Oracle® Fusion Middleware

Securing Applications with Oracle Platform Security Services

12*c* (12.1.3)

E47835-04

June 2015

Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services, 12*c* (12.1.3)

E47835-04

Primary Author: Carlos Subi

Contributing Author: Vinaye Misra, Devanshi Mohan

Contributor: Amit Agarwal, Soumya Aithal, Moushmi Banerjee, Pratik Datta, Jordan Douglas, Guru Dutt, Todd Elwood, Vineet Garg, Vikas Ghorpade, Sandeep Guggilam, Shiang-Jia Huang, Dan Hynes, Supriya Kalyanasundaram, Lakshmi Kethana, Rohit Koul, Nithya Muralidharan, Frank Nimphius, Sudip Regmi, Bhupindra Singh, Kk Sriramadhesikan, Mamta Suri, Kavita Tippana, Ramana Turlapati, Jane Xu, Sam Zhou.

Contents

List of Examples

List of Figures

List of Tables

Preface

This manual explains the features and administration of the Oracle Platform Security Services.

Audience

The intended audience of this guide are experienced Java developers, administrators, deployers, and application managers who want to understand and use Oracle Platform Security Services.

The overall structure of the guide is divided into parts, each of which groups related major topics. Parts I through III are relevant to administrators; parts IV contains information about the OPSS policy model and is intended for developers; and part V contains reference information.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc`.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info` or visit `http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs` if you are hearing impaired.

Related Documentation

Additional information is found in the following documents:

- *Administering Oracle Fusion Middleware*
- *Oracle Fusion Middleware 2 Day Administration Guide*
- *Securing Web Services and Managing Policies with OWSM*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Directory Integration Platform*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Identity Federation*

- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*
- *Understanding the WebLogic Scripting Tool*
- *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*
- *Creating Schemas with the Repository Creation Utility*
- For links to API documentation, see Section G.1, "OPSS API References."

For a comprehensive list of Oracle documentation or to search for a particular topic within Oracle documentation libraries, see `http://www.oracle.com/technetwork/indexes/documentation/index.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 in This Guide

This chapter describes the most important changes introduced in this and previous 12c releases.

Updates in the June 2015 Document Refresh for 12.1.3.0.0

The updated items in this document refresh includes the following:

- The Interoperability table has been introduced in Chapter 3.1, "Compatibility Matrix for 11g and 12c Versions."

Updates in the January 2015 Document Refresh for 12.1.3.0.0

The updated items in this document refresh includes the following:

- Book version number has been corrected.

Updates in the October 2014 Document Refresh for 12.1.3.0.0

The updated items in this document refresh includes the following:

- All topic titles brought to 12c standards.
- The Java API for Identity Directory Service. See Section G.1, "OPSS API References."

New Features in Release 12.1.3.0.0

The new features and major changes introduced in release 12.1.3.0.0 follow.

- OPSS and audit schemas support EBR. See EBR references at the end of Section 7.5.2, "Upgrading a DB-Based Security Store."
- Datasources are created automatically in domains created with the JRF Template. See Note 2 in Section 7.3.1, "Using a New Database Instance."
- Ability to upgrade component audit definitions to the dynamic metadata model. See Section 7.7.
- Naming conventions and other guidelines for auditing. See Section 14.7.
- New audit events. See Table C–2.
- The OPSS diagnostic framework. See Section J.2, "The OPSS Diagnostic Framework."

- New arguments (migrate and skip) to `reassociateSecurityStore`. See Section 10.4.1, "reassociateSecurityStore."
- New argument (skip) to `migrateSecurityStore`. See Section 9.6.2, "Migrating with the Script migrateSecurityStore."
- Enhancements to the class JpsStartup. See Section 25.1.1, "The Class JpsStartup."
- Ability to export or import keystores to and from wallets. See Section 12.1.2.
- Support for FIPS-140. See Appendix 4.6.

New Features in Release 12.1.2.0.0

The features and documentation changes introduced in release 12.1.2.0.0 follow.

Features

The features introduced in this release include the following:

- The use of templates to seed security artifacts at domain creation or extension, and a new procedure to upgrade to 12.1.2. For details, see Chapter 7, "Lifecycle of Security Artifacts."

Documentation

The documentation changes introduced in this release include the following:

- Reference information about WLST commands has been removed from the guide. This content now resides in the *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.
- The different options for audit reporting are explained. See Section 15.1.
- Reference details are available for audit system attributes and events. See Table C–1 and Table C–5.

Part I

Understanding Security Concepts

This part contains the following chapters:

1 Introduction to Oracle Platform Security Services

Oracle Platform Security Services (OPSS) is a security platform that secures applications deployed on any of the supported platforms or in standalone applications.

This chapter introduces the main features of this platform in the following sections:

- What is Oracle Platform Security Services?
- OPSS Architecture Overview
- Oracle ADF Security Overview
- OPSS for Administrators
- OPSS for Developers

The scope of this document does *not* include Oracle Web Services security. For details about that topic, see *Securing Web Services and Managing Policies with OWSM*.

1.1 What is Oracle Platform Security Services?

OPSS provides enterprise product development teams, systems integrators, and independent software vendors with a standards-based, portable, integrated, enterprise-grade security framework for Java SE and Java EE applications.

OPSS is the underlying security platform that provides security to Oracle Fusion Middleware including WebLogic Server, Server Oriented Architecture (SOA) applications, Oracle WebCenter, Oracle Application Development Framework (ADF) applications, and Oracle Entitlement Server.

OPSS provides an abstraction layer in the form of application programming interfaces (APIs) that insulate developers from security and identity management implementation details. With OPSS, developers do not need to know the details of, for example, cryptographic key management, repository interfaces, or other identity management infrastructures. Using OPSS, in-house developed applications, third-party applications, and integrated applications benefit from the same, uniform security, identity management, and audit services across the enterprise.

For OPSS-related news, including FAQs, a whitepaper, and code examples, and forum discussions, see `http://www.oracle.com/technology/products/id_mgmt/opss/index.html`.

1.1.1 OPSS Main Features

OPSS complies with the following standards: role-based-access-control (RBAC); Java Enterprise Edition (Java EE); and Java Authorization and Authentication Services (JAAS).

Built upon these standards, OPSS provides an integrated security platform that supports:

- Authentication
- Identity assertion
- Authorization, based on fine-grained JAAS permissions
- The specification and management of application policies
- Secure storage and access of system credentials through the Credential Store Framework
- Secure storage and access of keys and certificates through the Keystore Service
- Auditing
- Role administration and role mappings
- The User and Role API
- Identity Virtualization
- Security configuration and management
- SAML and XACML
- Oracle Security Developer Tools, including cryptography tools
- Policy Management API
- Java Authorization Contract for Containers (JACC)

Details about a given OPSS feature functionality are found in subsequent chapters of this guide.

For details about the WebLogic Auditing Provider, see section Configuring the WebLogic Auditing Provider in *Administering Security for Oracle WebLogic Server.*

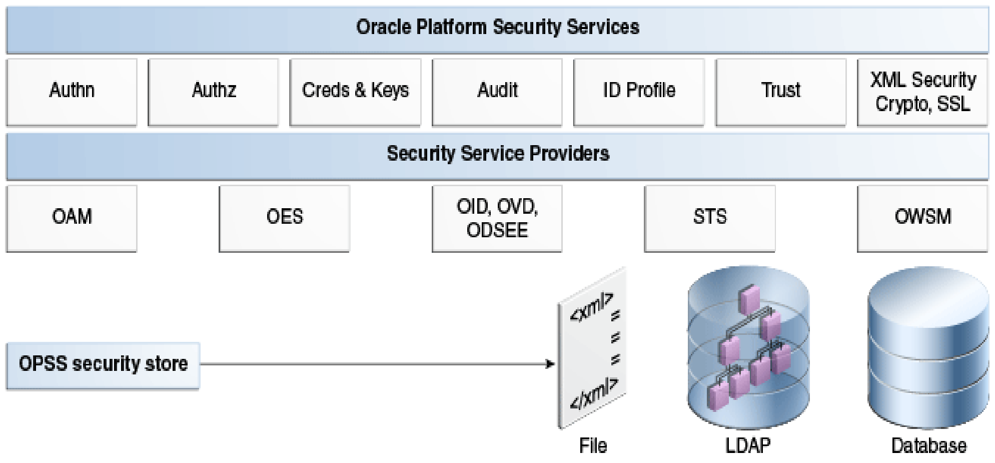
1.1.2 Supported Server Platforms

OPSS is supported in the Oracle WebLogic Server application platform.

1.2 OPSS Architecture Overview

OPSS comprises the application server's security and Oracle's Fusion Middleware security. Figure 1–1 illustrates the layered architecture that combines these two security frameworks:

Figure 1–1 The OPSS Architecture



The top layer includes the OPSS security services, the next layer includes the service providers, and the bottom layer includes the OPSS security store with a repository of one of three kinds.

Security Services Providers

Security Services Provider Interface (SSPI) provides Java EE container security and resource-based authorization for the environment.

SSPI is a set of APIs for implementing pluggable security providers. A module implementing any of these interfaces can be plugged into SSPI to provide a particular type of security service, such as custom authentication or a particular role mapping.

For details, see section The Security Service Provider Interfaces (SSPIs) in *Understanding Security for Oracle WebLogic Server*.

Oracle Platform Security Services

Oracle Platform Security Services (OPSS) includes the security store and several security services.

1.2.1 Benefits of Using OPSS

The benefits that OPSS offers include the following:

- Allows developers to focus on application and domain problems
- Supports enterprise deployments
- Supports several LDAP servers and SSO systems
- Is certified on the Oracle WebLogic Server
- Pre-integrates with Oracle products and technologies
- Offers a consistent security experience for developers and administrators
- Provides a uniform set of APIs for all types of applications
- Optimizes development time by offering abstraction layers (declarative APIs)
- Provides a simplified application security maintenance
- Allows changing security rules without affecting application code
- Eases the administrator’s job

- Integrates with identity management systems

OPSS supports security for Java EE applications and for Oracle Fusion Middleware applications, such as Oracle WebCenter and Oracle SOA Suite.

Developers can use OPSS APIs to secure all types of applications and integrate them with other security artifacts, such as LDAP servers, RDBMS, and custom security components.

Administrators can use OPSS to deploy large enterprise applications with a small, uniform set of tools and administer all security in them. OPSS simplifies the maintenance of application security because it allows the modification of security configuration without changing the application code.

By default and out-of-the-box, Oracle WebLogic Server stores users and groups in its embedded LDAP repository. Domains can be configured, however, to use identity data in other kinds of LDAP repositories, such as Oracle Internet Directory, ActiveDirectory, Novell eDirectory, and OpenLDAP. In addition, Oracle WebLogic Server provides a generic, default LDAP authenticator that can be used with other LDAP servers not in the preceding list.

Out-of-the-box, the security artifacts are stored in file-based stores; these stores can be moved (or reassociated) to an LDAP repository backed by an Oracle Internet Directory or to a DB-based repository backed by an Oracle Database.

1.3 Oracle ADF Security Overview

Oracle ADF is an end-to-end Java EE framework that simplifies development by providing out-of-the-box infrastructure services and a visual and declarative development experience.

Oracle ADF Security is based on the JAAS security model, and it uses OPSS. Oracle ADF Security supports a file-, LDAP-, or DB-based security store, uses permission-based fine-grained authorization provided by OPSS, and simplifies the configuration of application security with the aid of visual declarative editors and the Oracle ADF Security wizard, all of them available in Oracle JDeveloper.

Oracle ADF Security authorization allows protecting components (flows and pages), is integrated with Oracle JDeveloper at design time, and is available at run time when the application is deployed to the integrated server where testing of security features is typically carried out.

During the development of an Oracle ADF application, the authenticators are configured with the Oracle WebLogic Server Administration Console for the particular domain where the application is deployed, and the policy store is file-based. For deployment details, see Section 6.3.1, "Deploying to a Test Environment."

To summarize, Oracle ADF Security provides:

- Control over granular declarative security
- Visual and declarative development of security artifacts
- Assignment of simplified permission through a role hierarchy
- Use of EL (expression language) to access Oracle ADF resources
- Integration with Oracle JDeveloper that allows quick development and test cycles
- Rich Web user interfaces and simplified database access

For related information, see Scenario 2: Securing an Oracle ADF Application.

1.4 OPSS for Administrators

Depending on the application type, the guidelines to administer application security with Oracle WebLogic Administration Console, WLST commands, Fusion Middleware Control, or Oracle Entitlements Server are as follows:

- For Java EE applications, security is managed with Oracle WebLogic Administration Console, Oracle Entitlements Server, or WLST commands.
- For Oracle SOA, Oracle WebCenter, MDS, and Oracle ADF applications, authentication is managed with Oracle WebLogic Administration Console and authorization is managed with Fusion Middleware Control and Oracle Entitlements Server.
- For Java EE applications integrated with OPSS, authentication is managed using Oracle WebLogic Administration Console and authorization is managed with Fusion Middleware Control and Oracle Entitlements Server.

For details about security administration, see Chapter 5, "Security Administration."

1.5 OPSS for Developers

This section summarizes the main OPSS features typically used when securing applications, in the following scenarios:

- Scenario 1: Enhancing Security in a Java EE Application
- Scenario 2: Securing an Oracle ADF Application
- Scenario 3: Securing a Java SE Application

For other use cases, see Section 16.2, "Security Integration Use Cases."

1.5.1 Scenario 1: Enhancing Security in a Java EE Application

A Java EE application can be enhanced to use OPSS APIs such as the CSF, User and Role, or Policy Management: user attributes, such as a user's email, phone, or address, can be retrieved using the Identity Governance Framework API or the User and Role API; external system credentials (stored in a wallet or in a LDAP-based store) can be retrieved using the CSF API; authorization policy data can be managed with the policy management APIs; and application keys and certificates can be managed with Keystore Service APIs.

Java EE applications, such as servlets, JSPs, and EJBs, deployed on Oracle WebLogic Server can be configured to use authentication and authorization declaratively, with specifications in the file `web.xml`, or programmatically, with calls to `isUserInRole` and `isCallerInRole`.

Custom authenticators include the standard basic, form, and client certification methods. Authentication between servlets and EJBs is controlled using user roles and enterprise groups, typically stored in an LDAP repository, a database, or a custom authenticator.

1.5.2 Scenario 2: Securing an Oracle ADF Application

Oracle Application Development Framework (ADF) is a Java EE development framework available in Oracle JDeveloper that simplifies the development of Java EE applications by minimizing the need to write code that implements the application’s infrastructure, thus allowing developers to focus on the application features. Oracle ADF provides these infrastructure implementations as part of the Oracle JDeveloper framework, therefore enhancing the development experience with visual and declarative approaches to Java EE development.

Oracle ADF implicitly uses OPSS, and, for most part, the developer does not have to code directly to OPSS APIs; of course, the developer can nevertheless use direct calls to OPSS APIs.

Oracle ADF leverages container authentication and subsequently uses JAAS based authorization to control access to Oracle ADF resources. These authorization policies may include application-specific roles and JAAS authorization permissions. Oracle ADF connection credentials are stored securely in the credential store.

Oracle ADF and Oracle WebCenter applications deployed on Oracle WebLogic Server include WebLogic authenticators, such as the default WebLogic authenticator, and may include a single sign-on solution (Oracle Access Manager or Oracle Application Server Single Sign-On).

Usually, applications also use one or several of the following OPSS features: anonymous and authenticated role support, policy management APIs, and the Credential Store Framework.

For details about these topics, see the following sections:

- Section 2.3, "About the Authenticated Role"
- Section 2.4, "About the Anonymous User and Role"
- Section 3.3, "Policy Store Basics"
- Section 3.4, "Credential Store Basics"

For details on how to develop and secure Oracle ADF applications, see chapter 29 in *Developing Fusion Web Applications with Oracle Application Development Framework*.

1.5.3 Scenario 3: Securing a Java SE Application

Most of the OPSS features that work in Java EE applications work in Java SE applications, but there are some differences, which are noted in this section.

Starting Up OPSS Services

Java SE applications must use the OPSS method `AppSecurityContext.JpsStartup.start()` before invoking any OPSS security operations. For details, see Section 25.1, "Using OPSS in Java SE Applications."

Configuration

All OPSS-related configuration and data files are located under configuration directory in the domain home. For example, the configuration file for a Java SE environment is defined in the file `jps-config-jse.xml` by default installed in the following location:

```
$DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml
```

To specify a different location, use the following switch:

```
-Doracle.security.jps.config=pathToConfigFile
```

The syntax of this file is identical to that of the file `jps-config.xml`. This file is used by code running in WebLogic containers. For details, see Appendix A, "OPSS Configuration File Reference."

For details about security configuration for Java SE applications, see Chapter 25, "Configuring Java SE Applications to Use OPSS."

Required JAR in Class Path

To make OPSS services available to a Java SE application, ensure that the following JAR file is added to your class path, located in the modules area of the Oracle installation home:

```
$ORACLE_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-manifest.jar
```

Login Modules

Java SE applications can use standard JAAS login modules. However, to use the same login module on WLS, implement a custom authentication provider that invokes the login module. The SSPI interfaces allow integrating custom authentication providers in WLS.

The login module recommended for Java SE applications is the IdentityStore login module. For details about this and other login modules, see Section 25.3.3, "Login Modules."

For details, see section Authentication Providers in *Developing Security Providers for Oracle WebLogic Server*.

2 Understanding Users and Roles

This chapter describes various characteristics of users and roles, such as the anonymous role, the authenticated role, role mapping, and the role category. It also includes the definition of terms used throughout this guide.

OPSS delegates authentication to Oracle WebLogic Server authenticator providers managed with the WebLogic Administration Console.

This chapter is divided into the following sections:

- Terminology
- Mapping Roles
- About the Authenticated Role
- About the Anonymous User and Role
- About Administrative Users and Roles
- Managing User Accounts
- About Principal Name Comparison Logic
- About the Role Category

For further details about managing users and roles programmatically, see Chapter 21, "Developing with the Identity Directory API."

2.1 Terminology

This section defines most of the OPSS security terms.

Users

A *user*, or *enterprise user*, is an end-user accessing a service. User information is stored in the identity store. An *authenticated user* is a user whose credentials have been validated.

An *anonymous user* is a user whose credentials have not been validated (hence unauthenticated) that is permitted access to only unprotected resources. This user is specific to OPSS and its use can be enabled or disabled by an application. For details about anonymous user support, see Section 2.4, "About the Anonymous User and Role."

Roles

An *enterprise role* or *enterprise group* is a collection of users and other groups. It can be hierarchical, that is, a group can include arbitrarily nested groups (other than itself).

A Java EE *logical role* is a role specified declaratively or programmatically by a Java EE application. It is defined in an application deployment descriptor and, typically, used in the application code. It can be mapped to only enterprise groups or users, and it cannot be mapped directly to application roles.

An *application role* is a collection of users, groups, and other application roles; it can be hierarchical. Application roles are defined by application policies and not necessarily known to a Java EE container. Application roles can be many-to-many mapped to external roles. For example, the external group `employee` (stored in the identity store) can be mapped to the application role `helpdesk service request` (in one stripe) and to the application role `self service HR` (in another stripe).

For details about the *anonymous role*, see Section 2.4, "About the Anonymous User and Role." For details about the *authenticated role*, see Section 2.3, "About the Authenticated Role."

Principal

A *principal* is the identity to which the authorization in the policy is granted. A principal can be a user, an external role, or an application role. Most frequently, it is an application role.

Application and System Policies

An *application policy* is a functional policy that specifies a set of permissions that a principal is allowed to perform within the application, such as viewing web pages or modifying reports.

An application policy uses:

- Principals as grantees, and must have at least one principal.
- Either one or more permissions, or an entitlement, but not both.

