2se/1.4.2/docs/api/java/util/logging/package-summary.html`

- *Oracle Containers for J2EE Developer's Guide* and *Oracle Containers for J2EE Configuration and Administration Guide* for information about logging features and logging configuration in OC4J

Using Oracle Diagnostic Logging with the OracleAS JAAS Provider

OC4J and the 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

The OracleAS JAAS Provider Admintool uses standard JDK logging. To run logging with the Admintool, change the logging level from `INFO` to `FINE`, `FINER`, or `FINEST`. (Most log messages from the Admintool are logged at level `FINE` or `FINER`.) 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 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 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_developers` and a security constraint for the servlet
- In `orion-application.xml`, a role mapping between `developers` and `sr_developers`

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, as described in "Configuring the File-Based Provider in Application Server Control" on page 7-2. 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_developers`. 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_developers</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_developers</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_developers` 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, as described in "Specifying a Security Provider" on page 6-9 and "Mapping Security Roles" on page 6-10.

```
<orion-application>
   ...
   <security-role-mapping name="sr_developers">
      <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_developers') = " +
                    request.isUserInRole("sr_developers") + "<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, as described in "Granting and Revoking Permissions" on page C-14.

```
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_developers') = " +
                    request.isUserInRole("ar_developers") + "<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 5-5

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_developers">
      <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, using JAAS policy to check whether the user has permission to read `foo.txt`. Due to the preceding configuration, `doasprivileged` mode is used.

For purposes of comparison, this example also shows equivalent code using `AccessController` to check permissions. Being inside a `doAsPrivileged()` block is equivalent to the `doasprivileged` configuration for the JAAS policy code.

```
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 {
       final 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_developers') = " +
                   request.isUserInRole("ar_developers") + "<br>");
       out.println
           ("request.getUserPrincipal = " + request.getUserPrincipal() + "<br>");

       //create Permission
       FilePermission perm = new FilePermission("/home/developer/foo.txt","read");

       // CHECK PERMISSION VIA JAAS POLICY
       //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));

       // CHECK USER'S PERMISSION VIA ACCESS CONTROLLER
       Subject.doAsPrivileged(s, new PrivilegedAction() {
            public Object run() {
                try {
                    AccessController.checkPermission(perm);
                    out.println("<br>");
                    out.println
                      ("AccessController checkPermission passed for permission: "
                       + perm);
                    out.println("<br>");
                } catch (IOException e) {
                    e.printStackTrace();
                }
                return null;
            }
```

```
        }, null);

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

C

OracleAS JAAS Provider Admintool Reference

This chapter contains reference information for the OracleAS JAAS Provider Admintool. It is divided into the following sections:

- Getting Started with the Admintool
- Summary of Admintool Command-Line Syntax and Options
- Admintool Shell
- Admintool Administrative Functions

Notes:

- The Admintool can be used with either `system-jazn-data.xml` or Oracle Internet Directory as the user repository and policy repository. User and role information in Oracle Internet Directory is read-only for the Admintool, however.
- Changes made by the Admintool when you are using the file-based provider do not take effect until OC4J is restarted.

See Also:

- "Overview of the OracleAS JAAS Provider Admintool" on page 4-3
- "Using Standard JDK Logging with the OracleAS JAAS Provider Admintool" on page A-6

Getting Started with the Admintool

This section will help you get started with the OracleAS JAAS Provider Admintool and covers the following topics:

- Running the Admintool
- User Repository Location for the Admintool
- Authentication for the Admintool
- Using Custom Principals and Permissions with the Admintool

Running the Admintool

Run the Admintool by executing the OracleAS JAAS Provider `jazn.jar` file using the `java -jar` option. Either make your current directory the directory where `jazn.jar` is located, or specify the path to `jazn.jar` on the Java command line, as in the following example:

```
% java -jar /myroot/mydir/jazn.jar ...
```

> **Note:** The location of `jazn.jar` is typically:
>
> *ORACLE_HOME*`/j2ee/home/jazn.jar`

By default, the Admintool looks for the `jazn.xml` configuration file in the `config` directory under your current directory. You can alter this by directly specifying a location through the `oracle.security.jazn.config` system property, or by specifying an Oracle home or J2EE home location through the `oracle.home` property or `oracle.j2ee.home` property, respectively. The precedence of search locations is described in "The jazn.xml File" on page 4-9.

The following example specifies the location of the `jazn.xml` file:

```
% java -jar -Doracle.security.jazn.config=/tmp/jazn.xml jazn.jar ...
```

The next example specifies the J2EE home location, where *ORACLE_HOME* is the path to the Oracle home directory, and *instancename* is the name of the OC4J instance (such as `home`, for example). Based on this, the Admintool will look for `jazn.xml` in the *ORACLE_HOME*`/j2ee/`*instancename*`/config` directory:

```
% java -jar -Doracle.j2ee.home=ORACLE_HOME/j2ee/instancename jazn.jar ...
```

User Repository Location for the Admintool

When you use the file-based security provider, the `jazn.xml` file specifies the location of the user repository (`system-jazn-data.xml` or `jazn-data.xml`) that the Admintool will use, through the `location` attribute of the `<jazn>` element. By default, this is the `system-jazn-data.xml` file, but you can update the `location` setting to use an application-specific `jazn-data.xml` file:

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

When you use Oracle Identity Management, the `<jazn> location` attribute specifies the Oracle Internet Directory location, such as in the example that follows. However, *do not* manually update the `location` value when you use Oracle Identity Management; this value is set automatically when you associate the Oracle Internet Directory instance with OC4J through Application Server Control (as described in "Associating Oracle Internet Directory with OC4J" on page 8-6). Here is an example:

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

Authentication for the Admintool

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, as in this example:

  ```
  % 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
  ```

  Important: 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-6.

Using Custom Principals and Permissions with the Admintool

For the Admintool to work with custom principals and permissions, configure the property `classpath` to specify the location of the JAR file containing the custom classes. Do this with a `<property>` subelement under the `<jazn>` element in the `jazn.xml` file, as in the following example:

```
<jazn ... >
   ...
   <property name="classpath" value="/tmp/customPrincipal.jar" />
   ...
</jazn>
```

Note: In previous versions, the workaround was to put the JAR file into the `jre/lib/ext` directory. This will still work, but is not recommended.

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-13

- Authentication options

  ```
  -user username -password password
  ```

  See Also:

  - "Authentication for the Admintool" on page C-2

- 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-10
  - "Listing Login Modules" on page C-15

- Migration option

  ```
  -convert filename realm
  ```

  See Also:

  - "Converting from the principals.xml File to JAAS" on page C-17

- 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 Only)" on page C-13
- "Setting Passwords (File-Based Provider Only)" on page C-13

- 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-14
- "Listing Permissions" on page C-15

- Realm manipulation options for file-based provider

```
-addrealm realm admin {adminpwd adminrole | adminrole
  userbase rolebase realmtype }
-addrole realm role
-adduser realm username password
-remrealm realm
-remrole realm role
-remuser realm user
```

Important: Do not use the OracleAS JAAS Provider Admintool to create realms for Oracle Internet Directory. Realms created with this tool are suitable for the file-based provider only, and would not include sufficient information for use with Oracle Internet Directory.

See Also:

- "Adding and Removing Realms (File-Based Provider Only)" on page C-11
- "Adding and Removing Roles (File-Based Provider Only)" on page C-12
- "Adding and Removing Users (File-Based Provider Only)" on page C-12

- Realm options for general use

```
-grantrole role realm {user | -role to_role}
-revokerole role realm {user|-role from_role}
-listrealms realm
-listroles [realm [user | -role role]]
-listusers [realm [-role role | -perm permission]]
```

See Also:

- "Granting and Revoking Roles" on page C-14
- "Listing Realms" on page C-16
- "Listing Roles" on page C-16
- "Listing Users" on page C-16

- 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 the `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 admintoolcommand
```

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
- Admintool Shell Directory Structure
- Summary of Admintool Special Shell Commands

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:

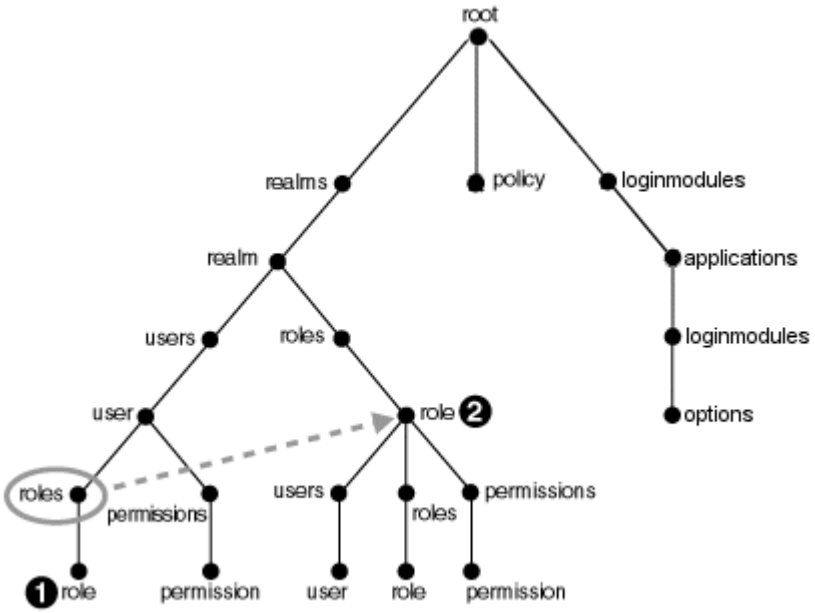
```
% java -jar jazn.jar -option1 -option2 ... -optionN
```