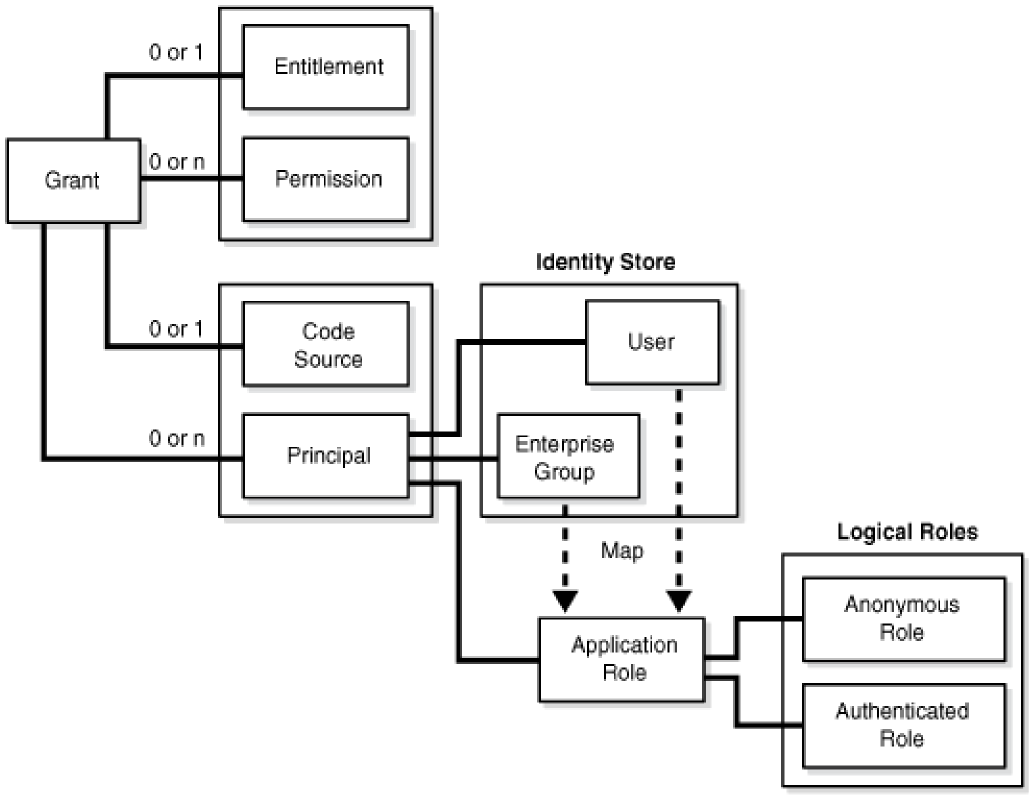
  Policies that use an entitlement are called *entitlement-based policies*; policies that use one or more permissions are called *resource-based policies*.

A *system policy* is a policy that specifies a set of permissions that a principal or a code source is allowed to perform, and it holds for an entire domain. System policies grant privileges to code sources and principals, while application policies can grant privileges to principals only. For an example of a source code system policy with multiple permissions, see Section 24.6.6, "Using Supported Permission Classes."

Application and system policies are packed in the application's jazn-data.xml file. In that file, system policies should be defined *outside* the application's scope.

Figure 2–1 illustrates the application policy model.

Figure 2–1 Application Policy Logical Model



OPSS Subject

An OPSS *subject* is a collection of principals and, possibly, user credentials such as passwords or cryptographic keys. The server authentication populates the subject with users and groups, and then augments the subject with application roles. The OPSS Subject is key in identity propagation using other Oracle Identity Management products such as OAM, for example. For details about how anonymous data is handled, see Section 2.4.1, "Anonymous Support and Subject."

Security Stores

The *identity store* is the repository of enterprise users and groups and must be LDAP-based. Out-of-the-box the identity store is the WebLogic LDAP DefaultAuthenticator. Other types of identity stores include Oracle Internet Directory, Sun Directory Server, and Oracle Virtual Directory.

The *policy store* is the repository of application and system policies. This store is administered with Oracle Enterprise Manager Fusion Middleware Control.

The *credential store* is the repository of credentials. This store is administered with Oracle Enterprise Manager Fusion Middleware Control.

The *OPSS security store* is the logical repository of system and application-specific policies, credentials, audit metadata, and keys. The only type of LDAP-based OPSS security store supported is Oracle Internet Directory; the only type of DB-based OPSS security store supported is Oracle database.

For details, see Chapter 3, "Understanding Identities, Policies, Credentials, Keys, Certificates, and Auditing."

Other Terms

A *system component* is a manageable process that is not a WebLogic component. Examples include Oracle Internet Directory, WebCache, and Java SE components.

A *Java component* is a peer of a system component, but managed by an application server container. Generally it refers to a collection of applications and resources in one-to-one relationship with a domain extension template. Examples include Oracle SOA applications, Oracle WebCenter Spaces.

2.2 Mapping Roles

OPSS supports many-to-many mapping of application roles in the OPSS security store to enterprise groups in the identity store, which allows users in enterprise groups to access application resources as specified by application roles. Since this mapping is many-to-many, it is alternatively referred to as the role-to-group mapping or as the group-to-role mapping.

Notes: Oracle JDeveloper allows specifying this mapping when the application is being developed in that environment. Alternatively, the mapping can be also specified, after the application has been deployed, using WLST commands, Fusion Middleware Control, or Oracle Entitlements Server, as explained in Section 10.3.2, "Managing Application Roles."

The mapping of an application role to an enterprise group rewrites the privilege of the enterprise group as the union of its privileges and those of the mapped application role. Therefore, it (possibly) augments the privileges of the enterprise group but never removes any from it.

2.2.1 Permission Inheritance and the Role Hierarchy

OPSS roles can be structured hierarchically by the relation “is a member of.” Thus a role can have as members users or *other* roles.

Important: When building a role hierarchy, ensure that you do not introduce circular dependencies to prevent unwanted behavior. For example, setting roleA to be a member of roleB, and roleB to be a member of roleA would create such a circular dependency.

In a role hierarchy, role members inherit permissions from the parent role. Thus, if roleA is a member of roleB, then all permissions granted to roleB are also granted to roleA. Of course, roleA may have its own particular permissions, but, just by being a member of roleB, roleA inherits all the permissions granted to roleB.

For details about managing an application role hierarchy with WLST commands, see grantAppRole and revokeAppRole in *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

For details about managing an application role hierarchy with Oracle Entitlements Server, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

The following example illustrates a role hierarchy consisting of the following nested application users and roles:

- The role `developerAppRole` has the following members:

```
developer
developer_group
managerAppRole
directorAppRole
```

- In addition, the role `directorAppRole` has the following members:

```
developer
developer_group
```

Here is the relevant portions of the file `jazn-data.xml` specifying the above hierarchy:

```
<policy-store>
  <applications>
    <application>
      <name>MyApp</name>
      <app-roles>
        <app-role>
          <name>developerAppRole</name>
          <class>oracle.security.jps.service.policystore.ApplicationRole</class>
          <display-name>Application developer role</display-name>
          <description>Application developer role</description>
          <guid>61FD29C0D47E11DABF9BA765378CF9F5</guid>
          <members>
            <member>
              <class>weblogic.security.principal.WLSUserImpl</class>
              <name>developer</name>
            </member>
            <member>
              <class>weblogic.security.principal.WLSUserImpl</class>
              <name>directorAppRole</name>
            </member>
            <member>
              <class>weblogic.security.principal.WLSGroupImpl</class>
              <name>developer_group</name>
            </member>
            <member>
              <class>
oracle.security.jps.service.policystore.ApplicationRole</class>
              <name>managerAppRole</name>
            </member>
          </members>
        </app-role>
        <app-role>
          <name>directorAppRole</name>
          <class>oracle.security.jps.service.policystore.ApplicationRole</class>
          <display-name>Application director role </display-name>
          <description>Application director role</description>
          <guid>61FD29C0D47E11DABF9BA765378CF9F8</guid>
          <members>
            <member>
              <class>weblogic.security.principal.WLSUserImpl</class>
              <name>developer</name>
            </member>
            <member>
              <class>weblogic.security.principal.WLSGroupImpl</class>
              <name>developer_group</name>
            </member>
           </members>
         </app-role> ...
```

```
       </app-roles>

      <jazn-policy>
        <grant>
          <grantee>
             <principals>
                <principal>
                   <class>
           oracle.security.jps.service.policystore.ApplicationRole</class>
                   <name>developerAppRole</name>
                </principal>
             </principals>
          </grantee>
          <permissions>
            <permission>
              <class>java.io.FilePermission</class>
              <name>/tmp/oracle.txt</name>
              <actions>write</actions>
             </permission>
           </permissions>
         </grant>

         <grant>
           <grantee>
             <principals>
               <principal>
                 <class>
           oracle.security.jps.service.policystore.ApplicationRole</class>
                 <name>managerAppRole</name>
               </principal>
             </principals>
           </grantee>
           <permissions>
             <permission>
               <class>java.util.PropertyPermission</class>
               <name>myProperty</name>
               <actions>read</actions>
             </permission>
            </permissions>

           </grant>
           <grant>
             <grantee>
               <principals>
                 <principal>
                   <class>
oracle.security.jps.service.policystore.ApplicationRole</class>
                   <name>directorAppRole</name>
                 </principal>
               </principals>
             </grantee>
             <permissions>
               <permission>
                 <class>foo.CustomPermission</class>
                 <name>myProperty</name>
                 <actions>*</actions>
               </permission>
             </permissions>
           </grant>
         </jazn-policy>
```

```
    </policy-store>
```

Table 2–1 summarizes the permissions that each of the five users and roles in the above hierarchy gets according the inheritance rule:

Table 2–1 Granted and Inherited Permissions

Role	Permission Granted	Actual Permissions
developerAppRole	P1=java.io.FilePermission	P1
managerAppRole	P2= java.util.PropertyPermission	P2 and (inherited) P1
directorAppRole	P3=foo.CustomPermission	P3 and (inherited) P1
developer		P1 and P3 (both inherited)
developer_group		P1 and P3 (both inherited)

2.3 About the Authenticated Role

OPSS supports the use of a special role: the authenticated role. This role has the following characteristics:

- It need not be declared in any configuration file.
- It is always represented by a principal attached to a subject after a successful authentication. In another words: it is granted by default to any authenticated user.
- Its presence, within a subject, is mutually exclusive with the anonymous role, that is, either (a) a subject has *not* gone through authentication, in which case it contains a principal with the anonymous role as explained in Anonymous Support and Subject or (b) the subject has gone through authentication successfully, in which case it contains the authenticated role and, depending on the configuration, the anonymous role.
- It is an application role and, therefore, it can be used by any application and participate in the application's role hierarchy.

The permissions granted to the authenticated role need not be specified explicitly but are implicitly derived from the enterprise groups and application roles of which it is a member.

A typical use of the authenticated role is to allow authenticated users access to common application resources, that is, resources available to a user that has been authenticated.

For details on how an application can manually configure the use of the authenticated role, see Section 24.3, "Configuring the Servlet Filter and the EJB Interceptor."

2.4 About the Anonymous User and Role

OPSS supports the use of two special entities: the anonymous user and the anonymous role. Like the authenticated role, these entities need not be declared and applications configure their use in the JpsFilter or JpsInterceptor. Any of them can be used by an application in the application's role hierarchy.

When enabled, before the user is authenticated and while the user is accessing unprotected resources, the user is represented by a subject populated with just the anonymous user and the anonymous role. Eventually, if that subject attempts access to a *protected* resource, then authorization handles the subject as explained in Anonymous

Support and Subject.

The permissions granted to the anonymous user and role need not be specified explicitly but are implicitly derived from the enterprise groups and application roles of which they are a member.

A typical use of the anonymous user and role is to allow unauthenticated users to access public, unprotected resources.

For details on how an application can manually configure the use of the anonymous user and role, see Section 24.3, "Configuring the Servlet Filter and the EJB Interceptor."

2.4.1 Anonymous Support and Subject

Throughout this section, it is assumed that the use of the anonymous user and anonymous role are enabled.

When an end-user first accesses an unprotected resource, the system creates a subject and populates it with two principals corresponding with the anonymous user and the anonymous role. While unprotected resources are involved, that subject is not modified and authentication does not take place.

When a protected resource is accessed, then authentication kicks in, and the subject (which thus far contained just the anonymous role) is modified according to the result of the authentication process, as follows.

If authentication is successful, then:

1. The anonymous user is removed from the subject and replaced, as appropriate, by an authenticated user.
2. The anonymous role is removed and the authenticated role is added.
3. Other roles are added to the subject, as appropriate.

Notice that a successful authentication results then in a subject that has exactly one principal corresponding to a non-anonymous user, one principal corresponding to the authenticated role, and possibly other principals corresponding to enterprise or application roles.

If authentication is not successful, then the anonymous user is retained, the anonymous role is removed or retained (according to how the application has configured the JpsFilter or JpsInterceptor), and no other principals are added. By default, the anonymous role is removed from the subject.

2.5 About Administrative Users and Roles

A WebLogic administrator is any user member of the group Administrators, and any user that exists in a security realm can be added to this group.

For details about the default groups that exist in a security realm, see section Users, Groups, And Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

Generally, there is no default name for an administrator, with just one exception: when you install the examples, you get a default user name and password for the administrator of the sample domain. It is recommended, however, that these examples not be used in any production environment.

For details, see section Install WebLogic Server in a Secure Manner in *Securing a Production Environment for Oracle WebLogic Server*.

Once a domain is configured, users that have been created in the security realm can be added or removed from the Administrators group at anytime by any member of the Administrators group. The basic tools for managing these accounts are the Oracle WebLogic Administration Console, the Oracle WebLogic Scripting Tool (WLST), and Fusion Middleware Control.

For details, see section Add Users to Groups in *Oracle WebLogic Server Administration Console Online Help*, section Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*.

2.6 Managing User Accounts

For information about creating user accounts and protecting their passwords, consult the following documents:

- For general guidelines on creating passwords, see section Manage Users and Groups in *Oracle WebLogic Server Administration Console Online Help*. The default authentication provider requires a minimum password length of 8 characters, but this is configurable.

  A few recommendations regarding password creation are explained in section Securing the WebLogic Server Host in *Securing a Production Environment for Oracle WebLogic Server*.
- In general, passwords are stored in either an LDAP server or an RDBMS. The particular location in which they are stored is determined by the specific authentication provider that is configured in the environment (or more precisely, the security realm of a domain). For details about out-of-the-box authentication providers, see section Managing the Embedded LDAP Server in *Administering Security for Oracle WebLogic Server*.
- For information about how to configure the optional Password Validation provider, which is automatically called whenever you create a password and that enforces a set of customizable password composition rules, see section Configuring the Password Validation Provider in *Administering Security for Oracle WebLogic Server*.
- When adding or deleting a user, consider the recommendations explained in Section J.5.4, "User Gets Unexpected Permissions."

2.7 About Principal Name Comparison Logic

This section explains how principal comparison affects OPSS authorization and describes the system parameters that control the principal name comparison logic, in the following sections:

- How Does Principal Comparison Affect Authorization?
- System Parameters Controlling Principal Name Comparison

2.7.1 How Does Principal Comparison Affect Authorization?

Upon a successful user authentication, the system populates a Subject with principals whose names accord with the user and enterprise group names (of enterprise groups the user is included in) stored in the identity store.

On the other hand, when the user (or enterprise group) needs to be authorized, the system considers how application roles have been mapped to enterprise groups, and builds another set of principals from names in application grants stored in the OPSS security store.

In order to authorized a principal, the principal names populated in the Subject (from names found in the identity store) and those built from names in the OPSS security store are compared. The user (or group) is authorized if and only if a match of principal names is found.

It is therefore crucial that principal names be compared properly for the authorization provider to work as expected.

Suppose, for instance, a scenario where the identity store contains the user name "jdoe", but, in grants, that user is referred to as "Jdoe". Then one would want the principal name comparison to be case *insensitive*, for otherwise the principals built from the names "jdoe" and "Jdoe" will not match (that is, they will be considered distinct) and the system will not authorize "jdoe" as expected.

2.7.2 System Parameters Controlling Principal Name Comparison

The following two WebLogic Server system parameters control the way principal names are compared in a domain and allow, furthermore, to compare principals using DN and GUID data:

```
PrincipalEqualsCaseInsensitive (True or False; False by default)
PrincipalEqualsCompareDnAndGuid (True or False; False by default)
```

To set these parameters using the WebLogic Server Console, proceed as follows:

1. In the left pane of the Console, under **Domain Structure**, select the domain for which you intend to set the parameters above.
2. Select **Configuration > Security** and click **Advanced**.
3. Set the following entries to true or fase, as appropriate:
   - **Principal Equals Case Insensitive**
   - **Principal Equals Compare DN and GUID**
4. Restart the server. Changes do not take effect until the server is restarted.

These parameters can alternatively be set using WLST commands. For more details about configuring the WebLogic server, see section Configuring a Domain to Use JAAS Authorization in *Administering Security for Oracle WebLogic Server*.

The name comparison logic chosen at runtime is described by the following pseudo-code fragment:

```
if PrincipalEqualsCompareDnAndGuid is true
//use GUID and DN to compare principals
{
  when GUID is present in both principals {
     use case insensitive to compare GUIDs
  }
  when DN is present in both principals {
     use case insensitive to compare DNs
  }
}

if PrincipalEqualsCaseInsensitive is true
//use just name to compare principals
{
  use case insensitive to compare principal names
}
else
{
  use case sensitive to compare principal names
}
```

Since by default both `PrincipalEqualsCompareDnAndGuid` and `PrincipalEqualsCaseInsensitive` are false, name principal comparison defaults to case sensitive.

2.8 About the Role Category

The role category allows a security administrator to organize application roles. Rather than displaying the flat list of roles in an application, an administrator can organize them arbitrarily in sets or categories. Role categories are flat sets, that is, they contain no hierarchical structure.

For details about managing an application role category with Oracle Entitlements Server, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

The following fragment illustrates the configuration of a role category:

```
<app-roles>
  <app-role>
    <name>AppRole_READONLY</name>
    <display-name>display name</display-name>
    <description>description</description>
    <class>oracle.security.jps.service.policystore.ApplicationRole</class>
    <extended-attributes>
      <attribute>
        <name>ROLE_CATEGORY</name>
        <values>
          <value>RC_READONLY</value>
        </values>
      </attribute>
    </extended-attributes>
  </app-role>
</app-roles>
<role-categories>
  <role-category>
    <name>RC_READONLY</name>
    <display-name>RC_READONLY display name</display-name>
    <description>RC_READONLY description</description>
  </role-category>
</role-categories>
```

The members of a role category are *not* configured within the `<role-category>` element but within the element `<extended-attributes>` of the corresponding application role. The role category name is case insensitive. The role category can be managed with the interface `RoleCategoryManager`.

For details about the `RoleCategoryManager` interface, see the Javadoc document *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services*.

3 Understanding Identities, Policies, Credentials, Keys, Certificates, and Auditing

Applications use the identity, policy, credential stores, keystores, and audit repository configured in the domain in which they run. This chapter introduces the following basic concepts: identity, policy, credential, keystore, and audit data. In addition, it presents a matrix that shows the compatible versions of security artifacts.

This chapter is divided into the following sections:

- Compatibility Matrix for 11g and 12c Versions
- Authentication Basics
- Policy Store Basics
- Credential Store Basics
- Keystore Service Basics
- Audit Service Basics

For definitions of the terms used in this chapter, see Section 2.1, "Terminology."

For scenarios illustrating the use of stores, see Chapter 4, "About Oracle Platform Security Services Scenarios."

3.1 Compatibility Matrix for 11g and 12c Versions

This section states the compatible versions of binaries, configurations, schemas, and stores for releases 11.1.1.5.0, 11.1.1.6.0, 11.1.1.7.0, 11.1.1.9.0, 12.1.2.0.0. and 12.1.3.0.0. The compatible versions of these artifacts apply to both DB-based and OID-based OPSS stores.

The OPSS security store is a composite store, that is, a store that includes all security artifacts such as policies, keys, credentials, and audit metadata. In DB-based stores, exactly one logical security store is assumed per DB schema.

The following table shows the versions compatible and it applies to both DB-based and OID-based security stores. Note the following terminology symbols:

- The prefix => next to a version number denotes a version *equal to or higher than* the stated version number.
- The prefix > next to a version number denotes a version *higher than* the stated version number.
- The prefix < next to a version number denotes a version *lower than* the stated version number.

Binary	Configuration	Schema	Store	Status
11.1.1.5.0	11.1.1.5.0	=>11.1.1.5.0	11.1.1.5.0	Certified
11.1.1.5.0	11.1.1.5.0	>11.1.1.5.0	>11.1.1.5.0	Not supported
11.1.1.6.0	11.1.1.5.0	=>11.1.1.5.0	11.1.1.5.0	Certified
11.1.1.6.0	11.1.1.5.0	>11.1.1.5.0	>11.1.1.5.0	Not supported
11.1.1.6.0	11.1.1.6.0	=>11.1.1.6.0	11.1.1.6.0	Certified
11.1.1.6.0	11.1.1.6.0	>11.1.1.6.0	>11.1.1.6.0	Not supported
11.1.1.7.0	11.1.1.7.0	=>11.1.1.6.0	<11.1.1.7.0	Not supported
11.1.1.7.0	11.1.1.6.0	=>11.1.1.6.0	11.1.1.6.0	Certified
11.1.1.7.0	11.1.1.6.0	>11.1.1.6.0	>11.1.1.6.0	Not supported
11.1.1.7.0	11.1.1.7.0	=>11.1.1.7.0	11.1.1.7.0	Certified
11.1.1.9.0	11.1.1.7.0 11.1.1.6.0 11.1.1.5.0	=>11.1.1.7.0 =>11.1.1.6.0 =>11.1.1.5.0	11.1.1.7.0 11.1.1.6.0 11.1.1.5.0	Certified
11.1.1.9.0	11.1.1.9.0	=>11.1.1.9.0	11.1.1.9.0	Certified
12.1.2.0.0	12.1.2.0.0	=>12.1.2.0.0	12.1.2.0.0	Certified (schema only upgrade)
12.1.2.0.0	<12.1.2.0.0	<12.1.2.0.0	<12.1.2.0.0	Not supported
12.1.3.0.0	12.1.3.0.0	=>12.1.3.0.0	12.1.3.0.0	Certified (schema only upgrade)
12.1.3.0.0	<12.1.3.0.0	<12.1.3.0.0	<12.1.3.0.0	Not supported

3.2 Authentication Basics

OPSS uses server authentication providers, components that validate user credentials or system processes based on a user name-password combination or a digital certificate. Authentication providers also make user identity information available to other components in a domain (through subjects) when needed.

Java EE applications use LDAP- or DB-based authentication providers. Java SE applications use a file-based identity store out-of-the-box, but they can be configured to use also LDAP- or DB-based authenticator providers.

For further details, see section Authentication in *Understanding Security for Oracle WebLogic Server*.

This section covers the following topics:

- Identity Store Types and WebLogic Authenticators
- WebLogic Authenticators

3.2.1 Identity Store Types and WebLogic Authenticators

The information in this section applies only to Java EE applications. For details about using authenticators in Java SE applications, see Section 25.3.2, "Configuring an LDAP Identity Store in Java SE Applications."

The following list enumerates the repositories supported for an identity store (note that all are LDAP-based except for the last one):

- Oracle Internet Directory 11g
- Oracle Virtual Directory
- Open LDAP 2.2
- Oracle Unified Directory 11gR1
- Oracle Directory Server Enterprise Edition 11.1.1.3.0
- Sun DS 6.3, 7.0
- Novell eDirectory 8.8
- Tivoli Access Manager
- Active Directory
- Active Directory AM
- Active Directory 2008
- Oracle DB 10g, 11gR1, 11gR2

Open LDAP 2.2 requires a special set up; for details see Appendix H, "Using an OpenLDAP Identity Store."

For support for reference integrity in Oracle Internet Directory servers, see Important note Section 9.2, "Using an LDAP-Based OPSS Security Store."

The following table lists the WebLogic authenticators to use with each of the repositories:

Table 3–1 Identity Store Repositories and Authenticators

Store Type	WebLogic Authenticator
Oracle Internet Directory 11g	OracleInternetDirectoryAuthenticator
Oracle Virtual Directory	OracleVirtualDirectoryAuthenticator
Open LDAP 2.2	OpenLDAPAuthenticator
Oracle Unified Directory 11gR1	iPlanetAuthenticator
Oracle Directory Server Enterprise Edition 11.1.1.3.0	iPlanetAuthenticator
Sun DS 6.3, 7.0	iPlanetAuthenticator
Novell eDirectory 8.8	NovellAuthenticator
Tivoli Access Manager	OpenLDAPAuthenticator
Active Directory	ActiveDirectoryAuthenticator
Active Directory AM	ActiveDirectoryAuthenticator
Active Directory 2008	ActiveDirectoryAuthenticator
Oracle DB 10g, 11gR1, 11gR2	CustomDBMSAuthenticator ReadOnlySQLAuthenticator SQLAuthenticator

To create and configure a new authenticator using the WebLogic console, proceed as follows:

1. Create a new authenticator. For details see list of topics following this procedure.

Depending on your particular identity store repository and according to Table 3–1, select the apporpriate WebLogic authenticator; the parameters you enter depend on the authenticator selected; for example, in case of OidAuthenticator, you enter the host information, as illustrated in the following sample:

```
host name: example.com
port: 5555
principal: cn=orcladmin
credential: myPassword
user base DN: cn=Users,dc=us,dc=oracle,dc=com
group base DN: cn=Groups,dc=us,dc=oracle,dc=com
```

2. Save your input to create a new authenticator.
3. Set the appropriate control flag in the created authenticator.
4. Reorder the list of authenticators in your domain, as appropriate.
5. Restart the WebLogic server.

Important: Any LDAP-based authenticator used in a domain, other than the DefaultAuthenticator, requires that the flag `UseRetrievedUserNameAsPrincipal` be set. Out-of-the-box, this flag is set in the DefaultAuthenticator.

For further information about WebLogic authenticators, consult the following topics:

- Chapter 4, Authentication Providers in *Developing Security Providers for Oracle WebLogic Server*.
- Configuring Authentication Providers in *Administering Security for Oracle WebLogic Server*, and section Configure Authentication and Identity Assertion providers in *Oracle WebLogic Server Administration Console Online Help*.
- Section Configure Authentication and Identity Assertion providers in *Oracle WebLogic Server Administration Console Online Help*.

3.2.2 WebLogic Authenticators

By default and out-of-the-box, Oracle WebLogic Server stores users and groups in the DefaultAuthenticator. This authenticator is setup to use `cn` as the default attribute.

The data stored in any LDAP authenticator can be accessed by the User and Role API to query user profile attributes, but custom code may be required to query non-LDAP identity store repositories.

This section contains the following topics:

- Multiple Authenticators
- Additional Authentication Methods

For details about X.509 identity assertion, see section How an LDAP X509 Identity Assertion Provider Works in *Administering Security for Oracle WebLogic Server.*

For details about authentication using the SAML 1.1 or SAML 2.0 identity assertion provider, see section Configuring the SAML Authentication Provider in *Administering Security for Oracle WebLogic Server.*

3.2.2.1 Multiple Authenticators

Oracle WebLogic Server allows the configuration of multiple authenticators in a given context. One of them must be an LDAP-based authenticator.

OPSS initializes the identity store service with the LDAP authenticator chosen from the list of configured authenticators according to the following algorithm:

1. Consider the subset of LDAP authenticators configured. Note that, since the context is assumed to contain at least one LDAP authenticator, this subset is not empty.
2. Within that subset, consider those that have set the maximum flag. The flag ordering used to compute this subset is the following:

   ```
   REQUIRED > REQUISITE > SUFFICIENT > OPTIONAL
   ```

   Again, this subset (of LDAPs realizing the maximum flag) is not empty.
3. Within that subset, consider the first configured in the context.

The LDAP authenticator singled out in step 3 is the one chosen to initialize the identity store service. For details about host name verification when establishing an SSL connection with an LDAP authenticator, see *Securing a Production Environment for Oracle WebLogic Server*.

For details about the default values that OPPS uses to initialize the various supported LDAP authenticators, see javadoc User and Role API documentation in Section G.1, "OPSS API References." If a service instance initialization value is provided by default and also (explicitly) in the service instance configuration, the value configured takes precedence over the default one.

3.2.2.2 Additional Authentication Methods

The WebLogic Identity Assertion providers support certificate authentication using X.509 certificates, SPNEGO tokens, SAML assertion tokens, and CORBA Common Secure Interoperability version 2 (CSIv2) identity assertion.

The Negotiate Identity provider is used for SSO with Microsoft clients that support the SPNEGO protocol. This provider decodes SPNEGO tokens to obtain Kerberos tokens, validates the Kerberos tokens, and maps Kerberos tokens to WebLogic users.

For general information about identity assertion providers, see section Identity Assertion Providers in *Understanding Security for Oracle WebLogic Server*.

For an overview of SSO with Microsoft clients, see section Overview of Single Sign-On with Microsoft Clients in *Administering Security for Oracle WebLogic Server*.

For details about Kerberos identification, see section Creating a Kerberos Identification for WebLogic Server in *Administering Security for Oracle WebLogic Server*.

3.3 Policy Store Basics

A Java 2 policy specifies the permissions granted to signed code loaded from a given location.

A JAAS policy extends Java 2 grants by allowing an optional list of principals; the semantics of the permissions are granted to only code from a given location, possibly signed, and run by a user represented by those principals.

JACC extends the Java 2 and JAAS permission-based policy to EJBs and Servlets by defining an interface to plug custom authorization providers, that is, pluggable components that allow the control and customizing of authorizations granted to running Java EE applications.

An application policy is a collection of Java 2 and JAAS policies, which is applicable to just that application (in contrast to a Java 2 policy, which is applicable to the whole JVM).

The policy store is a repository of system and application-specific policies and roles. Application roles can include enterprise users and groups specific to the application (such as administrative roles). A policy can use any of these groups or users as principals.

In the case of applications that manage their own roles, Java EE application roles (configured in files `web.xml` or `ejb-jar.xml`) get mapped to enterprise users and groups and used by application-specific policies.

Policy Store Types

A policy store can be file-, LDAP-, or DB-based. A file-based policy store is an XML file, and this store is represented by the out-of-the-box policy store provider. The only LDAP-based policy store type supported is Oracle Internet Directory. The only DB-based policy store type supported is Oracle RDBMS.

Policy Store Scope, Migration, and Reassociation

There is exactly one policy store per domain. During development, application policies are file-based and specified in the file `jazn-data.xml`.

When the application is deployed on WebLogic with Fusion Middleware Control, they can be automatically migrated into the policy store. For details about this feature, see Section 9.6.1, "Migrating with Fusion Middleware Control." By default, the policy store is file-based.

For reassociation details, see Section 9.5, "Reassociating the OPSS Security Store."

Note: All permission classes must be specified in the system class path.

For details about the resource catalog support within a policy store, see Section 17.3.1, "The Resource Catalog."

3.4 Credential Store Basics

A credential store is a repository of security data (credentials) that certify the authority of users, Java components, and system components. A credential can hold user name and password combinations, tickets, or public key certificates. This data is used during authentication, when principals are populated in subjects, and, further, during authorization, when determining what actions the subject can perform.

OPSS provides the Credential Store Framework, a set of APIs that applications can use to create, read, update, and manage credentials securely.

Credential Store Types

A credential store can be file-, LDAP-, or DB-based. A file-based credential store, also referred to as wallet-based and represented by the file `cwallet.sso`, is the out-of-the-box credential store. The only LDAP-based credential store type supported is Oracle Internet Directory. The only DB-based credential store type supported is Oracle RDBMS.

For versions supported see Oracle Fusion Middleware Supported System Configurations at `http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`.

Credential Store Scope, Migration, and Reassociation

An application can use either the domain credential store or its own wallet-based credential store. The domain credential store can be wallet-based (by default), LDAP-, or DB-based. The only LDAP-based credential store type supported is Oracle Internet Directory.

The migration of application credentials to the credential store can be configured to take place automatically when the application is deployed. For details, see Section 9.6.1, "Migrating with Fusion Middleware Control."

Credentials can also be reassociated from one type of store to another. For details, see Section 9.5, "Reassociating the OPSS Security Store."

3.5 Keystore Service Basics

The Keystore Service provides a central repository for keystores and trust stores containing all the keys and certificates used by a domain's components and applications. This eliminates the need to associate keystores with individual applications.

The administrator works with a single user interface providing a unified way to view and manage all keystores.

3.5.1 Keystore Repository Types

The central repository can be any of the following:

- XML file-based

  This is the out-of-the-box keystore repository, and it is named keystores.xml.

  Note: This file is not present immediately after installation; rather, it is generated the first time the Administration Server is started.

- Oracle Database
- Oracle Internet Directory (*Note*: No other LDAP servers are supported.)

See Also: Chapter 12 for details about the Keystore Service.

3.5.2 Keystore Repository Scope and Reassociation

Keys and certificates in the domain keystore repository can be reassociated from one type to another. For details, see Section 9.5, "Reassociating the OPSS Security Store".

3.6 Audit Service Basics

The Audit Service supports a central repository of audit records for the domain. Administrators can conveniently leverage the service to audit events triggered by configuration changes as well as operational activity for components and deployed applications.

Audit Repository Types

The repository for audit records can be:

- file-based
- Oracle Database

See Also: Chapter 13 for details about the audit service.

4 About Oracle Platform Security Services Scenarios

This chapter describes some typical security scenarios supported by Oracle Platform Security Services. It also includes the list of LDAP, DB, and XML servers supported, the management tools that an administrator would use to administer security data in each scenario, and the package requirements for policies and credentials.

These topics are explained in the following sections:

4.1 Supported File-, LDAP-, and DB-Based Services

Oracle Platform Security Services supports the following repositories:

- For the OPSS security store:
  - File-based, XML for the OPSS security store and cwallet for the credential store.
  - LDAP-based, Oracle Internet Directory.
  - DB-based, Oracle RDBMS.
- For the identity store, any of the LDAP authenticators supported by the Oracle WebLogic Server. An XML identity store is supported in only Java SE applications.
- For keystores:
  - File-based, XML file.
  - LDAP-based, Oracle Internet Directory.
  - DB-based, Oracle RDBMS.

For versions supported see Oracle Fusion Middleware Supported System Configurations at `http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`.

Important: If using Oracle Internet Directory 10.1.4.3 with OPSS, a mandatory one-off patch for bug number 8351672 is required on top of Oracle Internet Directory 10.1.4.3. For a list of patches to various versions of Oracle Internet Directory, see Chapter 9.2, "Using an LDAP-Based OPSS Security Store."

To ensure optimal performance, the following Oracle Internet Directory tuning is recommended:

```
ldapmodify -D cn=orcladmin -w <password> -v <<EOF
dn: cn=dsaconfig,cn=configsets,cn=oracle internet directory
changetype: modify
add: orclinmemfiltprocess
orclinmemfiltprocess: (objectclass=orcljaznpermission)
orclinmemfiltprocess: (objectclass=orcljazngrantee)
EOF
```

For details about LDAP authenticators, see section Configuring LDAP Authentication Providers in *Administering Security for Oracle WebLogic Server*. In particular, the DefaultAuthenticator is available out-of-the-box, but its use is recommended only in developing environments with no more than ten thousand users and with no more than twenty five hundred groups.

Policies, credentials, and keys stored in an LDAP-based store must use the same physical persistent repository. For details, see the following chapters:

- Chapter 10, "Managing the Policy Store"
- Chapter 11, "Managing the Credential Store"
- Chapter 12, "Managing Keys and Certificates with the Keystore Service"

4.2 Management Tools

The tools available to a security administrator are the following:

- WebLogic Administration Console
- Oracle Enterprise Manager Fusion Middleware Control
- Oracle Entitlements Server
- WLST commands
- LDAP server-specific utilities

The tool to manage security data depends on the type of data stored and the kind of store used to keep that data.

Users and Groups

If a domain uses the DefaultAuthenticator to store identities, then use the Oracle WebLogic Server Administration Console to manage the stored data. The data stored in the DefaultAuthenticator can be accessed by the User and Role API to query user profile attributes or to insert additional attributes to users or groups.

Important: If your domain uses the DefaultAuthenticator, then the domain administration server *must* be running for an application to operate on identity data using the User and Role API.

Otherwise, if authentication uses any other LDAP server different from the default authenticator or a DB, then, to manage users and groups, use the services of that LDAP server.

Policies, Credentials, Keys, and Certificates

Policies, keys, and credentials must use the same kind of storage (file-, LDAP-, or DB-based); if LDAP-based, the same LDAP server (Oracle Internet Directory only); if DB-based, the same DB (Oracle DB).

To manage policies and credentials use Fusion Middleware Control as explained in Section 10.3, "Managing Policies with Fusion Middleware Control" and Section 11.3, "Managing Credentials with Fusion Middleware Control," or the WLST commands, as explained in Section 10.4, "Managing Application Policies with WLST commands" and Section 11.4, "Managing Credentials with WLST commands."

Alternatively, to manage policy data, use Oracle Entitlements Server as explained in *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

Keys and certificates are managed with Fusion Middleware Control and WLST. For details, see Chapter 12, "Managing Keys and Certificates with the Keystore Service".

The following list summarizes the tools used to manage security data:

- Identity data
  - Default Authenticator: use Administration Console
  - Other LDAP or DB stores: use utilities provided by the LDAP server or DB
- Policy and Credential data
  - File-based: use Fusion Middleware Control or WLST commands
  - LDAP-based: use Fusion Middleware Control, WLST commands, or Oracle Entitlements Server
- Keys and Certificates
  - Use WLST commands

Changes to policies, credentials, or keys do not require server restart; changes to the file `jps-config.xml` *do* require server restart.

Note: In general, domain configuration changes require the server to be restarted; however, changes to the domain data do not require the server to be restarted. An example of a domain configuration change is the reassociation of domain stores.

For details about the automatic migration of application policies and credentials to the domain stores when the application is deployed, see Section 9.6, "Migrating the OPSS Security Store."

4.3 Packaging Requirements

File-based application policies are defined in the file `jazn-data.xml`. The only supported way to package this file with an application is to place it in the directory META-INF of an EAR file.

File-based application credentials are defined in a file that must be named `cwallet.sso`. The only supported way to package this file with an application is to place it in the directory META-INF of an EAR file. For details, see Section 24.5, "Packaging a Java EE Application Manually."

For information about deployment on WebLogic, see Chapter 6, "Deploying Secure Applications."

Note: Oracle JDeveloper automatically packages the EAR file for a secured Oracle ADF application with all the required files (and with the appropriate security configurations), when the EAR file is produced within that environment.

4.4 Example Scenarios

The scenarios explained in this section describe the security features adopted by most Oracle ADF applications, Oracle WebCenter, and Web Services Manager Control.

They assume that the application employs a security scheme that has the following characteristics:

- Authentication: it uses the WebLogic Default Authenticator to store users and groups.
- Authorization: it uses fine-grained JAAS authorization supported by file-based policies and credentials packaged with the application and by policy and credential stores (file- or LDAP-based).

One of these security schemes is typically employed by applications, such as Oracle ADF or Oracle SOA applications, that require fine-grained JAAS authorization. The various security components in these cases are managed with the appropriate tool.

Based on these assumptions, the following scenarios are typical variations on the basic theme; note that the list of variations presented is not exhaustive.

Related Documentation

For details about configuring the Default Authenticator, see section Configure Authentication and Identity Assertion Providers in *Oracle WebLogic Server Administration Console Online Help*.

For details about configuring the OPSS security store, see Chapter 9, "Configuring the OPSS Security Store."

For details about managing policies, see Chapter 10, "Managing the Policy Store."

For details about managing credentials, see Chapter 11, "Managing the Credential Store."

For details about managing the Keystore Service, see Chapter 12, "Managing Keys and Certificates with the Keystore Service."

Common Scenario 1

This scenario describes a Java EE application during development.

Authentication: The application uses the Default Authenticator, typical in development environments.

Authorization: The policy and credential stores are file-based.

Variation: The application uses the WebLogic support for Oracle Application Server Single Sign-On and Java EE security.

For details about WebLogic support for SSO, see section Configuring Single Sign-On with Web Browsers and HTTP Clients in *Administering Security for Oracle WebLogic Server*.

Common Scenario 2

This scenario describes a Java EE application during development.

Authentication: The application uses the Default Authenticator, typical in development environments.

Authorization: The policy and credential stores are LDAP-based using the services of the same instance of an Oracle Internet Directory LDAP server.

Variation: JAAS is enabled and policies include permissions for the anonymous and the authenticated roles.

For details about configuring support for the anonymous and authenticated roles, see Section 2.3, "About the Authenticated Role," and Section 2.4, "About the Anonymous User and Role."

Common Scenario 3

This scenario describes a Java EE application during development.

Authentication: The application uses the Default Authenticator, typical in development environments.

Authorization: The policy and credential stores are LDAP-based using the services of the same instance of an Oracle Internet Directory LDAP server.

Variation: The application uses Java EE security, JAAS is enabled, and policies include permissions for the anonymous and the authenticated role. It also uses the Credential Store Framework (CSF) APIs to query, retrieve, and manage policies.

For details about configuring support for the anonymous and authenticated roles, see Section 2.3, "About the Authenticated Role," and Section 2.4, "About the Anonymous User and Role."

For details about CSF APIs, see Section 19.1, "About the Credential Store Framework API."

4.5 Other Scenarios

In the following scenarios the application uses an authenticator other than the DefaultAuthenticator (typically used in the application development phase) or some API to access security data.

Scenario 4

Authentication: The application uses an LDAP authenticator (other than the DefaultAuthenticator).

Authorization: the application uses an Oracle Internet Directory LDAP-based store.

Variation: The application uses the User and Role API to access user profiles in the DB and the Credential Store Framework (CSF) APIs to access credentials.

For details about CSF APIs, see Section 19.1, "About the Credential Store Framework API."