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

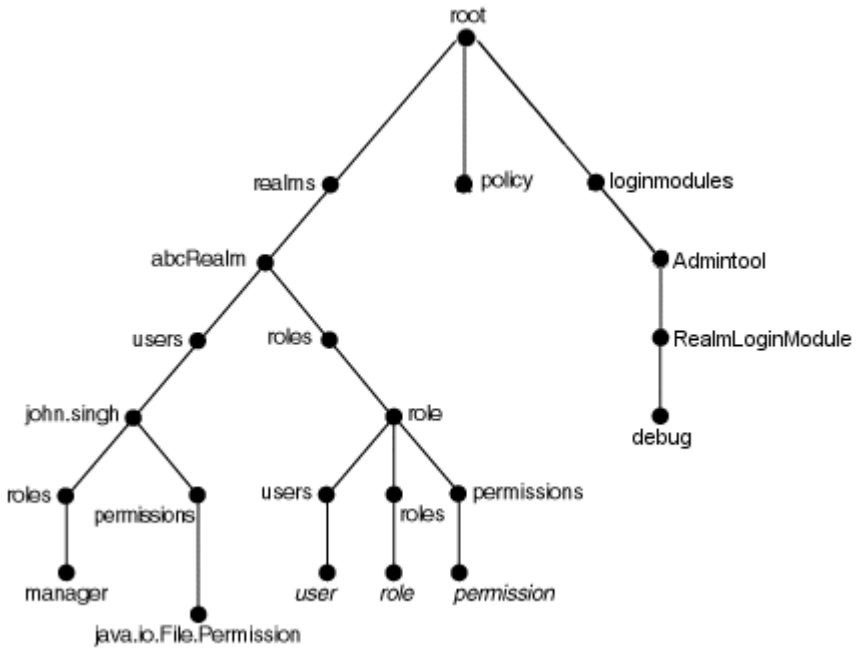


In this structure, the `user` and `role` nodes are linked together. This means that the `roles` link under `user` is the same link as the `roles` link under `realm`. In Unix terms, the `role` at numeral 1 in the diagram is a symbolic link to `role` at numeral 2 in the diagram.

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

Figure C–2 following 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 name [other_parameter]
mkdir name [other_parameter]
mk name [other_parameter]
```

The `add`, `mkdir`, and `mk` commands are equivalent; they each create a subdirectory or node under the current directory. The effect of the command depends on the current directory location. For example, if the current directory is the root, then the command creates a realm. If the current directory is `/realm/users`, then the command creates a user. Depending on the situation, these commands would be equivalent to Admintool commands such as `-addrealm` and `-adduser`.

These shell commands may require additional parameters—essentially, the same as what would be required for the equivalent Admintool commands. For example, consider the following shell command:

```
JAZN:> add myrealm myuser mypassword myrole
```

When executed under the directory `/realms`, this shell command is equivalent to the following:

```
% java -jar jazn.jar -addrealm myrealm myuser mypassword myrole
```

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.

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 *name*

The `rm` command removes the directory or node in the current directory. The effect of the command depends on 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. An error message is displayed if the specified name 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 (File-Based Provider Only)
- Adding and Removing Roles (File-Based Provider Only)
- Adding and Removing Users (File-Based Provider Only)
- Setting Passwords (File-Based Provider Only)
- Checking Passwords (File-Based Provider Only)
- 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

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 Table 9–5, " Login Module Control Flags" on page 9-17.

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, and where the login module supports a `debug` option:

```
% 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 (File-Based Provider Only)

`-addrealm` *`realm admin adminpwd adminrole`*
`-remrealm` *`realm`*

The `-addrealm` option creates a realm with the specified name and specified administrator.

For the file-based provider, specify the name of the realm, the realm administrator, the administrator password, and the administrator role:

`-addrealm` *`realm admin adminpwd adminrole`*

The `-remrealm` option deletes a realm.

Important: Do not use the OracleAS JAAS Provider Admintool to create realms for Oracle Internet Directory. Realms created with this tool are suitable for the file-based provider only, and would not include sufficient information for use with Oracle Internet Directory. Instead, use Oracle Delegated Administration Services (DAS).

See Also:

- "Overview of Delegated Administration Services" on page 4-4
- *Oracle Identity Management Guide to Delegated Administration*

For example, to create a realm `employees` with administrator `martha`, who has password `mypass` and is a member of role `hr`:

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

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 Only)

-**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 Only)

-**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
```

Setting Passwords (File-Based Provider Only)

-**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
```

Checking Passwords (File-Based Provider Only)

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

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

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

See Also:

- "Activation of the oc4jadmin Account (Standalone OC4J)" on page 4-12

Granting and Revoking Permissions

```
-grantperm {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 `java.security.Principal` interface, and *principal_params* is a string to be interpreted by the principal class.

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

Always specify a realm when you grant or revoke permissions for a user or role, but not when you grant or revoke permissions for a principal.

Permission specifications include the explicit class name of a permission, and its action and target parameters. Note that there may be multiple action and target parameters, as shown in the examples below.

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. See "Using Custom Principals and Permissions with the Admintool" on page C-3.

For example, to grant `RuntimePermission` to the principal `LDAPPrincipal`, with principal parameter `hobbes` (a value understood by `LDAPPrincipal`) and permission parameter `getProtectionDomain` (a value understood by `RuntimePermission`):

```
% 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

Always specify a realm when you list permissions for a user or role, but not when you list permissions for a principal.

> **Important:** PermissionClassManager and related classes and operations, including -listperms, are deprecated in OC4J 10.1.3.x implementations 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
- All roles granted to 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 7-16 for important additional information

D

OracleAS JAAS Provider Configuration Files

This chapter contains reference information for the `jazn.xml` and `system-jazn-data.xml` configuration files for the OracleAS JAAS Provider, covering the following topics:

- Hierarchy of jazn.xml
- Elements and Attributes of jazn.xml
- Hierarchy of system-jazn-data.xml
- Elements and Attributes of system-jazn-data.xml

> **Note:** Elements in `system-jazn-data.xml` relating to a user and role repository for the file-based provider also may appear in an application-specific `jazn-data.xml` file.

Hierarchy of jazn.xml

The `jazn.xml` file has a simple hierarchy, as follows:

```
<jazn>
   <property>
```

> **Note:** Do not use the `<jazn-web-app>` subelement of `<jazn>` in the `jazn.xml` file. The `<jazn-web-app>` element is intended for use in the `orion-application.xml` file.

See Also:

- "The jazn.xml File" on page 4-9 for an overview of this file

Elements and Attributes of jazn.xml

This section is an alphabetical dictionary of elements of the `jazn.xml` file.

> **Note:** Where attributes are discussed, note that attribute values are always set inside quotes: `attribute="value"`.