Scenario 5

Authentication: The application uses the Oracle Internet Directory LDAP authenticator, typical in test and production environments.

Authorization: The policy and credential stores are file-based and packaged with the application. These data is automatically mapped to domain security data at deployment.

Variation: Post-deployment, the policy and credential stores are reassociated to an LDAP-based store configured through one-way SSL transmission channel.

For details about automatic migration of application security data at deployment, see Section 9.6, "Migrating the OPSS Security Store."

For details about reassociation, see Section 9.5, "Reassociating the OPSS Security Store."

For details about SSL configuration and related topics, see the following:

- Section Configuring SSL in *Administering Security for Oracle WebLogic Server*.
- *Administering Oracle Fusion Middleware*.
- Section Set up SSL in *Oracle WebLogic Server Administration Console Online Help*.
- Section Using SSL Authentication in Java Clients in *Developing Applications with the WebLogic Security Service*.

Scenario 6

This scenario describes a Java SE application using OPPS APIs.

Authentication: The application uses the LoginService API.

Authorization: The application uses the method `CheckPermission`.

In addition, the application uses the User and Role API to query attributes into the domain authenticator, and the Credential Store Framework API to query the credential store.

4.6 FIPS Support in Oracle Platform Security Services

Oracle Fusion Middleware uses the following OPSS services (among others) at run-time:

- Keystore Service

  All data in keystore service is encrypted using the AES symmetric key, which supports FIPS.

- Credential Store Framework

  When credential encryption is enabled, all the credential store data is encrypted using the AES symmetric key.

If using Oracle Internet Directory as your credential store, the store requires explicitly enabling encryption. For details, see Section 11.2.

Note: FIPS is enabled in the entire Oracle Fusion Middleware stack. For details, see "FIPS-140 Support in Oracle Fusion Middleware" in *Administering Oracle Fusion Middleware*.

Part II

Basic OPSS Administration

This part describes basic OPSS administration features in the following chapters:

5 Security Administration

This chapter introduces the tools available to a security administrator and the typical tasks required to manage application security.

These topics are presented in the following sections:

- Choosing the Administration Tool According to Technology
- Basic Security Administration Tasks
- Typical Security Practices with Fusion Middleware Control
- Typical Security Practices with the Administration Console
- Typical Security Practices with Oracle Entitlements Server
- Typical Security Practices with WLST Commands

For advanced administrator tasks, see Appendix E, "Administration with Scripting and MBean Programming."

5.1 Choosing the Administration Tool According to Technology

The four basic tools available to a security administrator are Oracle Enterprise Manager Fusion Middleware Control, Oracle WebLogic Administration Console, Oracle Entitlements Server, and the Oracle WebLogic Scripting Tool (WLST). For further details on these and other tools, see chapter 3, Getting Started Managing Oracle Fusion Middleware in *Administering Oracle Fusion Middleware*.

The main criterion that determines the tool to use to administer application security is whether the application uses just container-managed security (Java EE application) or it includes Oracle ADF security (Oracle ADF application).

Oracle-specific applications, such as Oracle Application Development Framework (Oracle ADF) applications, Oracle Server-Oriented Architecture (SOA) applications, and Web Center applications, are deployed, secured, and maintained with Fusion Middleware Control and Oracle Entitlements Server.

Other applications, such as those developed by third parties, Java SE, and Java EE applications, are typically deployed, secured, and administered with Oracle WebLogic Administration Console or with WLST.

The recommended tool to develop Java applications is Oracle JDeveloper 11g. This tool helps the developer configure file-based identity, policy, and credential stores through specialized graphical editors. In particular, when developing Oracle ADF applications, the developer can run a wizard to configure security for web pages associated with Oracle ADF resources (such as Oracle ADF task flows and page definitions), and define security artifacts using a specialized, visual editor for the file `jazn-data.xml`.

For details about procedures and related topics, see the following sections in the Oracle JDeveloper online help documentation:

- Securing a Web Application Using Oracle ADF Security
- Securing a Web Application Using Java EE Security
- About Oracle ADF Security as an Alternative to Security Constraints
- About Securing Web Applications

For further details about Oracle ADF Security and its integration with Oracle JDeveloper, see Accessing the Oracle ADF Security Design Time Tools, in *Developing Fusion Web Applications with Oracle Application Development Framework*.

For further details about Oracle Entitlements Server, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

5.2 Basic Security Administration Tasks

Table 5–1 lists some basic security tasks and the tools used to execute them. Recall that the tool chosen to configure and manage application security depends on the type of the application: for Java EE applications, which use just container-managed security, use the Oracle WebLogic Administration Console; for Oracle ADF applications, which use OPSS authorization, use Fusion Middleware Control and Oracle Entitlements Server.

Manual settings without the aid of the tools listed below are not recommended. For information about using the Oracle WebLogic Administration Console, see the list of links following the table below. For details about Oracle Entitlements Server, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

Table 5–1 Basic Administrative Security Tasks and Tools

Task	Use Fusion Middleware Control Security Menu	Use Other Tool
Configure WebLogic Domains		WebLogic Admin Console
Configure WebLogic Security Realms		WebLogic Admin Console
Manage WebLogic Domain Authenticators		WebLogic Admin Console
Enable SSO for MS clients, Web Browsers, and HTTP clients.		WebLogic Admin Console
Manage Domain Administrative Accounts		WebLogic Admin Console
Configuring the identity store service		WebLogic Admin Console
Manage application users and groups	Users and Groups	
Manage Credentials for Oracle ADF Application	Credentials	
Enable anonymous role in Oracle ADF Application	Security Provider Configuration	
Enable authenticated role in Oracle ADF Application	Security Provider Configuration	

Table 5–1 (Cont.) Basic Administrative Security Tasks and Tools

Task	Use Fusion Middleware Control Security Menu	Use Other Tool
Enable JAAS in Oracle ADF Application	Security Provider Configuration	
Map application to enterprise groups for Oracle ADF Application	Application Roles or Application Policies	Oracle Entitlements Server
Manage systemwide policies for Oracle ADF Applications	System Policies	
Configure OPSS Properties	Security Provider Configuration	
Reassociate Policy and Credential Stores	Security Provider Configuration	

Details about using the Oracle WebLogic Administration Console for the tasks above are found in the following documents:

- For general use of the Administration Console, see Oracle WebLogic Server Administration Console Online Help.
- To configure WebLogic domains, see *Oracle Fusion Middleware Understanding Domain Configuration for Oracle WebLogic Server*.
- To configure WebLogic security realms, see section Creating and Configuring a New Security Realm: Main Steps in *Oracle Fusion Middleware Securing Oracle WebLogic Server*.
- To manage WebLogic domain authenticators, see chapter 5 in *Oracle Fusion Middleware Securing Oracle WebLogic Server*.
- To configure Oracle Application Server Single Sign-On with MS clients, see chapter 6 in *Administering Security for Oracle WebLogic Server*.
- To manage domain administrative accounts, see chapter 6 in *Oracle Fusion Middleware Securing Resources Using Roles and Policies for Oracle WebLogic Server*.
- For details about configuring an LDAP identity store, see Section 3.2.2, "WebLogic Authenticators."

Note: OPSS does not support automatic backup or recovery of server files. It is recommended that the server administrator periodically back up all server configuration files, as appropriate.

For details about backing up and recovering Oracle Fusion Middleware, see chapter 15, Introducing Backup and Recovery, in *Administering Oracle Fusion Middleware*.

5.2.1 Setting Up a Brand New Production Environment

A new production environment based on an existing environment can be set up in either of the following ways:

- Replicating an established environment using Oracle Cloning utilities. For details, see section 9.5, Cloning Oracle Fusion Middleware Entities, in *Administering Oracle Fusion Middleware*.

- Reinstalling software and configuring the environment, as it was done to set up the established environment.

5.3 Typical Security Practices with Fusion Middleware Control

Fusion Middleware Control is a Web-based tool that allows the administration of a network of applications from a single point. Fusion Middleware Control is used to deploy, configure, monitor, diagnose, and audit Oracle SOA applications, Oracle ADF applications, Oracle WebCenter, and other Oracle applications using OPSS. This section mentions only security-related operations.

Fusion Middleware Control provides several security-related administration tasks; using this tool, an administrator can:

- Post-installation and before deploying applications, reassociate the policy and credential stores. For details, see Section 9.5.1, "Reassociating with Fusion Middleware Control."
- Post-installation and before deploying applications, define OPSS properties. For details, see Section 9.7, "Configuring Services Providers with Fusion Middleware Control."
- At deploy time, configure the automatic migration of file-based application policies and credentials to LDAP-based domain policies and credentials.

  For details see:

  – Section 6.3, "Deploying Oracle ADF Applications to a Test Environment."

  – Section 9.6, "Migrating the OPSS Security Store."
- For each application after it is deployed:

  – Manage application policies. For details, see Section 10.2, "Managing the Policy Store."

  – Manage credentials. For details, see Chapter 11, "Managing the Credential Store."

  – Manage users and groups. For details, see *Oracle Fusion Middleware Administering Oracle WebLogic Server with Fusion Middleware Control*.

  – Specify the mapping from application roles to users, groups, and application roles. For details, see Section 10.3.2, "Managing Application Roles."
- For the domain, manage system policies. For details see Section 10.3.3, "Managing System Policies."
- For the domain, manage OPSS properties. For details see Section 9.7, "Configuring Services Providers with Fusion Middleware Control."

For a summary of security administrative tasks and the tools used to execute them, see Basic Security Administration Tasks.

For further details about other functions, see the Fusion Middleware Control online help documentation.

5.4 Typical Security Practices with the Administration Console

The Oracle WebLogic Administration Console is a Web-based tool that allows, among other functions, application deployment and redeployment, domain configuration, and monitoring of application status. This section mentions only security-related operations.

Typical tasks performed with the Oracle WebLogic Administration Console include the following:

- Starting and stopping Oracle WebLogic Servers. For details see section Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*.
- Configuring Oracle WebLogic Servers and Domains. For details see section Configuring Existing Domains in *Understanding the WebLogic Scripting Tool*.
- Deploying applications. For details, see *Deploying Applications to Oracle WebLogic Server*.
- Configuring fail over support. For details see section Failover and Replication in a Cluster in *Administering Clusters for Oracle WebLogic Server*.
- Configuring WebLogic domains and WebLogic realms.
- Managing users and groups in domain authenticators.
- Enabling the use of Single Sign-On for MS clients, Web browsers, and HTTP clients.
- Managing administrative users and administrative policies.

For details about Oracle WebLogic Administration Console, see *Oracle WebLogic Server Administration Console Online Help*.

5.5 Typical Security Practices with Oracle Entitlements Server

Typical security tasks performed with Oracle Entitlements Server include the following:

- Searching application security artifacts.
- Managing application security artifacts, including policies.
- Viewing the external role hierarchy.
- Managing the application role hierarchy.

For a list of some of the most frequent security tasks to administer application security with Oracle Entitlements Server, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

5.6 Typical Security Practices with WLST Commands

Most of the security-related operations available in the Oracle WebLogic Administration Console can be carried out with WLST commands, a set of command-line interface that allows the scripting and automation of administration tasks, including domain configuration and application deployment.

Note: A WLST shell session is associated with a unique configuration file; more generally, a JVM instance can point to at most one configuration file.

This requirement of a unique `jps-config.xml` file per JVM instance has the following consequence: suppose that within a WLST shell you invoke a WLST command, such as `migrateSecurityStore`, that takes a configuration file; then all subsequent commands invoked *within that same WLST shell* use the same configuration file regardless of the configuration location passed to a command.

6 Deploying Secure Applications

An application can be deployed to an Oracle WebLogic Server using any of the following tools: the Oracle WebLogic Server Administration Console, Oracle Enterprise Manager Fusion Middleware Control, or Oracle JDeveloper. An application can also be started by setting its bits in a location known to the WebLogic server, without the need to restart the server; this kind of application start is known as *hot deployment*.

The recommended way to deploy an application depends on the platform, the application type, and whether the application is in the developing phase or in a post-development phase. For example, in the post-development phase, typically, the appliction is started in a production environment by means of a hot deployment.

The recommendations stated in this chapter apply to Oracle ADF applications and to Java EE applications using OPSS.

During development, the application is typically deployed with Oracle JDeveloper to the embedded Oracle WebLogic Server. Once the application transitions to test or production environments, it is typically deployed with Fusion Middleware Control or the Oracle WebLogic Server Administration Console or by a hot deployment.

This chapter focuses on administrative tasks performed at deployment of an Oracle ADF or pure Java EE application. The last section explains the packaging requirements to secure Java EE applications, a topic relevant only when the application is packaged manually.

These topics are presented in the following sections:

Additional Documentation

For further details about deployment, see Chapter 8, Deploying Applications, in *Administering Oracle Fusion Middleware*.

For details about securing an Oracle ADF application during development, see *Developing Fusion Web Applications with Oracle Application Development Framework.*

For details about the application lifecycle, see Section 16.7, "Securing an ADF Application."

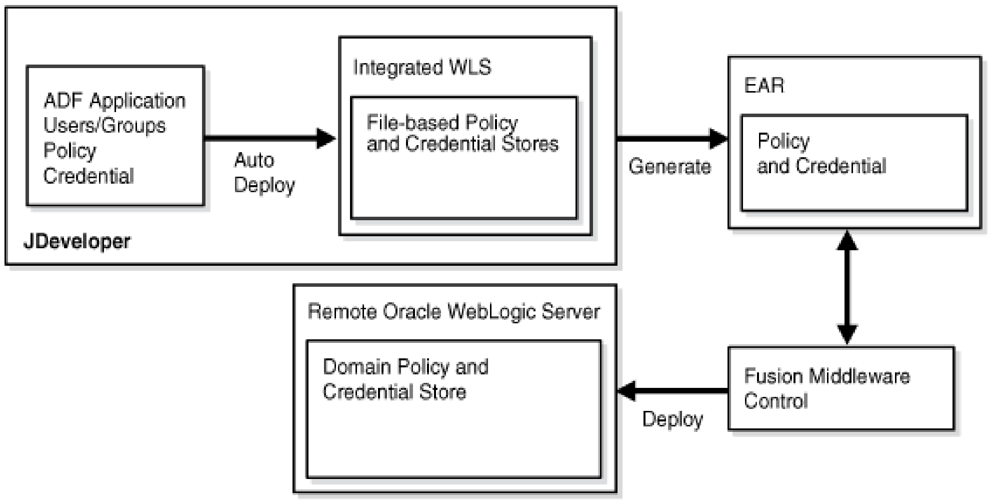
For details about the files in an EAR file relevant to application security management and configuration, such as `web.xml` and `weblogic-application.xml`, see Chapter 24, "Configuring Java EE Applications to Use OPSS."

6.1 Overview

The steps that lead to the deployment of an Oracle ADF application into a remote Oracle WebLogic Server are, typically, as follows:

- Using Oracle JDeveloper, a developer develops an Oracle ADF application into which Oracle ADF security is included with the Oracle ADF Security Wizard.
- Application users and groups, authorization policies, and credentials are copied by Oracle JDeveloper to the integrated WebLogic Server, into which the application is auto-deployed during the test cycles in that environment.
- The developer creates an application EAR file which packs policies and credentials.
- The domain administrator deploys the EAR file to a remote Oracle WebLogic Server using Fusion Middleware Control.

This flow is illustrated in the following graphic:


ADF Application
Users/Groups
Policy
Credential
Auto Deploy
Integrated WLS
File-based Policy and Credential Stores
JDeveloper
Generate
EAR
Policy and Credential
Remote Oracle WebLogic Server
Domain Policy and Credential Store
Deploy
Fusion Middleware Control

6.2 Selecting the Tool for Deployment

The types of application we consider in this chapter are Java EE applications, which are further categorized into pure Java EE applications and Oracle Fusion Middleware ADF applications. The distinction of these two kinds of Java EE applications is explained in sections Section 1.5.1, "Scenario 1: Enhancing Security in a Java EE Application," and Section 1.5.2, "Scenario 2: Securing an Oracle ADF Application."

Table 6–1 lists the tool used to deploy a developed application according to its type.

Table 6–1 Tools to Deploy Applications after Development

Application Type	Tool to Use
Pure Java EE Application	Oracle WebLogic Administration Console, Fusion Middleware Control. The recommended tool is Oracle WebLogic Administration Console.

Table 6–1 (Cont.) Tools to Deploy Applications after Development

Application Type	Tool to Use
Oracle ADF Application	Fusion Middleware Control or WLST commands. The recommended tool is Fusion Middleware Control.

6.2.1 Deploying Java EE and Oracle ADF Applications with Fusion Middleware Control

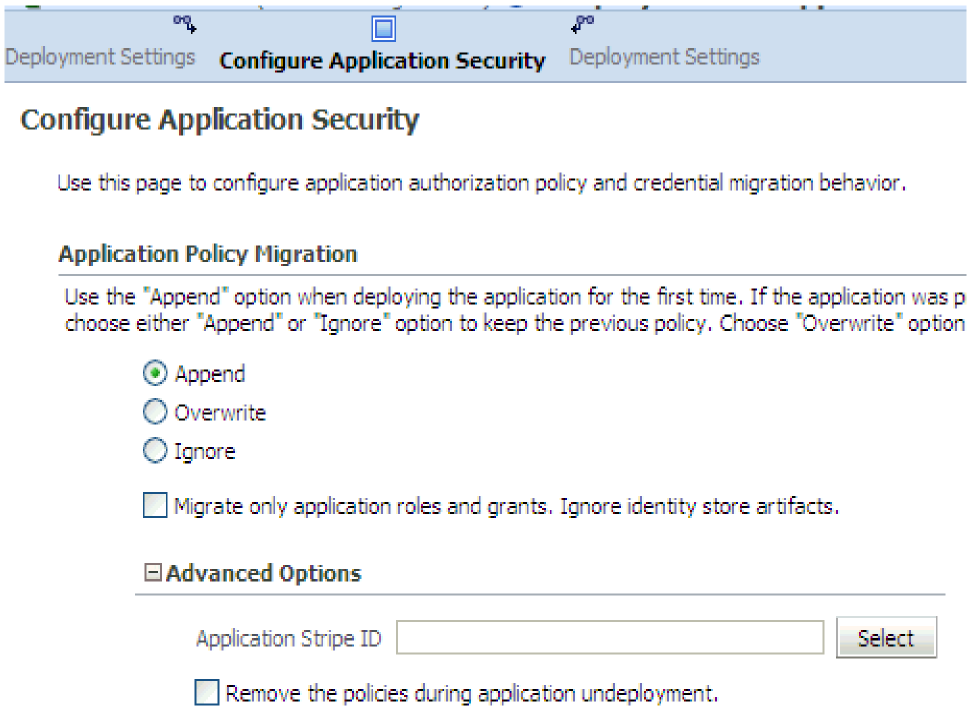
This section focuses on the security configurations available when deploying an application that uses Oracle ADF security or a Java EE application that uses OPSS with Fusion Middleware Control on the WebLogic server.

Specifically, it describes the options you find in the page **Configure Application Security** at the third stage of the deploy settings.

The appearance of this page varies according to what is packaged in the EAR fie, as follows:

- If the EAR file packages `jazn-data.xml` with application policies, the application policy migration section is shown.
- If the EAR file packages credentials in `cwallet.sso`, the credential migration section is shown.
- If the EAR file does not include any of the above, then the page displays the default Java EE security options.

This page, showing the policy migration sections, is partially illustrated in the following graphic:



Configure Application Security

Use this page to configure application authorization policy and credential migration behavior.

Application Policy Migration

Use the "Append" option when deploying the application for the first time. If the application was p
choose either "Append" or "Ignore" option to keep the previous policy. Choose "Overwrite" option

[✓ Append]
Overwrite
Ignore

☐ Migrate only application roles and grants. Ignore identity store artifacts.

Advanced Options

Application Stripe ID

☐ Remove the policies during application undeployment.

The settings in this page concern the migration of application policies and credentials (packed in application EAR file) to the corresponding domain store, and they are explained next.

Application Policy Migration Settings

These settings control of the policy migration in the following scenarios:

- If you are deploying the application for the first time, you typically want application policies to be migrated to the OPSS security store. Therefore, select **Append** in the **Application Policy Migration** area.

  If for some reason you do not want the migration to take place, select instead **Ignore**. The option **Overwrite** is also supported.

- If you are redeploying the application, and assuming that the migration of application policies has taken place in a previous deployment, you can choose **Append**, to merge the packed policies with the existing ones in the domain, or **Ignore**, to prevent policy migration.

  The option **Ignore** is typically selected when an application is redeployed and you want to leave the current application policies in the domain unchanged, that is, when you want to preserve changes made to the OPSS security store during previous deployments.