See Also:

- "The jazn.xml File" on page 4-9 for an overview of this file

<jazn>

Parent element: n/a (root)

Child elements: <property>

Required? Required; one only

This is the top-level element in the `jazn.xml` file, for configuration of the OracleAS JAAS Provider.

> **Note:** This element (optionally with any of its subelements) may also appear in an `orion-application.xml` file for application-level settings.

Table D–1 <jazn> Attributes

Name	Description
config	Values: n/a Default: n/a This attribute is unused in the OC4J 10.1.3.1 implementation.
default-realm	Values: String Default: n/a This specifies the realm that is used for an authentication or authorization request if no realm is explicitly specified. If multiple realms are defined in your repository, you must specify a default realm. **Note:** While there is no default if this attribute is not set, be aware that `jazn.xml` as shipped with OC4J sets the default realm to be `jazn.com`. (And if there is no default-realm setting in `orion-application.xml`, the default realm for the application defaults to that specified in `jazn.xml`.)
jaas-mode	Values: `null|doas|doasprivileged` Default: `null` This is used to specify JAAS mode, a fine-grained authorization feature provided by OC4J that is related to standard functionality of the `Subject` class static methods `doAs()` and `doAsPrivileged()`. With the setting `jaas-mode="doAs"`, application modules (Web modules and EJBs) are executed by OC4J within a `Subject.doAs()` block. With the setting `jaas-mode="doAsPrivileged"`, application modules are executed within a `Subject.doAsPrivileged()` block, using a null access control context. With the setting `jaas-mode="null"` (default), modules are executed under neither method. **See Also:** "Introduction to JAAS Mode" on page 5-5

Table D–1 (Cont.) <jazn> Attributes

Name	Description
location	Values: String Default: n/a For the file-based provider, this attribute in `jazn.xml` specifies the location of the instance-level user repository. In the `jazn.xml` file shipped with OC4J, this is specified to be `system-jazn-data.xml`. This setting can be an absolute path or a path relative to the location of the `jazn.xml` file. (This attribute in `orion-application.xml` can specify an application-specific user repository.) For Oracle Identity Management (the LDAP-based provider), this specifies the URL of the Oracle Internet Directory instance and is set automatically when you associate the Oracle Internet Directory instance with the OC4J instance through Application Server Control.
persistence	Values: `NONE` | `ALL` | `VM_EXIT` Default: `VM_EXIT` This specifies 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. With a setting of "`NONE`", changes are not written. With a setting of "`ALL`", changes are written after every modification. With a setting of "`VM_EXIT`" (default), changes are written when the JVM exits.
provider	Values: `XML` | `LDAP` Default: `XML` Specifies an instance-level security provider setting. At the OC4J instance level in `jazn.xml`, the `provider` attribute specifies the policy repository—"`XML`" for `system-jazn-data.xml` or "`LDAP`" for Oracle Internet Directory, as discussed in "Policy Repository Setting in jazn.xml" on page 5-15. **Note:** An application-level security provider is specified by the `provider` attribute in the `<jazn>` element in `orion-application.xml`. (By convention, `provider="XML"` in `orion-application.xml` is also used when the security provider is an external LDAP provider, a custom login module, or Oracle Access Manager.)
schema-major-version	Values: String Default: No default The major version number of the `jazn.xml` XSD. The value of this attribute is 10 for use with OC4J 10.1.3.x implementations. **Note**: This attribute is not defined directly in the XSD for `jazn.xml`. It is according to the `attributeGroup` specification in the top-level OC4J XSD.
schema-minor-version	Values: String Default: No default The minor version number of the `jazn.xml` XSD. The value of this attribute is 0 for use with OC4J 10.1.3.x implementations. **Note**: This attribute is not defined directly in the XSD for `jazn.xml`. It is according to the `attributeGroup` specification in the top-level OC4J XSD.

<property>

Parent element: <jazn>

Child elements: None

Required? Optional; zero or more

Specify a property setting as a name/value pair. Each security provider and usage mode supports its own set of properties. For example, there are properties specific to the LDAP-based provider (when using Oracle Identity Management), and properties specific to the identity management framework and Java SSO, as documented in earlier chapters. For example:

For LDAP:

```
<property name="ldap.protocol" value="no-ssl"/>
```

For the identity management framework:

```
<property name="idm.token.asserter.class"
          value="oracle.security.jazn.sso.SSOCookieTokenAsserter" />
```

For Java SSO (which is an implementation of the identity management framework):

```
<property name="idm.authentication.name" value="JavaSSO" />
<property name="custom.sso.url.login"
          value="http://host:port/jsso/SSOLogin" />
```

Table D–2 <property> Attributes

Name	Description
name	Values: String Default: n/a The name of the property.
value	Values: String Default: n/a The value of the property.

Hierarchy of system-jazn-data.xml

This section shows the element hierarchy of the `system-jazn-data.xml` file. The immediate subelements of `<jazn-data>` are `<jazn-policy>`, `<jazn-realm>`, `<jazn-loginconfig>`, `<jacc-repository>`, `<jazn-permission-classes>`, and `<jazn-principal-classes>`, but the latter three are not intended for customer use in this release.

> **Note:** Elements under `<jazn-realm>` can also be used in an application-specific `jazn-data.xml` file.