- When you choose **Append**, you can further specify which grants and roles should be migrated; the basic distinction is between ADF application roles and grants (needed in a production environment), and development-time only roles and grants (not needed in a production environment).

  To migrate ADF application roles and grants, and not to migrate development-time only security roles and grants, check the box **Migrate only application roles and grants. Ignore identity store artifacts**. Typically, this box is checked when deploying to a production environment. Note that when this box is checked, you will need to map application roles to enterprise groups once the application has been deployed.

- When you choose **Append**, you can further specify a particular stripe (different from the default stripe, which is the application name) into which the application policies should be migrated, by entering the name of that stripe in the box **Application Stripe Id**.

About Application Stripes: The OPSS security store is logically partitioned in stripes, one for each application name specified in the file `system-jazn-data.xml` under the element <applications>. Each stripe identifies the subset of domain policies that pertains to a particular application.

Typical Use Cases: This page supports specifying the migration of policies in the following two most common scenarios:

- Resolving inconsistent specifications found in the EAR file - The specifications in the EAR file are validated; if specifications regarding the application stripe found in the files `web.application.xml`, `web.xml`, and `ejb-jar.xml` (packed in the EAR file) are inconsistent (that is, do not match), you can enter a new stripe to use or select one from the drop-down list. The specified value trumps any other specified value in the EAR file and it is used as the target of the migration and in the runtime environment.
- Allowing two or more applications to share an application stripe - If your application is to share an existing stripe (populated originally by some other application), you can specify that stripe. The **Overwrite** option should be used carefully when sharing an existing application stripe.

- I f nothing is specified, the default settings are **Append** (in deployment) and **Ignore** (in redeployment).

Application Credential Migration Settings

These settings control of the credential migration in the following scenarios:

- If you are deploying the application for the first time, you typically want application credentials to be migrated to the credential store. Therefore, select **Append** in the **Application Credential Migration** area.
- In any case (first or succeeding deployment), if for some reason you do not want the migration to take place, select instead **Ignore**.

Note: Application code using credentials may not work if the credential migration is ignored. Typically, one would choose the **Ignore** option under the assumption that the credentials are manually created with the same map and key, but with different values.

- The option **Overwrite** is supported *only* when the WebLogic server is running in development mode.
- If nothing is entered, the default is **Ignore**.

6.3 Deploying Oracle ADF Applications to a Test Environment

An Oracle ADF application is a Java EE application using JAAS authorization, and it is typically developed and tested using Oracle JDeveloper; this environment allows a developer to package the application and deploy it in the Embedded Oracle WebLogic Server integrated with the tool. When transitioning to a test or production environment, the application is deployed using Oracle Fusion Middleware Control to leverage all the Oracle ADF security features that the framework offers. For details, see Overview.

For step-by-step instructions on how to deploy an Oracle ADF application with Fusion Middleware Control, see:

- Section Deploy an Application Using Fusion Middleware Control in the Oracle Fusion Middleware Control online help system.
- Section 8.4, Deploying and Undeploying Oracle ADF Applications, in *Administering Oracle Fusion Middleware*.

This section is divided into the following topics:

- Deploying to a Test Environment
- Migrating from a Test to a Production Environment

6.3.1 Deploying to a Test Environment

The security options available at deployment are explained in Deploying Java EE and Oracle ADF Applications with Fusion Middleware Control.

When deploying an Oracle ADF application to a test environment with Fusion Middleware Control, the following operations take place:

Policy Management

- Application-specific policies packed with the application are automatically migrated to the OPSS security store when the application is deployed.

  Oracle JDeveloper automatically writes the necessary configuration for this migration to occur.

> **Note:** Before migrating a file-based OPSS security store (that is, the file `jazn-data.xml`) to a production environment, verify that any grant contains no duplicate permissions. If a duplicate permission (one that has the same name and class) appears in a grant, the migration runs into an error and it is halted. In this case, edit the `jazn-data.xml` file to remove any duplicate permissions from a grant definition, and then invoke the migration again.

Credential Management

- Application-specific credentials packed with the application are automatically migrated to the credential store when the application is deployed.

  Oracle JDeveloper automatically writes the necessary configuration for this migration to occur.
- The bootstrap credentials necessary to access LDAP repositories during migration are automatically produced by Fusion Middleware Control.

Identity Management

Identities packed with the application are not migrated. The domain administrator must configure the domain authenticator (with the Administration Console), update identities (enterprise users and groups) in the environment, as appropriate, and map application roles to enterprise users and groups (with Fusion Middleware Control).

Other Considerations

- When deploying to a domain with LDAP-based security stores and to preserve application data integrity, it is recommended that the application be deployed at the cluster level or, otherwise, to just one managed server.

- When deploying an application to multiple managed servers, be sure to include the administration server so that data is migrated as expected.
- The reassociation of domain stores is an infrequent operation and, typically, takes place when the domain is set up before applications are deployed. For procedure details, see Section 9.5.1, "Reassociating with Fusion Middleware Control."

6.3.1.1 Typical Administrative Tasks after Deployment in a Test Environment

At any time after an application is deployed ondeployed on a test environment, an administrator can perform the following tasks using Fusion Middleware Control or the Administration Console:

- Map application roles to enterprise groups. Until this mapping is accomplished, security does not work as expected. For procedure details, see Section 10.3.2, "Managing Application Roles."
- Create additional application roles or customize existing ones. For details, see Section 10.3.2, "Managing Application Roles."
- Manage system policies. For procedure details, see Section 10.3.3, "Managing System Policies."
- Manage credentials. For procedure details, see Chapter 11, "Managing the Credential Store."

Notes: If the application is undeployed with Fusion Middleware Control from a server running in production mode, then the application-specific policies are automatically removed from the OPSS security store. Otherwise, if you use any other tool to undeploy the application, then the application-specific policies must be removed manually.

Credentials are not deleted upon an application undeployment. A credential may have started it life as being packaged with an application, but when the application is undeployed credentials are *not* removed.

6.4 Deploying Standard Java EE Applications

There are two ways to secure Java EE applications that do not use OPSS but that use standard Java authorization: administratively, with the Administration Console or a WLST command; or programmatically, with deployment descriptors.

A Java EE application deployed to the Oracle WebLogic Server *is* a WebLogic resource. Therefore, an administrator would set security for the deployed application the same way that he would for any other resource.

For details about deployment procedures, see section 8.3, Deploying and Undeploying Java EE Applications, in *Administering Oracle Fusion Middleware*.

For an overview of WebLogic Server deployment features, see chapter Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*.

Related Documentation

Further information about securing application resources, can be found in the following documents:

In *Securing Resources Using Roles and Policies for Oracle WebLogic Server*

- Section Application Resources
- Section Options for Securing Web Application and EJB Resources

In *Oracle WebLogic Server Administration Console Online Help*:

- Section Use Roles and Policies to Secure Resources

In *Securing WebLogic Web Services for Oracle WebLogic Server*:

- Section Overview of Web Services Security

In *Developing Applications with the WebLogic Security Service*:

- Section Securing Web Applications. Particularly relevant is the subsection Using Declarative Security with Web Applications
- Section Securing Enterprise JavaBeans (EJBs)
- Section Using Java Security to Protect WebLogic Resources

6.5 Deploying Applications with Auditing

Applications can use the Oracle Fusion Middleware Audit Framework to provide auditing capabilities. OPSS provides an audit service that enables you to integrate with the audit framework.

When you deploy the application, you must register it with the audit service to leverage audit features.

For general background about the audit framework, see Chapter 13.

For complete details about how to integrate your application with the audit service, see Section 23.2.

6.5.1 Packaging Requirements for Auditing

You must package certain configuration files with the application EAR file:

- an event definitions file describing the auditable events for the application
- translation files containing localizable elements

For details, see Section 23.3 and Section 23.4.

6.5.2 Registration with the Audit Service

Along with the packaged files mentioned in Section 6.5.1, you also configure certain audit registration parameters in the OPSS deployment descriptor file, which is also packaged with the EAR. The files are processed automatically by the audit registration service when the application is deployed.

For details, see Section 23.4.

6.5.3 Migrating Audit Data

For details on this topic, see Section 6.6.3.

6.6 Migrating from a Test to a Production Environment

The recommendations that follow apply only to Java EE applications using JAAS authorization, such as Oracle Application Development Framework, Oracle SOA, and Oracle WebCenter applications, and they do not apply to Java EE applications using standard authorization. For deploying the latter, see Deploying Standard Java EE Applications.

The recommended tool to deploy applications is Fusion Middleware Control, and the user performing the operations described in the following sections must have the appropriate privileges, including the privilege to seed a schema in an LDAP repository.

It is assumed that a production environment has been set up as explained in Section 5.2.1, "Setting Up a Brand New Production Environment."

Important Note: File-based stores are not recommended in production environments.

The migration to a new production environment is divided into the following phases:

- Migrating Identities
- Migrating Policies and Credentials
- Migrating Audit Information
- Migrating Keystore Service Artifacts

Migration can also be used to backup and recover security data: to backup security data, migrate to an XML-based store; to recover security data, migrate from a saved XML-based store to the target security store.

6.6.1 Migrating Identities

The configuration of providers (other than policy and credential providers) in the production environment must be repeated as it was done in the test environment. This task may include:

- The identity store configuration, including the provisioning of required users and groups using the WebLogic Administrator Console or the WLST command `migrateSecurityStore`. For details about this last command, see Migrating Identities with migrateSecurityStore.
- Any particular provider configuration that you have performed in the test environment.

Note: Oracle WebLogic Server provides several tools to facilitate the creation of domains, such as the `pack` and `unpack` commands. For details, see *Oracle Fusion Middleware Creating Templates and Domains Using the Pack and Unpack Commands*.

6.6.1.1 Migrating Identities with migrateSecurityStore

Identity data can be migrated from a source repository to a target repository using the WLST command `migrateSecurityStore`. This migration is needed, for example, when transitioning from a test environment that uses a file-based identity store to a production environment that uses an LDAP-based identity store.

This script is **offline**, that is, it does not require a connection to a running server to operate; therefore, the configuration file passed to the argument `configFile` need not be an actual domain configuration file, but it can be assembled *just* to specify the source and destination repositories of the migration.

This script can be run in interactive mode or in script mode. In interactive mode, you enter the script at a command-line prompt and view the response immediately after. In script mode, you write scripts in a text file (with a py file name extension) and run it without requiring input, much like the directives in a shell script.

For platform-specific requirements to run an WLST command, see note in Section 10.4, "Managing Application Policies with WLST commands."

Script and Interactive Modes Syntaxes

To migrate identities on WebLogic, use the script (first) or interactive (second) syntaxes (arguments are written in separate lines for clarity):

```
migrateSecurityStore -type idStore
                     -configFile jpsConfigFileLocation
                     -src srcJpsContext
                     -dst dstJpsContext
                     [-dstLdifFile LdifFileLocation]
                     [-overwrite trueOrfalse]

migrateSecurityStore(type="idStore", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext", [dstLdifFile="LdifFileLocation"]
[,overwrite="trueOrfalse"])
```

The meaning of the arguments (all required except `dstLdifFile`) is as follows:

- `configFile` specifies the relative or absolute path of a configuration file `jps-config.xml`.
- `src` specifies the name of a jps-context in the configuration file passed to the argument `configFile`, where the source store is specified. The WLS Embedded LDAP cannot be the source store of a migration with this command.
- `dst` specifies the name of another jps-context in the configuration file passed to the argument `configFile`, where the destination store is specified. The destination store must be an LDAP-based identity store. For list of supported types, see Section 3.2.1, "Identity Store Types and WebLogic Authenticators."
- `dstLdifFile` specifies the relative or absolute path to the LDIF file created. Applies only when the destination is an LDAP-based Oracle Internet Directory store, such as the embedded LDAP. Notice that the LDIF file is not imported into the LDAP server and, typically, requires manual editing.
- `overwrite` specifies whether to overwrite existing data in the destination store; set to true to overwrite existing destination data; set to false to not overwrite existing destination data. Optional; default = false.

The contexts passed to `src` and `dst` must be defined in the passed configuration file and must have distinct names. From these two contexts, the script determines the locations of the source and the target repositories involved in the migration.

After an LDIF file is generated, the next step typically involves manual editing this file to customize the attributes of the LDAP repository where the LDIF file would, eventually, be imported.

Important: The password of every user in the output LDIF file is not the real user password, but the fake string *weblogic*. In case the destination is an LDAP-based Oracle Internet Directory store, the fake string is *change*.

Before importing the LDIF file into the target LDAP store, the security administrator would typically edit this file and change the fake passwords for real ones.

6.6.2 Migrating Policies and Credentials

In a production environment, it is strongly recommended that the OPSS security store (policy, credential, and key stores) be reassociated to an LDAP-based Oracle Internet Directory; if the test policy and credential stores were also LDAP, the production LDAP is assumed to be distinct from the test LDAP; if the test OPSS security store was file-based, verify that no grant has duplicate permissions; see note in Policy Management.

The migration of policies, credentials, and keys can take place automatically, when an application is deployed; or with the WLST command `migrateSecurityStore`, before or after the application is deployed; for details see the following sections:

- Migrating Policies with migrateSecurityStore
- Migrating Credentials with migrateSecurityStore

Important Note: If the application is hot deployed, that is without stoping and restarting the server, the migration of data in the file `jazn-data.xml` to the domain security store is carried out *provided* the security store does not contain a stripe with the same name as the application. In particular, if the application is hot re-deployed (that is, hot deployed for a second or later time), any changes introduced in the file `jazn-data.xml` are *not* migrated over the domain security store.

To disable the automatic migration of policies and credentials for *all* applications deployed on a WebLogic Server (regardless of the application migration particular settings), set the system property `jps.deployment.handler.disabled` to TRUE.

When deploying an application to a production environment, an administrator should know the answer to the following question:

Have policies or credentials packed in the application EAR been modified in the test environment?

Assuming that you know the answer to the above question, to deploy an application to a production environment, proceed as follows:

1. Use Fusion Middleware Control to deploy the application EAR file to the production environment using the following options:
   - If policies (application or system) have been modified in the test environment, then disable the option to migrate policies at deploy time by selecting the option **Ignore** under the **Application Policy Migration** area in Fusion Middleware Control's page **Configuration Application Security**; otherwise, select **Append**.

Note: You can select **Append** (that is, to migrate application policies) *in combination with* checking the box **Migrate only application roles and grants. Ignore identity store artifacts**, even when application roles have been modified in the test environment to the extent of mapping them to test enterprise groups.

Selecting this combination migrates application policies but disregards the maps to test enterprise groups. Later on, in step 3 below, you must remap application roles to production enterprise groups.

- If credentials have been modified in the test environment, then disable the option to migrate credentials at deploy time by selecting the option **Ignore** under the **Application Credential Migration** area in Fusion Middleware Control's page **Configuration Application Security**; otherwise, select **Append**.

2. Use the script `migrateSecurityStore` to migrate modified data, as follows:

   - If you chose to **Ignore** application policy migration, then migrate application and system policies from the test to the production LDAP. See example in Migrating Policies with migrateSecurityStore.

   - If you chose to **Ignore** application credential migration, then migrate credentials from the test to the production LDAP. See example in Migrating Credentials with migrateSecurityStore.

3. In any case, use Fusion Middleware Control to map application roles to production enterprise groups, as appropriate.

4. Use Fusion Middleware Control to verify that administrative credentials in the production environment are valid; in particular, test passwords versus production passwords; if necessary, modify the production data, as appropriate.

Note: There is a way to configure the application so that, at deployment, the migration of policies preserves GUIDs (instead of recreating them).

This setting can only be configured manually. For details, see parameter `jps.approle.preserveguid` in Section 24.6.1, "Controlling Policy Migration."

6.6.2.1 Migrating Policies with migrateSecurityStore

By default, the script `migrateSecurityStore` recreates GUIDs and may take a long time to migrate large volume of policies; for these reasons, during the transition from a test to a production environment, you may want to consider migrating policies and credentials with an alternate procedure that uses Oracle Internet Directory bulk operations. For details, see Section 7.6.2, "Backing Up and Recovering an OID-Based Security Store.".

Migrating policies manually with the script `migrateSecurityStore` requires assembling a configuration file where the source and destination are specified.

Here is a complete sample of a configuration file, named `t2p-policies.xml`, illustrating the specification of policy sources in LDAP, DB, and XML storages, and of policy destinations in LDAP and DB storages:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
```

```
<jpsConfig xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_
1.xsd" schema-major-version="11" schema-minor-version="1">

<serviceProviders>
  <serviceProvider
class="oracle.security.jps.internal.policystore.xml.XmlPolicyStoreProvider"
name="policystore.xml.provider" type="POLICY_STORE">
 <description>XML-based policy store provider</description>
 </serviceProvider>

 <serviceProvider
class="oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"
name="ldap.policystore.provider" type="POLICY_STORE">
 <property value="OID" name="policystore.type"/>
 <description>LDAP-based policy store provider</description>
 </serviceProvider>

 <serviceProvider
class="oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"
name="db.policystore.provider" type="POLICY_STORE">
 <property value="DB_ORACLE" name="policystore.type"/>
 <description>DB-based policy store provider</description>
 </serviceProvider>
</serviceProviders>

<serviceInstances>
 <!-- Source XML-based policy store instance -->
 <serviceInstance location="./system-jazn-data.xml"
provider="policystore.xml.provider" name="policystore.xml.source">
 <description>Replace location with the full path of the folder where the
system-jazn-data.xml is located in the source file system </description>
 </serviceInstance>

<!-- Source LDAP-based policy store instance -->
<serviceInstance provider="ldap.policystore.provider"
name="policystore.ldap.source">
 <description>Replace: A. mySourceDomain and mySourceRootName to appropriate
   values according to your source LDAP directory structure; B. OID with OVD,
   if your source LDAP is OVD; C. ldap://mySourceHost.com:3060 with the URL
   and port number of your source LDAP</description>
 <property value="OID" name="policystore.type"/>
 <property value="bootstrap" name="bootstrap.security.principal.key"/>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://mySourceHost.com:3060" name="ldap.url"/>
</serviceInstance>

<!-- Source DB-based policy store instance -->
<serviceInstance provider="db.policystore.provider" name="policystore.db.source">
 <description>Replace: mySourceDomain and mySourceRootName to appropriate
   values according to your source DB policy store structure
 </description>
 <property value="DB_ORACLE" name="policystore.type"/>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@mySourceHost.com:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the source
      datasource was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="mySourceKeyName" />
 <property name="bootstrap.security.principal.map" value="mySourceMapName" />
 <!-- the values of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the values entered when the bootstrap credential was set up -->
</serviceInstance>

 <!-- Destination LDAP-based policy store instance -->
 <serviceInstance provider="ldap.policystore.provider"
name="policystore.ldap.destination">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your destination LDAP directory structure; B. OID with OVD, if your
destination LDAP is OVD; C. ldap://myDestHost.com:3060 with the URL and port
number of your destination LDAP</description>
 <property value="OID" name="policystore.type"/>
 <property value="bootstrap" name="bootstrap.security.principal.key"/>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://myDestHost.com:3060" name="ldap.url"/>
</serviceInstance>

<!-- Destination DB-based policy store instance -->
 <serviceInstance provider="db.policystore.provider"
name="policystore.db.destination">
<description>Replace: myDestDomain and myDestRootName to appropriate values
 according to your destination DB policy store structure</description>
 <property value="DB_ORACLE" name="policystore.type"/>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@myDestHostcom:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the destination
datasource was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="myDestKeyName" />
 <property name="bootstrap.security.principal.map" value="myDestMapName" />
 <!-- the value of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the value entered when the bootstrap credential was set up -->
</serviceInstance>

<!-- Bootstrap credentials to access source and destination LDAPs or DBs-->
 <serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.credstore">
  <description>Replace location with the full path of the directory where the
bootstrap file cwallet.sso is located; typically found in
destinationDomain/config/fmwconfig/</description>
 </serviceInstance>
 </serviceInstances>

 <jpsContexts>
 <jpsContext name="XMLsourceContext">
 <serviceInstanceRef ref="policystore.xml.source"/>
 </jpsContext>

 <jpsContext name="LDAPsourceContext">
 <serviceInstanceRef ref="policystore.ldap.source"/>
 </jpsContext>

<jpsContext name="DBsourceContext">
```

```
 <serviceInstanceRef ref="policystore.db.source"/>
 </jpsContext>

 <jpsContext name="LDAPdestinationContext">
 <serviceInstanceRef ref="policystore.ldap.destination"/>
 </jpsContext>

<jpsContext name="DBdestinationContext">
 <serviceInstanceRef ref="policystore.db.destination"/>
 </jpsContext>

 <!-- Do not change the name of the next context -->
 <jpsContext name="bootstrap_credstore_context">
 <serviceInstanceRef ref="bootstrap.credstore"/>
 </jpsContext>
 </jpsContexts>
</jpsConfig>
```

Note that since the migration involves LDAP and DB stores, the file includes a jps-context named `bootstrap_credstore_context` that specifies the directory where the bootstrap credential file `cwallet.sso` is located. Furthermore, for each pair of map name and key name in the sample above, you must provide the corresponding bootstrap credentials using the WLST command `addBootStrapCredential` as illustrated in the following example:

```
wls:/offline> addBootStrapCredential(jpsConfigFile='jps-config.xml',
 map='myMapName', key='myKeyName', username='myUserName',
 password='myPassword')
```

where `myUserName` and `myPassaword` specify the user account name and password to access the target database.

The following examples of use of `migrateSecurityStore` assume that:

- The file `t2p-policies.xml` is located on the target system in the directory where the script is run.
- The directory structure of LDAP or DB system policies in the test and production environments should be *identical*. If this is not the case, before using the script, restructure manually the system policy directory in the production environment to match the corresponding structure in the test environment.

Under these assumptions, to migrate policies from a test (or source) LDAP store to a production (or destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="policyStore",configFile="t2p-policies.xml",src="LDAPso
urceContext",dst="LDAPdestinationContext")
```

To migrate policies from a test (or source) XML store to a production (or destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="policyStore",configFile="t2p-policies.xml",src="XMLsou
rceContext",dst="LDAPdestinationContext")
```

To migrate policies from a test (or source) DB store to a production (or destination) DB store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="policyStore",configFile="t2p-policies.xml",src="DBsour
ceContext",dst="DBdestinationContext")
```

6.6.2.2 Migrating Credentials with migrateSecurityStore

The script `migrateSecurityStore` recreates GUIDs and may take a long time to migrate large volume of credentials; for these reasons, during the transition from a test to a production environment, you may want to consider migrating policies and credentials with an alternate procedure that uses Oracle Internet Directory bulk operations. For details, see Section 7.6.2, "Backing Up and Recovering an OID-Based Security Store.".

Migrating credentials manually with `migrateSecurityStore` requires assembling a configuration file where the source and destination are specified.

Since `migrateSecurityStore` recreates GUIDs and takes a long time to migrate large volume of data, you may want to consider migrating stores with an alternate procedure that uses Oracle Internet Directory bulk operations. For details, see Section 7.6.2, "Backing Up and Recovering an OID-Based Security Store.".

Here is a complete sample of a configuration file, named `t2p-credentials.xml`, illustrating the specification of credential sources in LDAP, DB, and XML storages, and of credential destinations in LDAP or DB storages:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jpsConfig xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_
1.xsd" schema-major-version="11" schema-minor-version="1">

<serviceProviders>
 <serviceProvider
class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
name="credstoressp" type="CREDENTIAL_STORE">
 <description>File-based credential provider</description>
 </serviceProvider>

 <serviceProvider
class="oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider"
name="ldap.credentialstore.provider" type="CREDENTIAL_STORE">
 <description>LDAP-based credential provider</description>
 </serviceProvider>

<serviceProvider
class="oracle.security.jps.internal.credstore.rdbms.DbmsCredentialStoreProvider"
name="db.credentialstore.provider" type="CREDENTIAL_STORE">
 <description>DB-based credential provider</description>
 </serviceProvider>

</serviceProviders>

<serviceInstances>
 <!-- Source file-based credential store instance -->
 <serviceInstance location="myFileBasedCredStoreLocation" provider="credstoressp"
name="credential.file.source">
 <description>Replace location with the full path of the folder where the
file-based source credential store cwallet.sso is located in the source file
system; typically located in sourceDomain/config/fmwconfig/
</description>
 </serviceInstance>

<!-- Source LDAP-based credential store instance -->
<serviceInstance provider="ldap.credentialstore.provider"
name="credential.ldap.source">
```

```
 <description>Replace: A. mySourceDomain and mySourceRootName to appropriate
 values according to your source LDAP directory structure; B.
ldap://mySourceHost.com:3060 with the URL and port number of your source
LDAP</description>
 <property value="bootstrap" name="bootstrap.security.credential.key"/>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://mySourceHost.com:3060" name="ldap.url"/>
</serviceInstance>


<!-- Source DB-based credential store instance -->
<serviceInstance provider="db.credentialstore.provider"
name="credential.db.source">
 <description>Replace: A. mySourceDomain and mySourceRootName to appropriate
 values according to your source DB credential store</description>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@mySourceHost:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the source datasource
was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="mySourceKeyName" />
 <property name="bootstrap.security.principal.map" value="mySourceMapName" />
 <!-- the values of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the values entered when the bootstrap credential was set up -->
</serviceInstance>

 <!-- Destination LDAP-based credential store instance -->
 <serviceInstance provider="ldap.credentialstore.provider"
name="credential.ldap.destination">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your destination LDAP directory structure; B.
ldap://myDestHost.com:3060 with the URL and port number of your destination
LDAP</description>
 <property value="bootstrap" name="bootstrap.security.credential.key"/>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://myDestHost.com:3060" name="ldap.url"/>
</serviceInstance>

<!-- Destination DB-based credential store instance -->
 <serviceInstance provider="db.credentialstore.provider"
name="credential.db.destination">
<description>Replace: myDestDomain and myDestRootName to appropriate values
according to your destination DB credential store</description>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@myDestHost.com:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the destination
datasource was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="myDestKeyName" />
 <property name="bootstrap.security.principal.map" value="myDestMapName" />
 <!-- the values of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the values entered when the bootstrap credential was set up -->
</serviceInstance>
```

```
<!-- **Bootstrap credentials to access source and destination LDAPs and DBs** -->
 <serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.credstore">
 <description>Replace location with the full path of the directory where the
bootstrap file cwallet.sso is located; typically found in
destinationDomain/config/fmwconfig/</description>
 </serviceInstance>
 </serviceInstances>

 <jpsContexts>
 <jpsContext name="FileSourceContext">
 <serviceInstanceRef ref="credential.file.source"/>
 </jpsContext>

 <jpsContext name="LDAPsourceContext">
 <serviceInstanceRef ref="credential.ldap.source"/>
 </jpsContext>

<jpsContext name="DBsourceContext">
 <serviceInstanceRef ref="credential.db.source"/>
 </jpsContext>

 <jpsContext name="LDAPdestinationContext">
 <serviceInstanceRef ref="credential.ldap.destination"/>
 </jpsContext>

<jpsContext name="DBdestinationContext">
 <serviceInstanceRef ref="credential.db.destination"/>
 </jpsContext>

 <!-- **Do not change the name of the next context** -->
 <jpsContext name="bootstrap_credstore_context">
 <serviceInstanceRef ref="bootstrap.credstore"/>
 </jpsContext>
 </jpsContexts>
</jpsConfig>
```

Note that since the migration involves LDAP and/or DB stores, the file includes a jps-context named `bootstrap_credstore_context` that specifies the directory where the bootstrap credential file `cwallet.sso` is located.

The following examples of use of `migrateSecurityStore` assume that the file `t2p-credentials.xml` is located on the target system in the directory where the script is run.

Under that assumption, to migrate credentials from a test (or source) LDAP store to a production (or destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="credStore",configFile="t2p-credentials.xml",src="LDAPs
ourceContext",dst="LDAPdestinationContext")
```

To migrate credentials from a test (or source) XML store to a production (or destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="credStore",configFile="t2p-credentials.xml",src="FileS
ourceContext",dst="LDAPdestinationContext")
```

To migrate credentials from a test (or source) DB store to a production (or destination) DB store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="credStore",configFile="t2p-credentials.xml",src="DBSou
rceContext",dst="DBdestinationContext")
```

6.6.3 Migrating Audit Information

This section explains how to migrate various types of audit content.

Migrating Audit Metadata and Audit Policies

Audit metadata consists of audit artifacts such as attribute groups, categories, and events. This information resides in the audit metadata store.

You use the WLST command `migrateSecurityStore` to migrate audit metadata between source and destination contexts. Metadata in either context can reside in XML, database, or LDAP-based stores. Thus, for example, you can:

- migrate audit metadata from an XML file into the domain security store
- migrate audit metadata out of the domain security store into an XML file

Note that:

- There are two types of audit metadata: system definitions and component definitions.
- When migrating audit metadata, you can choose whether to overwrite or preserve existing data in the destination store. Based on your choice, `migrateSecurityStore` follows specific rules to determine how to replace system and component definitions in the destination store, as explained below.

Use the following syntax to migrate audit metadata:

```
migrateSecurityStore(type="auditStore",
configFile="jps_config_file_location",
src="sourceContext",
dst="destinationContext"
[,overWrite="{true|false}"])
```

Using this command, you can migrate metadata of the audit service configured in source context into audit service of destination context. The audit metadata in source and destination context can be in XML, LDAP, and DB-based stores.

The parameters are as follows:

- `configFile` specifies the absolute or relative location of a configuration file. Typically, this configuration file is created just to be used with the script and serves no other purpose. This file contains two jps-contexts that specify the source and destination stores.
- `src` specifies the name of a jps-context in the configuration file passed to the argument `configFile`. It is the source metadata store.
- `dst` specifies the name of another jps-context in the configuration file passed to the argument `configFile`. It is the destination metadata store.
- `overWrite` indicates whether to overwrite existing metadata in the destination store; `true` = always overwrite existing metadata, `false` = do not overwrite existing metadata unless specific conditions are met. Optional, default = `false`. The `overWrite` flag is interpreted as follows:
    1. System definitions are never overwritten regardless of the value of the flag.
    2. If `overwrite=true`, component definitions in the destination store are replaced with the definitions in the source store.

3. If `overwrite=false`, major and minor versions of the component definition in source and destination store are compared. If they have the same major version, and the minor version in the source component definition is higher, the component definition in the destination store is replaced with the definition in the source store. Otherwise, overwriting is skipped.

Migrating Audit Records

Normally, you would not want audit data records from a test environment to be migrated to production; however, to do so, use the database import/export utilities for that purpose. For details, see Section 14.6.5, "Importing and Exporting Data."

6.6.4 Migrating Keystore Service Artifacts

This section explains how to migrate keystore artifacts (namely, keys, and certificates) from one keystore to another. It contains these topics:

- About Keystore Migration
- Migrating Keystore Service Artifacts Within a Domain
- Migrating Keystore Service Artifacts Across Domains

6.6.4.1 About Keystore Migration

You can migrate keystore artifacts (keys and certificates) in two distinct contexts:

- when source and destination keystores lie in the same domain; to be more explicit, when source and destination keystores use the same encryption key.
- when source and destination keystores lie in different domains; that is, when source and destination keystores use different encryption keys.

You use the `migrateSecurityStore` WLST command to migrate security artifacts. For migration within a domain, the command takes a single configuration file; migration across domains requires separate files as explained below.

6.6.4.2 Migrating Keystore Service Artifacts Within a Domain

To migrate keys and certificates with `migrateSecurityStore` when both stores reside in the same domain, create a configuration file to specify the source and destination service instances. Next use the WLST command `migrateSecurityStore` with appropriate options (see the examples at the end of this section).

Note that a single configuration file is sufficient to specify source and destination service contexts when the keystores reside in the same domain.

The following example of a configuration file, named `t2p-keys.xml`, shows how to specify keystore service sources in LDAP, DB, and XML stores, and keystore service destinations in LDAP or DB stores:

```
<?xml version="1.0" encoding="UTF-8" standalone='yes'?>
<jpsConfig xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_
1.xsd" schema-major-version="11" schema-minor-version="1">

<serviceProviders>

<serviceProvider
```

```
class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
name="credstoressp" type="CREDENTIAL_STORE">
 <description>File-based credential provider</description>
 </serviceProvider>

<!--The following service provider configuration serves file-based, LDAP based
     And DB based keystore service instance -->
<serviceProvider type="KEY_STORE" name="keystore.provider"
class="oracle.security.jps.internal.keystore.KeyStoreProvider">
<description>PKI Based Keystore Provider</description>
</serviceProvider>
</serviceProviders>
<serviceInstances>


<!-- Source XML-based keystore service instance -->
 <serviceInstance location="./" provider="keystore.provider"
name="keystore.file.source">
<property name="keystore.provider.type" value="file"/>
<property name="keystore.file.path" value="./"/>
<description>Replace keystore.file.path with the full path of the folder where the
file-based source keystore service keystores.xml is located in the source file
system; typically located in sourceDomain/config/fmwconfig/</description>
 </serviceInstance>


<!-- Source LDAP-based keystore service instance -->
<serviceInstance provider="keystore.provider" name="keystore.ldap.source">
 <description>Replace: A. mySourceDomain and mySourceRootName to appropriate
 values according to your source LDAP directory structure; B.
ldap://mySourceHost.com:3060 with the URL and port number of your source
LDAP</description>
 <property value="bootstrap" name="bootstrap.security.credential.key"/>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://mySourceHost.com:3060" name="ldap.url"/>
<property name="keystore.provider.type" value="ldap"/>
</serviceInstance>


<!-- Source DB-based keystore service instance -->
<serviceInstance provider="keystore.provider" name="keystore.db.source">
 <description>Replace: A. mySourceDomain and mySourceRootName to appropriate
 values according to your source DB </description>
 <property value="cn=mySourceDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=mySourceRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@mySourceHost:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the source datasource
was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="mySourceKeyName" />
 <property name="bootstrap.security.principal.map" value="mySourceMapName" />
 <property name="keystore.provider.type" value="db"/>
 <!-- the values of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the values entered when the bootstrap credential was set up -->
</serviceInstance>


<!-- Destination LDAP-based keystore service instance -->
```

```
 <serviceInstance provider="keystore.provider" name="keystore.ldap.destination">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your destination LDAP directory structure; B.
ldap://myDestHost.com:3060 with the URL and port number of your destination
LDAP</description>
 <property value="bootstrap" name="bootstrap.security.credential.key"/>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="ldap://myDestHost.com:3060" name="ldap.url"/>
 <property name="keystore.provider.type" value="ldap"/>
</serviceInstance>


<!-- Destination DB-based keystore service instance -->
 <serviceInstance provider="keystore.provider" name="keystore.db.destination">
<description>Replace: myDestDomain and myDestRootName to appropriate values
according to your destination DB </description>
 <property value="cn=myDestDomain" name="oracle.security.jps.farm.name"/>
 <property value="cn=myDestRootName" name="oracle.security.jps.ldap.root.name"/>
 <property value="jdbc:oracle:thin:@myDestHost.com:1722:orcl" name="jdbc.url"/>
 <!-- the value of jdbc.url should be the value entered when the destination
datasource was set up -->
 <property value="oracle.jdbc.driver.OracleDriver" name="jdbc.driver"/>
 <property name="bootstrap.security.principal.key" value="myDestKeyName" />
 <property name="bootstrap.security.principal.map" value="myDestMapName" />
 <property name="keystore.provider.type" value="db"/>
 <!-- the values of bootstrap.security.principal.key and
      bootstratp.security.principal.map
      should be the values entered when the bootstrap credential was set up -->
</serviceInstance>


<!-- Bootstrap credentials to access source and destination LDAPs and DBs -->

 <serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.credstore">
 <description>Replace location with the full path of the directory where the
bootstrap file cwallet.sso is located; typically found in
destinationDomain/config/fmwconfig/bootstrap</description>
 </serviceInstance>
 </serviceInstances>

 <jpsContexts>
 <jpsContext name="FileSourceContext">
 <serviceInstanceRef ref="keystore.file.source"/>
 </jpsContext>

 <jpsContext name="LDAPsourceContext">
 <serviceInstanceRef ref="keystore.ldap.source"/>
 </jpsContext>

<jpsContext name="DBsourceContext">
 <serviceInstanceRef ref="keystore.db.source"/>
 </jpsContext>

 <jpsContext name="LDAPdestinationContext">
 <serviceInstanceRef ref="keystore.ldap.destination"/>
 </jpsContext>

<jpsContext name="DBdestinationContext">
```

```
 <serviceInstanceRef ref="keystore.db.destination"/>
 </jpsContext>

 <!-- Do not change the name of the next context -->
 <jpsContext name="bootstrap_credstore_context">
 <serviceInstanceRef ref="bootstrap.credstore"/>
 </jpsContext>
 </jpsContexts>
</jpsConfig>
```

Note that since the migration involves LDAP and/or DB stores, the file includes a jps-context named `bootstrap_credstore_context` that specifies the directory location of the bootstrap credential file `cwallet.sso`.

Examples

Note: The following `migrateSecurityStore` examples assume that the file `t2p-keys.xml` is located on the target system in the directory where the script is run.

To migrate all keys and certificates from a test (source) LDAP store to a production (destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="keyStore",configFile="t2p-keys.xml",
src="LDAPsourceContext",dst="LDAPdestinationContext")
```

To migrate all keys and certificates from a test (source) XML store to a production (destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="keyStore",configFile="t2p-keys.xml",
src="FileSourceContext",dst="LDAPdestinationContext")
```

To migrate keys and certificates for a specific application stripe from a test (source) database store to a production (destination) database store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="stripeKeyStore",configFile="t2p-keys.xml",
src="DBSourceContext",dst="DBdestinationContext", srcStripe="application1",
dstStripe="application2")
```

6.6.4.3 Migrating Keystore Service Artifacts Across Domains

To migrate keystore data when source and target keystores are located in different domains, you need to use two different configuration files. This is because the encryption keys used to encrypt the keystores are different, so the bootstrap credential stores are distinct.

When using the WLST command `migrateSecurityStore`:

- the context of the source keystore in the configuration file is passed as a distinct parameter `srcConfigFile`.
- the context of the destination keystore in the configuration file is passed as parameter `configFile` in a manner similar to in-domain migration as discussed in Section 6.6.4.2.

Example

To migrate all keys and certificates from a test (source) LDAP store to a production (destination) LDAP store, invoke `migrateSecurityStore` in the target system as follows:

```
>migrateSecurityStore(type="keyStore",
srcConfigFile="/source_domain/config/fmwconfig/jps-config.xml",
configFile="/target_domain/config/fmwconfig/jps-config.xml",
src="ksSrc", dst="ksDst")
```

Part III

OPSS Services

This part describes OPSS services in the following chapters:

7 Lifecycle of Security Artifacts

This chapter describes how applications can specify the seeding of application-specific security artifacts in the domain security store when a domain is created or extended. It also describes the procedure to upgrade to 12.1.2.

This chapter is divided into the following sections:

- Introduction
- FMW Domains
- Creating an FMW Domain
- Layered Component Security Artifacts
- Upgrading Security to 12.1.3
- Backing Up and Recovering the OPSS Security Store
- Upgrading Component Audit Definitions to 12c

7.1 Introduction

In 11g domains, the specification of product-specific security artifacts would typically be bundled in the application EAR file and those artifacts would be migrated to the security store at application deployment, provided the deployment descriptors were set appropriately.

In 12c domains, the bundling of product-specific security artifacts in a product template provides a way to seed those artifacts to the domain security store.

7.2 FMW Domains

A domain created or extended using the JRF template is called an FMW domain; the template specifies the provisioning of security artifacts in the domain security store. Specifically, during creation or extension of an FMW domain, the following tasks are accomplished automatically:

- Creation of OPSS security artifacts.
- Creation of a bootstrap wallet seeded with an encryption key.
- Creation, out-of-the-box, of a keystore service that includes:
  - a domain trust store
  - a demo CA keystore
  - a domain identity key store to be used by the Oracle WebLogic Server

- Creation, out-of-the-box, of a trust service wired to the keystore service.
- Creation of three datasources: one for the OPSS schema, another for the OPSS audit viewer schema, and another for the OPSS audit append schema.
- Configuration, out-of-the-box, of a DB-based security store.
- Configuration, by default, of a DB-based audit repository.

Important Note: The JRF template does not create managed servers. In a domain that has been created or expanded with the JRF template, all resources are targeted *only* to the administration server. If later on, a managed server is added to the domain, you must invoke the utility `applyJRF` so that resources are targeted to that manage server too.

Therefore, in order for OPSS and audit data sources to be targeted to managed servers after domain reconfiguration, `applyJRF` must be run with the following format:

```
applyJRF('managedServer1', '<domain_dir>')
```

For more information about `applyJFR`, see *Oracle Fusion Middleware WLST Command Reference for Infrastructure Components*.

For details on component templates, see Layered Component Security Artifacts.

7.3 Creating an FMW Domain

A production FMW domain must be an expanded domain, and an expanded domain requires a DB-based OPSS security store. This database can be a brand new database or a shared database, that is, an existing database associated with some other FMW domain. In regards to resources targeted to manage servers and the use of the utility `applyJRF`, see Important Note in FMW Domains.

The following sections explain how to create an FMW domain:

- Using a New Database Instance
- Sharing a Database Instance

7.3.1 Using a New Database Instance

To create or expand a FMW domain associated with a brand new database, proceed as follows:

1. Create a new database schema. For details, see Section 9.3.1, "Prerequisites to Using a DB-Based Security Store."
2. Use the Configuration Wizard to create or expand a domain, as explained in chapter 5, Configuration Screens, in *Oracle Fusion Middleware Creating WebLogic Domains Using the Configuration Wizard*.

   This task includes, among others, supplying information about the database schema to use, such as the one created in step 1, and selecting to use the JFR Template.

Note 1: The database schema associated with the domain being created must be brand new and cannot be one that has been previously used.

Note 2: When the JRF Template is processed, during creation or extension of a domain, three datasources are automatically created, namely, one for the OPSS schema, another for the OPSS audit viewer schema, and another for the OPSS audit append schema. For more information about other features of the JRF Template, see FMW Domains.

For details about all the tools to use create, extend, or upgrade a domain, see section Template Tools in *Oracle Fusion Middleware Domain Template Reference*.

7.3.2 Sharing a Database Instance

To create an expanded FMW domain to be associated with an existing database in a domain, proceed as follows:

Note: In contrast with the previous procedure, which allows the use of the Configuration Wizard, the following procedure allows using WLST commands only.

---

1. Assuming that the domain `domain1` is using the existing database `db1`, (to which other domains want to joint) use the WLST Offline command `exportEncryptionKey` to export the encrypt key from `domain1` to a specified location, as illustrated in the following called:

   ```
   exportEncryptionKey(location, password)
   ```

   For command details see Oracle Fusion Middleware *Infrastructure Security WLST Command Reference*.

2. Use a WLST script like the following to create an expanded FMW domain, which is to share the database `db1`:

   ```
   ## arg1 - wls.jar loc
   ## arg2 - jrf.jar loc
   ## arg3 - domain home
   ## arg4 - adminserver port
   ## arg5 - Db host
   ## arg6 - db port
   ## arg 7 - DB service name (pdb)
   ## arg8 - STB schema user,

   readTemplate(sys.argv[1],"Expanded")
   cd('/Security/base_domain/User/weblogic')
   cmo.setName('weblogic')
   cmo.setPassword('welcome1')
   writeDomain(sys.argv[3])
   closeTemplate()

   #Set AdminServer Port
   readDomain(sys.argv[3])
   ```

```
cd('/Servers/AdminServer')
set('ListenAddress','')
set('ListenPort', int(sys.argv[4]))
updateDomain()
closeDomain()

readDomain(sys.argv[3])
addTemplate(sys.argv[2])

cd('/JDBCSystemResource/LocalSvcTblDataSource/JdbcResource/LocalSvcTblDataSourc
e')
cd('JDBCDriverParams/NO_NAME_0')
set('DriverName','oracle.jdbc.OracleDriver')
set('PasswordEncrypted', 'myPassw')
set('URL','jdbc:oracle:thin:@'+sys.argv[5]+':'+sys.argv[6]+'/'+sys.argv[7])
set('UsePasswordIndirection', 'false')
set('UseXADataSourceInterface', 'false')
create('myProps','Properties')
cd('Properties/NO_NAME_0/Property/user')
cmo.setValue(sys.argv[8])

getDatabaseDefaults()

setSharedSecretStoreWithPassword(location, password)
updateDomain()
```

where `location` and `password` have the same values as those used in the call to `exportEncryptionKey`. For syntax details, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

3. Start all servers in `domain1` and in the domain just created with the WLST script. Both domains share the same security store.

7.4 Layered Component Security Artifacts

To streamline the seeding and processing of security artifacts, components consuming OPSS need to provide a template during domain creation or extension. This template defines and bundles artifacts specific to just the components that are required for the component's execution, and includes the files listed in Table 7–1.

Table 7–1 Files in a Component Template Used by OPSS

File name	Description	Location relative to the template root folder
component-security-info.xml	Required. Specifies the lifecycle of security artifacts.	./security/component-security-info.xml
jazn-data.xml	Optional. Specifies authorization policies.	./security/authorization/jazn-data.xml
keystore.xml	Optional. Specifies the keystore metadata.	./security/keystore/keystore.xml
credentials.xml	Optional. Specifies the credential metadata used by the component.	./security/credential/credentials.xml
component_events.xml	Optional. Specifies the audit metadata.	./security/audit/component_events.xml
component_events_xlf.jar	Optional. Specifies the localized audit metadata.	./security/audit/component_events_xlf.jar

---

Note: OPSS security artifacts bundled with a product template, *require* a `component-security-info.xml` file to indicate how artifacts are handled.

7.5 Upgrading Security to 12.1.3

This section explains how to upgrade security artifacts from release 11.1.1.6, or 11.1.1.7, or 12.1.2 to release 12.1.3.

After the upgrade is completed, if the Keystore must be moved from JKS to KSS, see Section 12.2.2.8, "Importing a Keystore at the Command Line."

7.5.1 Before Upgrading

Before upgrading the OPSS security store, back it up so that it can be recovered in case the upgrade fails. For details about backing up the security store, see Backing Up and Recovering the OPSS Security Store.

The upgrade procedure varies depending on the type of security store you start from. The security store to be upgraded can be file-, OID-, or DB-based. Note the procedures below vary according to the type of source audit data store (file- or DB-based).

- Upgrading a DB-Based Security Store
- Upgrading an OID-Based Security Store
- Upgrading a Shared Security Store
- Upgrading a File-Based Security Store

7.5.2 Upgrading a DB-Based Security Store

To upgrade a DB-based security store from 11.1.1.6, or 11.1.1.7, or 12.1.2 to 12.1.3, proceed as follows:

1. Run Oracle Fusion Middleware Upgrade Assistant to upgrade the OPSS schema and, if the source audit data store is DB-based, to upgrade the audit schema.

   In the Available Components page, be sure to select Oracle Platform Security Services. For details about Upgrade Assistant, see *Oracle Fusion Middleware Upgrading with the Upgrade Assistant*.

2. If you are upgrading from 12.1.2, skip this step and continue to step 3; otherwise, run the Repository Creation Utility (RCU) to create just the service table schema. For details about RCU, see *Creating Schemas with the Repository Creation Utility*.

3. If the source audit data store is file-based, run RCU to create an audit schema.

4. Run the Reconfiguration Wizard to effect the domain reconfiguration and to upgrade OPSS data, DIT, configuration and product security artifacts.

   Note the following points about this step:

   - In the Database Configuration Type page, select RCU Data and enter the service table schema, and then click Get RCU Configuration.
   - In the Component Datasources page, select the following components and enter the schema details:
     – OPSS Audit Schema - IAU_APPEND
     – OPSS Audit Viewer - IAU_VIEWER

   - OPSS Schema - OPSS

5. If the source audit data store is file-based, an audit data source must exist. For details, see Section 14.2.2.

> **Note:** Joining from a 12c expanded domain to a 12c-upgraded domain is not supported.

OPSS supports Edition-Based Redefinition (EBR). For details about setting the default edition before using OPSS, see the procedures in the following sections:

- Section 2.7.5, in *Oracle Fusion Middleware Planning an Upgrade of Oracle Fusion Middleware*.
- Section 1.6.3, in *Oracle Fusion Middleware Upgrading with the Upgrade Assistant*.

For complete details about upgrading Oracle Fusion Middleware, see *Oracle Fusion Middleware Planning an Upgrade of Oracle Fusion Middleware*.

7.5.3 Upgrading an OID-Based Security Store

To upgrade an OID-based security store from 11.1.1.6, or 11.1.1.7, or 12.1.2 to 12.1.3, proceed as follows:

1. Run Oracle Fusion Middleware Upgrade Assistant to upgrade the OPSS schema and, if the source audit data store is DB-based, to upgrade the audit schema.

   In the Available Components page, be sure to select Oracle Platform Security Services. For details about Upgrade Assistant, see *Oracle Fusion Middleware Upgrading with the Upgrade Assistant*.

2. Run the Repository Creation Utility (RCU) to create the OPSS schema. The details of this schema are automatically inserted in the Database Configuration Type page. (If the audit schema is not present, it is also created in this step.) For details about RCU, see *Creating Schemas with the Repository Creation Utility*.

3. Run the Reconfiguration Wizard to effect the domain reconfiguration and to upgrade OPSS data, DIT, configuration and product security artifacts.

   Note the following points about this step:

   - In the Database Configuration Type page, select RCU Data and enter the service table schema created in step 2, and then click Get RCU Configuration.
   - In the Component Datasources page, select the following components and enter the schema details:
     - OPSS Audit Schema - IAU_APPEND
     - OPSS Audit Viewer - IAU_VIEWER
     - OPSS Schema - OPSS

4. If the source audit data store is file-based, an audit data source must exist. For details, see Section 14.2.2, "About Audit Data Sources."

7.5.4 Upgrading a Shared Security Store

This section explains the extra steps necessary to upgrade a security store shared (joined) by several domains. The procedures below vary according to the version of the shared the security store.

Before upgrading the OPSS security store, back it up so that it can be recover in case the upgrade fails. For details about backing up the security store, see Backing Up and Recovering the OPSS Security Store.

This section contains the following topics:

- Upgrading a Shared 12.1.2 Security Store
- Upgrading a Shared 11g Security Store

7.5.4.1 Upgrading a Shared 12.1.2 Security Store

To upgrade a 12.1.2 security store shared by several domains to 12.1.3, proceed as follows:

1. Shutdown all domains sharing the store to be upgraded.
2. Run Oracle Fusion Middleware Upgrade Assistant to upgrade the OPSS schema of the shared security store and, if the source audit data store is DB-based, to upgrade the audit schema.

   In the Available Components page, be sure to select Oracle Platform Security Services. For details about Upgrade Assistant, see *Oracle Fusion Middleware Upgrading with the Upgrade Assistant*.
3. In each of the domains sharing the security store, run the Reconfiguration Wizard to effect the domain reconfiguration and to upgrade OPSS data, DIT, configuration and product security artifacts.

   Note the following points about this step:

   - In the Database Configuration Type page, select RCU Data and enter the service table schema created in step 2, and then click Get RCU Configuration.
   - In the Component Datasources page, select the following components and enter the schema details:
     – OPSS Audit Schema - IAU_APPEND
     – OPSS Audit Viewer - IAU_VIEWER
     – OPSS Schema - OPSS

   Note: OPSS data will be upgraded when the Reconfiguration Wizard is run for the first time in *any* of the domains; after that, when the Reconfiguration Wizard is run in any of the other domains, it will reconfigure the domain to use 12.1.3 binaries in that domain.
4. Restart all domains sharing the security store.

7.5.4.2 Upgrading a Shared 11g Security Store

To upgrade a 11.1.1.6 or 11.1.1.7 shared security store to 12.1.3, proceed as follows:

1. Shutdown all domains sharing the store to be upgraded.
2. Upgrade binaries in each of the domains sharing the security store.

   For complete details about upgrading Oracle Fusion Middleware, see *Oracle Fusion Middleware Planning an Upgrade of Oracle Fusion Middleware*.

3. Run Oracle Fusion Middleware Upgrade Assistant to upgrade the OPSS schema of the shared security store and, if the source audit data store is DB-based, to upgrade the audit schema.

   In the Available Components page, be sure to select Oracle Platform Security Services. For details about Upgrade Assistant, see *Oracle Fusion Middleware Upgrading with the Upgrade Assistant*.

4. Run the 11g `upgradeOpss` command in each of the domains sharing the security store. When the command is run first time from a domain, it will upgrade the data of the security store and configuration of that domain; when run from any other domain, it will upgrade only the configuration of that domain.

5. Restart all upgraded domains.

7.5.5 Upgrading a File-Based Security Store

To upgrade a file-based security store from 11.1.1.6, or 11.1.1.7, to 12.1.3, proceed as follows:

1. If the source audit data store is file-based, run RCU version 11g to create both the OPSS schema and the OPSS Audit schema. Otherwise, if the audit data store is DB-based, run RCU version 11g to create just the OPSS schema.

2. If the audit data store is file-based, the target store must have an audit data source. See Section 14.2.2.

3. Reassociate the file-based security store to a DB-based security store using the command `reassociateSecurityStore`; for details, see Section 10.4.1, "reassociateSecurityStore."

4. Upgrade the reassociated DB-based security store to 12.1.3 as explained in section Upgrading a DB-Based Security Store.

7.6 Backing Up and Recovering the OPSS Security Store

This section describes how to backup and recover a domain's OPSS security store. The procedure varies according to the type of OPSS store to process (DB-based or OID-based).

In addition to backing up the security store, note that the following security store configuration and data files must also be saved for a possible recovery (a star stands for all files in the directory):

```
{domain}/Config/config.xml
{domain}/Config/Fmwconfig/jps-config.xml
{domain}/Config/Fmwconfig/jps-config-jse.xml
{domain}/Config/Fmwconfig/cwallet.sso
{domain}/Config/Fmwconfig/keystores.cml
{domain}/Config/Fmwconfig/audit-store.xml
{domain}/Config/Fmwconfig/system-jazn-data.xml
{domain}/Config/Fmwconfig/ids-config.xml
{domain}/Config/Fmwconfig/mbeans/jps_mbeans.xml
{domain}/Config/Fmwconfig/bootstrap/cwallet.sso
```

The above files are typically backed up as part of an Oracle WebLogic Server domain backup. For details about an Oracle WebLogic domain backup, see the following topics in the *Administering Oracle Fusion Middleware*:

- 17.3 Performing a Backup
- 18.2.2 Recovering an Oracle WebLogic Server Domain

This section contains the following topics:

- Backing Up and Recovering a DB-Based Security Store
- Backing Up and Recovering an OID-Based Security Store
- Recommendations

7.6.1 Backing Up and Recovering a DB-Based Security Store

The procedure described in this section uses the Oracle Database client Recovery Manager (RMAN) that allows active database duplication, a tool that is typically also used to automate backup strategies and recoveries. For full details about RMAN, see the following topics in the *Oracle Database Backup and Recovery User's Guide*:

- Getting Started with RMAN
- Performing Complete Database Recovery

The following procedure describes how to backup a domain DB-based security store instance in host A to an instance in host B; it assumes that the security store in host A has a jdbc url set to `proddb`, and the goal is to get that same domain to work with a (cloned) DB-based store with jdbc url set to `testdb` (in host B).

Under the above assumptions, to backup a DB-based security store, proceed as follows:

1. Prepare the instance `testdb` in host B, as follows:

   1. Create a pfile for the new instance: that is, create the file inittestdb.ora with a contents as illustrated be the following lines:

      ```
      #
      db_name=testdb
      #
      ```

   2. Add `testdb` to listener.ora, as illustrated in the following line:

      SID_LIST_LISTENER = (SID_LIST=(SID_DESC=(SID_
      NAME=testdb)(GLOBAL_DBNAME=testdb)(ORACLE_
      HOME=/ade/b/3882746433/oracle))

   3. Add testdb/proddb in tnsnames.ora, as illustrated in the following lines:

      proddb=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=hostA.com)(
      PORT=XXXX))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_
      NAME= proddb)))

      testdb=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=hostB.com)(P
      ORT=YYYY))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_
      NAME=testdb)))

   4. Restart the listener, as illustrated in the following line:

      lsnrctl stop, lsnrctl start

   5. Start the new instance using the pfile in nomount mode, as illustrated in the following lines:

      $ export ORACLE_SID=testdb

      $ sqlplus / as sysdba

      SYS@testdb SQL>startup nomount pfile=/scratch/rdbms/dbs/inittestdb.ora

2. Use RMAN to clone `proddb` to `testdb`, as follows:

1. Add add testdb/proddb in tnsnames.ora, as illustrated in the following lines:

   proddb=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhostA.co m)(PORT=XXXX))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_ NAME= proddb)))

   testdb=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhostB.com)(PORT=YYYY))(CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_ NAME=testdb)))

2. Make sure that instance `proddb` is using the spfile. If this is not yet the case, generate a binary spfile from the init file by login in as sysdba and running "create spfile from pfile". Then, restart the server.

3. Restart the listener, as illustrated in the following line:

   lsnrctl stop, lsnrctl start

4. Decide how to generate the names for the duplicate database files; specifically, decide how to name the control files, the datafiles, the online redo log files, and the tempfiles.

   For example, if in `proddb`, the files on host A are in the directory /oradata/proddb, and you want the them to be saved in /oradata/testdb on host B, then you would specify DB_FILE_NAME_CONVERT '/proddb',' /testdb', as in the sequence below.

   Run RMAN to clone `proddb` to `testdb` as illustrated in the following lines:

   ```
   $rman
   RMAN> CONNECT TARGET SYS@proddb
   RMAN> CONNECT AUXILIARY SYS@testdb
   RMAN> DUPLICATE TARGET DATABASE TO testdb
             FROM ACTIVE DATABASE
             DB_FILE_NAME_CONVERT '/proddb','/testdb'
             SPFILE
               PARAMETER_VALUE_CONVERT '/proddb','/testdb'
               SET LOG_FILE_NAME_CONVERT '/proddb','/testdb';
   ```

   Make sure that the RMAN sequence above completes successfully.

3. (Optional) Verify the backed up DB instance `testdb` works as expected, by switching your domain to use the just backed up database as the OPSS security store:

   1. Stop the Oracle WebLogic Server
   2. Change the jdbc url from `proddb` to `testdb` in the files `{domain}/config/fmwconfig/jps-config.xml` and `{domain}/config/jdbc/*xml`
   3. Restart the Oracle WebLogic Server
   4. Ensure that the domain security works as expected

7.6.2 Backing Up and Recovering an OID-Based Security Store

To backup a source Oracle Internet Directory store to a destination Oracle Internet Directory store proceed as follows:

1. In the system where the source Oracle Internet Directory is located, produce an LDIF file by running `ldifwrite` as illustrated in the following line:

   ```
   >ldifwrite connect="srcOidDbConnectStr" basedn="cn=jpsnode"
   ```

```
ldiffile="srcOid.ldif"
```

This command writes all entries under the node `cn=jpsnode, c=us` to the file `srcOid.ldif`. Then move this file to the destination Oracle Internet Directory file system so it is available to the commands that follow.

2. In the destination Oracle Internet Directory system, ensure that the JPS schema has been seeded.
3. In the destination Oracle Internet Directory system, verify that there are no schema errors or bad entries by running `bulkload` as illustrated in the following line:

   ```
   >bulkload connect="dstOidDbConnectStr" check=true generate=true restore=true
   file="fullPath2SrcOidLdif"
   ```

   If duplicated DNs (common entries between the source and the destination directories) are detected, review them to prevent unexpected results.
4. Load data into the destination Oracle Internet Directory, by running `bulkload` as illustrated in the following line:

   ```
   >bulkload connect="dstOidDbConnectStr" load=true file="fullPath2SrcOidLdif"
   ```

For more details about the above commands, see the following topics in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*:

- 15.6 Dumping Data from Oracle Internet Directory to a File by Using ldifwrite
- 37.2.1 Migrating LDAP Data by Using an LDIF File and bulkload

7.6.3 Recommendations

It is recommended that the OPSS security store be backed up periodically, according to a schedule appropriate to your enterprise. In addition, an unscheduled backup is recommended soon after the following events:

- a new encryption key has been (automatically) generated for the domain
- new policies have been created
- new binding credentials are set

7.7 Upgrading Component Audit Definitions to 12c

About the Audit Schema Upgrade Tool

For a component using the Release 11*g* audit model, you can upgrade the audit definitions to the dynamic metadata model used in Release 12c with the `AuditSchemaUpgradeTool` offline tool.

Tool Syntax

The syntax is as follows:

```
java -classpath $MW_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-manifest.jar
oracle.security.audit.tools.AuditSchemaUpgradeTool
-s source_file
-t target_file_or_directory
[, -v component_def_version]
```

where:

- `source_file` is the Release 11*g* audit event definition file, such as `component_events.xml`.
- `target_file_or_directory` is either a) the file in which the generated 12g dynamic model definition file will be stored, or b) a target directory, and the generated dynamic model definition file is placed here with the default file name *source_file*`_dynamic.xml`.
- `component_def_version` is the version number of the generated 12c definition. If not specified, default value is 1.0.

Examples

In this example, the OPSS component definition is upgraded to the Release 12*c* dynamic model using the default version number, but specifying the target file component_events_JPS.xml, use .