Hierarchy of system-jazn-data.xml

<jazn-data>

Hierarchy of system-jazn-data.xml

```
<jazn-policy>
   <grant>
      <grantee>
         <display-name>
         <principals>
            <principal>
               <realm-name>
               <type>
               <class>
               <name>
         <codesource>
            <url>
      <permissions>
         <permission>
            <class>
            <name>
            <actions>

<jazn-realm>
   <realm>
      <name>
      <users>
         <user>
            <name>
            <display-name>
            <description>
            <guid>
            <credentials>
      <roles>
         <role>
            <name>
            <display-name>
            <description>
            <guid>
            <members>
               <member>
                  <type>
                  <name>
      <jazn-policy>              DO NOT USE AS SUBELEMENT OF <realm>

<jazn-loginconfig>
   <application>
      <name>
      <login-modules>
         <login-module>
            <class>
            <control-flag>
            <options>
               <option>
                  <name>
                  <value>

<jacc-repository>          NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN

<jazn-permission-classes> NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN

<jazn-principal-classes>  NOT INTENDED FOR CUSTOMER USE; SUBHIERARCHY NOT SHOWN
```

See Also:

- "The system-jazn-data.xml File" on page 4-7 for an overview of this file
- "Application-Specific jazn-data.xml File (Optional)" on page 4-8

Elements and Attributes of system-jazn-data.xml

This section is an alphabetical dictionary of elements of the `system-jazn-data.xml` file.

Notes:

- Elements under `<jazn-realm>` can also be used in an application-specific `jazn-data.xml` file.
- Most settings in `system-jazn-data.xml` can be made through Application Server Control, as documented elsewhere in this document.
- Where attributes are discussed, note that attribute values are always set inside quotes: `attribute="value"`.

See Also:

- "The system-jazn-data.xml File" on page 4-7 for an overview of this file
- "Application-Specific jazn-data.xml File (Optional)" on page 4-8

<actions>

Parent element: <permission>

Child elements: None

Required? Optional; zero or one

As applicable, this element can specify the actions that are permitted with respect to the associated permission class and name. For example:

```
<permission>
   <class>oracle.security.jazn.realm.RealmPermission</class>
   <name>jazn.com</name>
   <actions>droprealm</actions>
</permission>
```

<application>

Parent element: <jazn-loginconfig>

Child elements: <name>, <login-modules>

Required? Optional; zero or more

In login module configuration, this element (through its subelements) specifies the name of an application and configures login modules to be used by that application.

See Also:

- <jazn-loginconfig> on page D-11 for an example

<class>

Parent element: <principal>, <permission>, or <login-module>

Child elements: None

Required? Required within parent element; one only

This element has several uses:

- Within the `<principal>` element (for granting permissions to a principal), it specifies the fully qualified name of the principal class—the class that is instantiated to represent a principal that is being granted a set of permissions. For example (for a principal of type "role"):

  ```
  <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
  ```

- Within the `<permission>` element (for granting permissions to a principal), it specifies the fully qualified name of the permission class. For example (for RMI permission, used in accessing EJBs):

  ```
  <class>com.evermind.server.rmi.RMIPermission</class>
  ```

- Within the `<login-module>` element, it specifies the fully qualified name of the login module class. For example:

  ```
  <class>
     oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
  </class>
  ```

<codesource>

Parent element: <grantee>

Child elements: <url>

Required? Optional; zero or one

For policy configuration, either a `<principals>` element or a `<codesource>` element is used within a `<grantee>` element to specify what the permissions in question are being granted to. A `<codesource>` element specifies a codesource URL, to grant permissions to that codesource.

<control-flag>

Parent element: <login-module>

Child elements: None

Required? Required within parent element; one only

This element specifies one of the following control settings for a login module:

```
<control-flag>required</control-flag>
```

```
<control-flag>requisite</control-flag>

<control-flag>sufficient</control-flag>

<control-flag>optional</control-flag>
```

These are used according to standard functionality of the `javax.security.auth.login.Configuration` class. The overall authentication succeeds only if all "required" and "requisite" login modules succeed, possibly unless a "sufficient" login module is configured and succeeds—in that case, only the required and requisite login modules prior to the sufficient login module in the login module list must succeed.

See Also:

- Table 9–5, " Login Module Control Flags" on page 9-17 for additional information about the control flag settings
- <jazn-loginconfig> on page D-11 for an example

<credentials>

Parent element: <user>

Child elements: None

Required? Optional; zero or one

This element contains the authentication password for the user.

By default, OC4J uses password obfuscation for passwords specified in `system-jazn-data.xml` (or optionally in an application-specific `jazn-data.xml` file).

To instead use a cleartext (human-readable) password, set the `clear` attribute to "`true`" or precede the password with "!" (in which case "!" is not considered part of the password). Using cleartext passwords is discouraged, however.

The following are equivalent:

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

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

Table D–3 <credentials> Attributes

Name	Description
clear	Values: `true|false` Default: `false` Set this to "`true`" to use a cleartext password instead of an obfuscated password.

Note: The `clear` attribute is not specified in the `system-jazn-data.xml` schema definition, but is supported by the OracleAS JAAS Provider runtime implementation.

See Also:

- <jazn-realm> on page D-14 for an example
- "Password Obfuscation in OC4J Configuration Files" on page 6-3

<description>

Parent element: <user> or <role>

Child elements: None

Required? Optional; zero or one

This contains a text string to describe the item (user or role, depending on the parent element). For example (for the user `oc4jadmin`):

```
<description>The OC4J user with administrative privileges</description>
```

<display-name>

Parent element: <grantee>, <user>, or <role>

Child elements: None

Required? Optional; zero or one

This contains a text string to specify a display name to be used for the item (grantee, user, or role, depending on the parent element), such as for display by a GUI tool. For example (for the user `oc4jadmin`):

```
<display-name>OC4J Administrator</display-name>
```

<grant>

Parent element: <jazn-policy>

Child elements: <grantee>, <permissions>

Required? Optional; zero or more

For policy configuration, this element contains a grant entry that assigns a set of permissions to a grantee (a codesource or set of principals).

Note: Do not use the `grantee-names` attribute (specified in the `system-jazn-data.xml` schema definition). Specify grantees through `<grantee>` subelements.

See Also:

- <jazn-policy> on page D-12 for an example

<grantee>

Parent element: <grant>

Child elements: <display-name>, <principals>, <codesource>

Required? Required within parent element; one only

When a policy grant is specified through a `<grant>` element, the `<grantee>` element (used in conjunction with a `<permissions>` element) specifies who or what the permissions are granted to—either a set of principals or a codesource.

See Also:

- <jazn-policy> on page D-12 for an example

<guid>

Parent element: <user> or <role>

Child elements: None

Required? Optional; zero or one

This element specifies the globally unique identifier (GUID) of the item (either a user or role, depending on the parent element). A GUID is sometimes generated and used internally by the OracleAS JAAS Provider, such as in migrating a user or role to a different security provider. It is not an item that you would set yourself.

<jacc-repository>

Parent element: <jazn-data>

Child elements: <jacc-policy>

Required? n/a

This element and its subhierarchy (shown here) are not intended for customer use in the OC4J 10.1.3.1 implementation.

```
<jacc-repository>
   <jacc-policy>
      <contextID>
      <excluded-policy>          SAME SUBHIERARCHY AS <jazn-policy>
      <unchecked-policy>         SAME SUBHIERARCHY AS <jazn-policy>
      <role-policy>              SAME SUBHIERARCHY AS <jazn-policy>
```

<jazn-data>

Parent element: n/a (root)

Child elements: <jazn-policy>, <jazn-realm>, <jazn-loginconfig> (considering only those intended for customer use)

Required? Required; one only

This is the top-level element in the `system-jazn-data.xml` file, for configuration of the OracleAS JAAS Provider.

Table D–4 <jazn-data> Attributes

Name	Description
schema-major-version	Values: String Default: No default The major version number of the `system-jazn-data.xml` XSD. The value of this attribute is 10 for use with OC4J 10.1.3.x implementations. **Note**: This attribute is not defined directly in the XSD for `system-jazn-data.xml`. It is according to the `attributeGroup` specification in the top-level OC4J XSD.
schema-minor-version	Values: String Default: No default The minor version number of the `system-jazn-data.xml` XSD. The value of this attribute is 0 for use with OC4J 10.1.3.x implementations. **Note**: This attribute is not defined directly in the XSD for `system-jazn-data.xml`. It is according to the `attributeGroup` specification in the top-level OC4J XSD.

<jazn-loginconfig>

Parent element: <jazn-data>

Child elements: <application>

Required? Optional; zero or one

This is the top-level element for configuring login modules associated with the specified applications (specified as part of the configuration). Here is an example for the Oracle-supplied `DBTableOraDataSourceLoginModule` (subelements shown here are described elsewhere in this appendix):

```
<jazn-loginconfig>
   <application>
      <name>application_name</name>
      <login-modules>
         <login-module>
            <class>
              oracle.security.jazn.login.module.db.DBTableOraDataSourceLoginModule
            </class>
            <control-flag>required</control-flag>
            <options>
               <option>
                  <name>data_source_name</name>
                  <value>jdbc/OracleDS</value>
               </option>
               <option>
                  <name>table</name>
                  <value>userinfo</value>
               </option>
               <option>
                  <name>roles_fk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>groupMembershipGroupFieldName</name>
```

```
                  <value>role</value>
               </option>
               <option>
                  <name>user_pk_column</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>passwordField</name>
                  <value>passWord</value>
               </option>
               <option>
                  <name>groupMembershipTableName</name>
                  <value>groupinfo</value>
               </option>
               <option>
                  <name>usernameField</name>
                  <value>userName</value>
               </option>
               <option>
                  <name>casing</name>
                  <value>sensitive</value>
               </option>
            </options>
         </login-module>
      </login-modules>
   </application>
   ...
</jazn-loginconfig>
```

<jazn-permission-classes>

Parent element: <jazn-data>

Child elements: <permission-class>

Required? n/a

This element and its subhierarchy (shown here) are not intended for customer use in the OC4J 10.1.3.1 implementation.

```
<jazn-permission-classes>
   <permission-class>
      <name>
      <description>
      <type>
      <class>
      <target-descriptors>
         <target-descriptor>
            <name>
            <description>
      <action-descriptors>
         <action-descriptor>
            <name>
            <description>
```

<jazn-policy>

Parent element: <jazn-data>

Child elements: <grant>

Required? Optional; zero or one

This is the top-level element for policy configuration, specifying policy grants that associate grantees (principals or codesources) with sets of permissions. Here is an example (subelements shown here are described elsewhere in this appendix):

```
<jazn-policy>
   <grant>
      <grantee>
         <principals>
            <principal>
               <realm-name>jazn.com</realm-name>
               <type>role</type>
               <class>oracle.security.jazn.spi.xml.XMLRealmRole</class>
               <name>jazn.com/oc4j-administrators</name>
            </principal>
         </principals>
      </grantee>
      <permissions>
         <permission>
            <class>com.evermind.server.AdministrationPermission</class>
            <name>administration</name>
            <actions>administration</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>modifyrealmmetadata</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>createrealm</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>dropuser</actions>
         </permission>
         <permission>
            <class>oracle.security.jazn.realm.RealmPermission</class>
            <name>jazn.com</name>
            <actions>droprealm</actions>
         </permission>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>login</name>
         </permission>
         <permission>
            <class>com.evermind.server.rmi.RMIPermission</class>
            <name>subject.propagation</name>
         </permission>
         <permission>
            <class>oracle.security.jazn.policy.RoleAdminPermission</class>
            <name>jazn.com/*</name>
         </permission>
      </permissions>
   </grant>
   ...
```

```
</jazn-policy>
```

Note: Do not use <jazn-policy> as a subelement of <realm>.

<jazn-principal-classes>

Parent element: <jazn-data>

Child elements: <principal-class>

Required? n/a

This element and its subhierarchy (shown here) are not intended for customer use in the OC4J 10.1.3.1 implementation.

```
<jazn-principal-classes>
   <principal-class>
      <name>
      <description>
      <type>
      <class>
      <name-description-map>
         <name-description-pair>
            <name>
            <description>
```

<jazn-realm>

Parent element: <jazn-data>

Child elements: <realm>

Required? Optional; zero or one

This is the top-level element for user and role information, specifying security realms and the users and roles they include. Here is an example (subelements shown here are described elsewhere in this appendix):

```
<jazn-realm>
   <realm>
      <name>jazn.com</name>
      <users>
         <user deactivated="true">
            <name>anonymous</name>
            <guid>D3D41721D3E311DABFFC25CB9F57C041</guid>
            <description>The default guest/anonymous user</description>
         </user>
         <user>
            <name>oc4jadmin</name>
            <display-name>OC4J Administrator</display-name>
            <guid>D3DB1C00D3E311DABFFC25CB9F57C041</guid>
            <description>OC4J Administrator</description>
            <credentials>{903}r7VKkMgJqP8fkDZCG7YMo7UZnT/B+HcK</credentials>
         </user>
         ...
      </users>
      <roles>
         <role>
```

```
            <name>ascontrol_admin</name>
            <display-name>ASControl Admin Role</display-name>
            <description>Administrative role for ASControl</description>
            <guid>D3DB1C05D3E311DABFFC25CB9F57C041</guid>
            <members>
               <member>
                  <type>user</type>
                  <name>oc4jadmin</name>
               </member>
            </members>
         </role>
         <role>
            <name>oc4j-administrators</name>
            <display-name>OC4J Admin Role</display-name>
            <description>Administrative role for OC4J</description>
            <guid>D3DB1C02D3E311DABFFC25CB9F57C041</guid>
            <members>
               <member>
                  <type>user</type>
                  <name>oc4jadmin</name>
               </member>
               ...
            </members>
         </role>
         ...
      </roles>
   </realm>
</jazn-realm>
```

<login-module>

Parent element: <login-modules>

Child elements: <class>, <control-flag>, <options>

Required? Required within parent element; one or more

This element specifies and configures a login module for a given application, with subelements specifying the class name, control flag, and option settings for the login module.

See Also:

- <jazn-loginconfig> on page D-11 for an example

<login-modules>

Parent element: <application>

Child elements: <login-module>

Required? Required within parent element; one only

This element, through one or more `<login-module>` subelements, configures the set of login modules for a given application.

See Also:

- <jazn-loginconfig> on page D-11 for an example

<member>

Parent element: <members>

Child elements: <type>, <name>

Required? Optional; zero or more

The element specifies the name of a member of the applicable role, and whether the member is a user or another role (according to the `<type>` subelement).

See Also:

- <jazn-realm> on page D-14 for an example

<members>

Parent element: <role>

Child elements: <member>

Required? Required within parent element; one only

This element specifies the members of a role. Members can be either users or other roles.

See Also:

- <jazn-realm> on page D-14 for an example

<name>

Parent element: <principal>, <realm>, <role>, <user>, <member>, <application>, or <option>

Child elements: None

Required? Required within parent element; one only

This element has several uses:

- Within the `<realm>` element, it specifies the name of a realm. For example:

  ```
  <name>jazn.com</name>
  ```

- Within the `<user>` element, it specifies the unique name of a user in the applicable realm. For example:

  ```
  <name>oc4jadmin</name>
  ```

- Within the `<role>` element, it specifies the unique name of a role in the applicable realm. For example:

  ```
  <name>oc4j-administrators</name>
  ```

- Within the `<member>` subelement of `<role>`, it specifies the name of a member of the role. For example (if the user `oc4jadmin` is to be a member of the role):

  ```
  <name>oc4jadmin</name>
  ```

- Within the <principal> element (for granting permissions to a principal), it specifies the unique name of a principal within the given realm. For example:

  ```
  <name>jazn.com/oc4j-administrators</name>
  ```

- Within the <application> element, it specifies the fully qualified name of the application whose login modules are being configured. For example:

  ```
  <name>oracle.security.jazn.tools.Admintool</name>
  ```

- Within the <option> element, it specifies the name of an option for configuration of a login module. (There is an accompanying <value> element for the option value.) For example, an option for DBTableOraDataSourceLoginModule:

  ```
  <option>
     <name>data_source_name</name>
     <value>jdbc/OracleDS</value>
  </option>
  ```

(This element as a subelement of <permission> is documented separately, immediately below.)

<name>

Parent element: <permission>

Child elements: None

Required? Optional; zero or one

As applicable, this element can specify the name of a permission that is meaningful to the permission class. For example:

```
<permission>
   <class>com.evermind.server.rmi.RMIPermission</class>
   <name>login</name>
</permission>
```

(This element as a subelement of <principal>, <realm>, <role>, <user>, <member>, <application>, or <option> is documented separately, immediately above.)

<option>

Parent element: <options>

Child elements: <name>, <value>

Required? Required within parent element; one or more

Each <option> element, through a <name> subelement and a <value> subelement, specifies the name and value for an option setting for a login module.

See Also:

- <jazn-loginconfig> on page D-11 for an example

<options>

Parent element: <login-module>

Child elements: <option>

Required? Optional; zero or one

This element, through its `<option>` subelements, specifies option settings for a login module.

See Also:

- <jazn-loginconfig> on page D-11 for an example

<permission>

Parent element: <permissions>

Child elements: <class>, <name>, <actions>

Required? Required within parent element; one or more

When a `<permissions>` element is used in policy grant configuration, each `<permission>` subelement specifies one permission being granted to the principal in question.

See Also:

- <jazn-policy> on page D-12 for an example

<permissions>

Parent element: <grant>

Child elements: <permission>

Required? Required within parent element; one only

When a policy grant is specified through a `<grant>` element, the `<permissions>` element (used in conjunction with a `<grantee>` element) specifies the permissions being granted, through a set of `<permission>` subelements.

Note: The `system-jazn-data.xml` schema definition does not specify this as a required element, but the OracleAS JAAS Provider runtime implementation requires its use within any `<grant>` element.

See Also:

- <jazn-policy> on page D-12 for an example

<principal>

Parent element: <principals>

Child elements: <realm-name>, <class>, <type>, <name>

Required? Optional; zero or more

When a `<principals>` element is used in policy grant configuration, each `<principal>` subelement specifies one principal being granted the permissions in question.

See Also:

- <jazn-policy> on page D-12 for an example

<principals>

Parent element: <grantee>

Child elements: <principal>

Required? Optional; zero or one

For policy configuration, either a `<principals>` element or a `<codesource>` element is used within a `<grantee>` element to specify what the permissions in question are being granted to. A `<principals>` element specifies a set of principals being granted the permissions.

For a subject to be granted these permissions, the subject should include all the specified principals.

See Also:

- <jazn-policy> on page D-12 for an example

<realm>

Parent element: <jazn-realm>

Child elements: <name>, <users>, <roles>

Required? Optional; zero or more

This element specifies a realm and the users and roles that belong to the realm.

Note: Do not use `<jazn-policy>` as a subelement of `<realm>`.

See Also:

- <jazn-realm> on page D-14 for an example

<realm-name>

Parent element: <principal>

Child elements: None

Required? Optional; zero or one

For granting permissions to a principal, this element specifies the name of the realm to which the principal belongs. (Its value would correspond to the value of a `<name>` subelement of a `<realm>` element where the realm is configured.) For example:

```
<realm-name>jazn.com</realm-name>
```

If a realm name is not specified, the default realm is assumed.

<role>

Parent element: <roles>

Child elements: <name>, <display-name>, <description>, <guid>, <members>

Required? Optional; zero or more

This element specifies a role and the members of that role.

See Also:

- <jazn-realm> on page D-14 for an example

<roles>

Parent element: <realm>

Child elements: <role>

Required? Optional; zero or one

This element specifies the set of roles that belong to a realm

See Also:

- <jazn-realm> on page D-14 for an example

<type>

Parent element: <member>

Child elements: None

Required? Required, one only

As a subelement of `<member>`, in specifying the member of a role, this element specifies the type of member—that is, whether the member is a user or another role. For example:

```
<type>user</type>
```

(This element as a subelement of `<principal>` is documented separately, immediately below.)

See Also:

- <jazn-realm> on page D-14 for an example

<type>

Parent element: <principal>

Child elements: None

Required? Optional; zero or one

As a subelement of `<principal>`, in granting permissions to a principal, this element can optionally specify the type of principal—that is, whether the principal is a user or a role. For example:

```
<type>role</type>
```

(This element as a subelement of `<member>` is documented separately, immediately above.)

See Also:

- <jazn-policy> on page D-12 for an example

<url>

Parent element: <codesource>

Child elements: None

Required? Required within parent element; one only

When a `<codesource>` element is used in policy grant configuration, the `<url>` subelement specifies the URL of the codesource being granted the permissions in question. For example:

```
"file:${oracle.home}/j2ee/home/jazn.jar"
```

(This follows the same format as shown for a `java2.policy` file in "Creating or Updating a Java 2 Policy File" on page 5-3.)

<user>

Parent element: <users>

Child elements: <name>, <display-name>, <description>, <guid>, <credentials>

Required? Optional; zero or more

This element specifies a user within a realm.

Table D–5 <user> Attributes

Name	Description
deactivated	Values: true | false Default: false You can set this attribute to "`true`" if you want to maintain a user in the configuration file but not have it be a currently valid user. This is the initial configuration of the `anonymous` user in the `jazn.com` realm, for example.

See Also:

- <jazn-realm> on page D-14 for an example

<users>

Parent element: <realm>

Child elements: <user>

Required? Optional; zero or one

This element specifies the set of users who belong to a realm.

See Also:

- <jazn-realm> on page D-14 for an example

<value>

Parent element: <option>

Child elements: None

Required? Required within the parent element; one only

The element specifies the value of an option for configuration of a login module. (There is an accompanying `<name>` element for the option name.) For example, an option for `DBTableOraDataSourceLoginModule`:

```
<option>
   <name>data_source_name</name>
   <value>jdbc/OracleDS</value>
</option>
```

E

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

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

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X