```
java -classpath $MW_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-manifest.jar
oracle.security.audit.tools.AuditSchemaUpgradeTool
-s component_events.xml
-t component_events_OPSS.xml
```

In this example, the OPSS component definition is upgraded to the Release 12*c* dynamic model by specifying a version number and a target directory; the tool uses the default target file name component_events_dynamic.xml:

```
java -classpath $MW_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-manifest.jar
oracle.security.audit.tools.AuditSchemaUpgradeTool
-s component_events.xml
-t /scratch/example -v 2.0
```

Notes

1. This tool supports Release 11*g* audit definition files containing no more than one component definition.
2. The component's `displayName` (if any) present in the Release 11*g* component_events.xml file should be added as the `AuditConfig` attribute before upgrade. For example:

   ```
   <AuditConfig componentType="SOA-HCFP"
   xmlns="http://xmlns.oracle.com/ias/audit/audit.xsd" displayName="Oracle SOA
   Suite for healthcare integration">
   ```

   Otherwise, after upgrade, manual update of displayName is required if the component requires a displayName. For example, in target events definition file, AuditComponent node can have a displayName such as:

   ```
   <AuditComponent minor="0" major="1" componentType="SOA-HCFP"
   displayName="Oracle SOA Suite for healthcare integration">
   ```

8 Configuring the Identity Store Service

This chapter describes how to use the OPSS identity store service. The identity store service enables querying for user and role (group) information.

This chapter includes the following sections:

- Introduction to the Identity Store Service
- Configuring the Identity Store Provider
- Configuring the Identity Store Service
- Querying the Identity Store Programmatically
- Configuring SSL for the Identity Store Service

See Also: "About the Identity Directory Service" in *Developer's Guide for Identity Governance Framework*.

8.1 Introduction to the Identity Store Service

This section describes key concepts of the OPSS identity store service:

- About the Identity Store Service
- Service Architecture
- Application Server Support

8.1.1 About the Identity Store Service

The identity store service enables you to query the identity store for user and role (group) information.

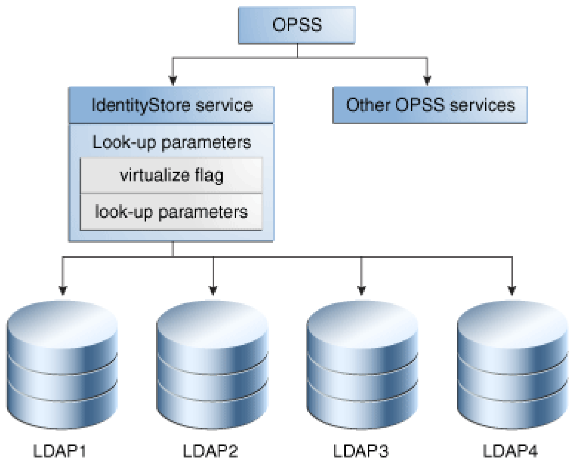
By default, a service instance supports querying against a single LDAP identity store. You can configure the service to support a virtualized identity store which queries multiple LDAP identity stores. For details about this feature, known as identity virtualization, see Section 8.3, "Configuring the Identity Store Service".

8.1.2 Service Architecture

Figure 8–1 shows the architecture of the identity store service. Depending on the configuration, the service can support either an XML file or one or more LDAP servers as the identity store.

When the service is configured for LDAP, it queries a single LDAP store by default. You can also configure the service to query multiple LDAP stores.

Figure 8–1 The OPSS Identity Store Service



8.1.3 Application Server Support

The identity store service is supported in the following platform:

- Oracle WebLogic Server

8.1.4 Java SE Support

The identity store service is available in a stand-alone Java SE environment.

For more information, see Section 8.3.6, "Defining Configuration in Java SE Environments".

8.2 Configuring the Identity Store Provider

Before using the identity store service, you must configure the identity store provider. OPSS supports both file-based (XML) and LDAP-based providers.

This fragment from the `jps-config.xml` file shows the configuration of both XML and LDAP providers. The `serviceProvider` elements are children of the `serviceProviders` element.

```
<serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
 class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
      <description>LDAP-based IdentityStore Provider</description>
</serviceProvider>

<serviceProvider type="IDENTITY_STORE" name="idstore.xml.provider"
 class="oracle.security.jps.internal.idstore.xml.XmlIdentityStoreProvider">
      <description>XML-based IdentityStore Provider</description>
</serviceProvider>
```

If Active Directory is the identity store provider, set USERNAME_ATTR and USER_LOGIN_ATTR properties to be `sAMAccountName` in `jps-config.xml` (or `jps-config-jse.xml`) if the default value (`cn`) is to be overridden. For example:

```
<property value="sAMAccountName" name="username.attr"/>
<property value="sAMAccountName" name="user.login.attr"/>
```

Note: When the virtualize flag is set to true, do not set `user.login.attr` and `username.attr` properties.

For details, see Section 9.7.2, "Configuring the Identity Store Provider".

8.3 Configuring the Identity Store Service

This section describes how to configure the identity store service to LDAP-based stores. Topics include:

- What is Configured?
- Understanding Configuration in WebLogic Server
- Configuring Split Profiles
- Configuring Custom Authenticators
- Understanding Configuration in Other Application Servers
- Defining Configuration in Java SE Environments

See Also: Appendix F, "OPSS System and Configuration Properties".

8.3.1 What is Configured?

This section explains the different configuration parameters for the identity store service. It includes:

- Configuring Multi-LDAP Lookup
- Global/Connection Parameters
- Back-End/Connection Parameters

8.3.1.1 Configuring Multi-LDAP Lookup

You use the following parameters to configure the service for multi-LDAP queries:

- The `virtualize` property - This property can be either true (multi-LDAP lookup) or false (single-LDAP lookup). The default is `false`.
- Global Connection Parameters (if '`virtualize`' is enabled) - The calling application uses these parameters to specify global LDAP configuration such as the search base, create base, and so on. If any of these parameters are not configured, OPSS uses default values.
- Back-end Connection Parameters - These parameters are specific to each LDAP store. One set of back-end parameters is specified for each LDAP. You do not need to set these parameters unless you want to overwrite existing values.

8.3.1.2 Global/Connection Parameters

Table 8–1 shows the global parameters and their default values, if applicable.

Table 8–1 Global LDAP Identity Store Parameters

Parameter	Default Value
group.create.bases	same as user.create.bases

Table 8–1 (Cont.) Global LDAP Identity Store Parameters

Parameter	Default Value
group.filter.object.classes	groupofuniquenames The global value is used if explicitly provided.
group.mandatory.attrs	-
group.member.attrs	uniquemember
group.object.classes	groupofuniquenames
group.search.bases	-
group.selected.create.base	-
group.selected.search.base	-
groupname.attr	cn If the global value is explicitly given, it is used.
max.search.filter.length	-
search.type	-
user.create.bases	If only one authenticator, uses it as the create base value. If multiple authenticators, no default value is set; user must explicitly set the global value.
user.filter.object.classes	inetorgperson
user.login.attr	uid
user.mandatory.attrs	-
user.object.classes	inetorgperson If the global value is explicitly given, it is used.
user.search.bases	Same as group.search.bases
username.attr	cn The global value is used if explicitly provided.

See Also: Section F–7, " Generic OID Properties"

8.3.1.3 Back-End/Connection Parameters

As mentioned earlier, these are specific to the back-end LDAP store. For details, see:

- Table F–6, " LDAP-Based Identity Store Properties"
- Section F.2.2, "Policy Store Properties"

8.3.2 Understanding Configuration in WebLogic Server

You configure LDAP authenticators in Oracle WebLogic Server using either the WebLogic console or WLST command-line. At runtime, Oracle WebLogic Server passes the configuration details to OPSS. Oracle WebLogic Server allows configuring multiple authenticators in a given context, and selects the first authenticator to initialize the identity store service by default. This process is explained in Section 3.2.2.1, "Multiple Authenticators".

After the authenticators are configured, the identity store service can be set up to query one LDAP identity store or multiple stores. Configuring for multiple stores requires setting up the `virtualize` property.

This section explains how to set up these options.

8.3.2.1 Configuring the Service for Single LDAP

You can configure the identity store service to query only one LDAP store. See Example 8–2 which displays a fragment of the `jps-config.xml` file with a single LDAP service instance.

8.3.2.2 Configuring the Service for Multiple LDAP Without `virtualize` Property

In cases when the `virtualize` property cannot be set, you can configure the identity store service to query more than one LDAP store and override the configuration in WebLogic Server. See Example 8–1 which displays a fragment of the `jps-config.xml` file where a multiple LDAP service instance is defined using a comma separated list of LDAP urls.

Example 8–1 Multiple-LDAP Configuration Without Setting virtualize Property

```
<property name="ldap.url", value="ldap://host1:port1,ldap://host2:port2" />
```

8.3.2.3 Configuring the Service for Multiple LDAP using Fusion Middleware Control

As in the single LDAP setup, you start by configuring the authentication providers in Oracle WebLogic Server.

To configure service for multiple LDAP in Fusion Middleware Control:

1. Select the WebLogic domain in the navigation pane on the left.
2. Navigate to Security, then Security Provider Configuration.
3. Expand the Identity Store Provider section of the page.
4. Click **Configure** (corresponding to "Configure parameters for User and Role APIs to interact with identity store").
5. The Identity Store Configuration page appears.
6. Under Custom Properties, click **Add**.
7. Add the new property as follows:

   ```
   Property Name=virtualize
   Value=true
   ```

   Note: Be sure to add the property to the identity store service instance in the default context.

8. Click **OK**.

8.3.2.4 Configuring the Service for Multiple LDAP Using WLST

To configure the virtualize property using WLST:

1. Create a `py` script file to connect to the administration server in the domain of interest. You must specify the `userName`, `userPass`, `localHost`, and `portNumber` for the operation.

   See Appendix E.1, "Configuring OPSS Service Provider Instances with a Script" for details about this script.
2. Navigate to `$ORACLE_HOME/common/bin`.

3. Run the `wlst.sh` command to execute the script.

   For example, if the domain configuration file contains an authenticator named idstore.ldap, the following command:

   ```
   wlst.sh /tmp/updateServiceInstanceProperty.py -si idstore.ldap
   -key "virtualize" -value "true"
   ```

   configures the provider for multi-LDAP lookup.

   See Also: Section E.1, "Configuring OPSS Service Provider Instances with a Script".

8.3.2.5 Configuring the Timeout Setting Using WLST

To configure the timeout setting using WLST:

1. Run the following WLST command to list all adapters: `listAdapters()`.
2. Run the following WLST command to set the timeout for each adapter. 120 second timeout is an example, set to zero for no timeout.

   ```
   modifyLDAPAdapter('<ADAPTER NAME>', 'OperationTimeout',
   120000)
   ```

3. Restart WebLogic Server.

8.3.2.6 Configuring Other Parameters

If desired, you can update `jps-config.xml` to set query parameters listed in Section 8.3.1, "What is Configured?". These parameters are optional; default values are provided.

8.3.2.7 Restarting Servers

After configuring for multi-LDAP query, restart the WebLogic administration and managed servers.

8.3.2.8 Viewing the Configuration File

Example 8–2 shows a sample `jps-config.xml` file configured for single-LDAP queries in the Oracle WebLogic Server environment:

Example 8–2 Single-LDAP Configuration in Oracle WebLogic Server

```
<!-- JPS WLS LDAP Identity Store Service Instance -->
        <serviceInstance name=idstore.ldap provider=idstore.ldap.provider>
            <property name=idstore.config.provider
                      value=oracle.security.jps.wls.internal.idstore.
                      WlsLdapIdStoreConfigProvider/>
            <property name=CONNECTION_POOL_CLASS
                      value=oracle.security.idm.providers.stdldap.JNDIPool/>
        </serviceInstance>
```

Example 8–3 shows a sample `jps-config.xml` file configured for multi-LDAP queries in the Oracle WebLogic Server environment:

Example 8–3 Multi-LDAP Configuration in Oracle WebLogic Server

```
<jpsConfig xmlns="http://xmlns.example.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.example.com/oracleas/schema/11/jps-config-11_
```

```
1.xsd" schema-major-version="11" schema-minor-version="1">

 <serviceProviders>
        <serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"

class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
            <description>LDAP-based IdentityStore Provider</description>
        </serviceProvider>
 </serviceProviders>

 <serviceInstances>
        <!-- IDstore instance connecting to multiple ldap  -->
        <serviceInstance name="idstore.virtualize"
provider="idstore.ldap.provider">

            <!-- following property indicates using WLS ldap Authenticators -->
            <property name="idstore.config.provider"

value="oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfigProvider"/>

     <!-- following property enables virtualization that is, support for multiple
stores -->
            <property name="virtualize" value="true"/>

            <!-- Front end ldap properties (if not supplied, will use default
values) -->
            <extendedProperty>
                <name>user.create.bases</name>
                <values>
                   <value>cn=users_front,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <extendedProperty>
                <name>group.create.bases</name>
                <values>
                  <value>cn=groups_front,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
        </serviceInstance>
 </serviceInstances>

  <jpsContexts default="default">

        <!-- the identity store uses multiple ldaps -->
        <jpsContext name="default">
            <!-- use multiple ldap -->
            <serviceInstanceRef ref="idstore.virtualize"/>
            <!-- .....other services -->
        </jpsContext>
  </jpsContexts>

</jpsConfig>
```

Note that:

- the `virtualize` property of the service instance is set to `true`, enabling multi-LDAP queries.
- the `extendedProperty` element enables you to set front-end parameters if desired to override default values.

For more information, see "Front-End Parameters" in Section 8.3.1, "What is Configured?".

8.3.3 Configuring Split Profiles

Identity Virtualization supports a "split profile," where an application makes use of attributes for a single identity that are stored on two different sources.

This feature requires additional configuration beyond that described in this chapter. For details, see Appendix I, "Adapter Configuration for Identity Virtualization".

8.3.4 Configuring Custom Authenticators

OPSS supports the set of LDAP-based Oracle WebLogic Server authentication providers (WebLogic authenticators) for access to identity stores. If the out-of-the-box WebLogic authenticators are not applicable to your LDAP server type, you can customize a generic authenticator for this task.

This section explains how you can configure such an authenticator when the 'virtualize' flag is enabled for the identity store service.

Note the following points in this context:

- When using a generic LDAP authenticator, you must tell the Identity Store Service of the exact LDAP type so that it can find the proper LDAP plug-in. You do this by overriding the '`idstore.type`' property in `jps-config.xml`.
- As the '`virtualize`' flag is enabled, and the Oracle WebLogic Server domain has two or more authenticators (for example, the defaultAuthenticator and generic LDAP), you must tell the Identity Store Service which LDAP server's '`idstore.type`' is to be overridden.

You provide this information as follows in `jps-config.xml`:

```
<serviceInstance name="idstore.ldap" provider="idstore.ldap.provider">
    <property name="idstore.config.provider"
value="oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfigProvider"
/>
    <property name="CONNECTION_POOL_CLASS"
value="oracle.security.idm.providers.stdldap.JNDIPool" />

    <property value="true" name="virtualize" />

    <!-- the following refer to the Generic ldap name configured on the WLS
authenticator provider, you get this name from WebLogic config.xml file or admin
console  -->
    <serviceInstanceRef ref="myGenericLDAPName"/>

  </serviceInstance>


  <!-- the following provide the overriding to the Generic ldap (e.g. AD)
-->
  <!-- "myGenericLDAPName" is the name used on WLS for the generic LDAP
authenticator provider  -->
  <serviceInstance name="myGenericLDAPName" provider="idstore.ldap.provider">

    <!-- the following overrides the 'idstore.type' property to
"ACTIVE_DIRECTORY" -->
    <property name="idstore.type" value="ACTIVE_DIRECTORY" />
  </serviceInstance>
```

If you must override an additional LDAP provider instance, simply add another similar entry to the file.

8.3.5 Understanding Configuration in Other Application Servers

Topics in this section include:

- Configuring the Service for Single LDAP
- Configuring the Service for Multiple LDAP

8.3.5.1 Configuring the Service for Single LDAP

See the example in Section 25.3.2, "Configuring an LDAP Identity Store in Java SE Applications," for details.

8.3.5.2 Configuring the Service for Multiple LDAP

To configure the identity store service to handle multiple LDAPs in third-party application servers:

1. Modify the `jps-config.xml` file to configure service instances for each supported LDAP directory.
2. Restart the application server to make the changes effective.

Example 8–4 shows a sample `jps-config.xml` file configured to run multi-LDAP queries for third-party application servers:

Example 8–4 Multi-LDAP Configuration in Third-Party Application Servers

```
<jpsConfig xmlns="http://xmlns.example.com/oracleas/schema/11/jps-config-11_1.xsd"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.example.com/oracleas/schema/11/jps-config-11_
1.xsd" schema-major-version="11" schema-minor-version="1">

 <serviceProviders>
        <serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
            <description>LDAP-based IdentityStore Provider</description>
        </serviceProvider>
 </serviceProviders>

 <serviceInstances>
          <!-- instance 'idstore.oid'  to represent an ldap server 'oid' -->
    <serviceInstance name="idstore.oid" provider="idstore.ldap.provider">
            <property name="subscriber.name" value="dc=us,dc=example,dc=com"/>
            <property name="idstore.type" value="OID"/>
            <property name="security.principal.key" value="oid.ldap.credentials"/>
            <property name="security.principal.alias" value="JPS"/>
            <property name="ldap.url"

value="ldap://hostname.example.com:389,ldap://hostname.example.com:389"/>
            <extendedProperty>
                <name>user.search.bases</name>
                <values>
                   <value>cn=users,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <extendedProperty>
                <name>group.search.bases</name>
                <values>
                  <value>cn=groups,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <property name="username.attr" value="uid" />
            <property name="groupname.attr" value="cn" />
    </serviceInstance>

          <!-- instance 'idstore.ad' to represent an ldap server 'ad' -->
    <serviceInstance name="idstore.ad" provider="idstore.ldap.provider">
            <property name="subscriber.name" value="dc=us,dc=example,dc=com"/>
            <property name="idstore.type" value="ACTIVE_DIRECTORY"/>
            <property name="security.principal.key"
value="msad.ldap.credentials"/>
            <property name="security.principal.alias" value="JPS"/>
            <property name="ldap.url"
value="ldap://hostname.example.com:389,ldap://hostname.example.com:389"/>
            <extendedProperty>
                <name>user.search.bases</name>
                <values>
                   <value>cn=users,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <extendedProperty>
                <name>group.search.bases</name>
                <values>
                  <value>cn=groups,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <property name="username.attr" value="uid" />
            <property name="groupname.attr" value="cn" />
     </serviceInstance>

         <!-- IDStore service "idservice.virtualize" to connect to multiple ldaps
( 'oid' and 'ad') using libOVD-->
        <serviceInstance name="idservice.virtualize"
         provider="idstore.ldap.provider">

     <!--following property enables virtualization that is, support for multiple
stores -->
               <property name="virtualize" value="true"/>
               <!-- backend ldap instance "idstore.oid"-->
               <serviceInstanceRef ref="idstore.oid"/>
               <!-- backend ldap instance "idstore.ad"-->
               <serviceInstanceRef ref="idstore.ad"/>
            <!-- Front end ldap properties (if not supplied, will use default
values) -->
            <extendedProperty>
                <name>user.create.bases</name>
                <values>
                   <value>cn=users_front,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
            <extendedProperty>
                <name>group.create.bases</name>
                <values>
                  <value>cn=groups_front,dc=us,dc=example,dc=com</value>
                </values>
            </extendedProperty>
        </serviceInstance>
```

```
 </serviceInstances>

  <jpsContexts default="default">

        <!-- IdStore service connect to  multiple ldaps ('oid'+'ad') through 
libOVD-->
        <jpsContext name="default">
<!-- use multiple ldaps ('oid'+'ad') through libOVD-->
            <serviceInstanceRef ref="idservice.virtualize"/>
            <!-- .....other services -->
        </jpsContext>

  </jpsContexts>

</jpsConfig>
```

Note that:

- the first service instance defines the provider for Oracle Internet Directory.
- the second service instance defines the provider for Microsoft Active Directory.
- the `virtualize` property of the service instance is set to `true`, enabling multi-LDAP queries.
- the `extendedProperty` elements enable you to set front-end parameters if desired to override default values.

  For more information, see "Front-End Parameters" in Section 8.3.1, "What is Configured?".

8.3.6 Defining Configuration in Java SE Environments

In a Java SE environment, all configurations are set in the `jps-config-jse.xml` file. According to your needs, you can:

1. Define a new identity store service instance.
2. Add the new service instance to the JPS context, replacing any previously defined IdentityStore instance.
3. Enable the virtualize flag in the identity store service; see Example 8–4.

For further information, see Section 25.3.2, "Configuring an LDAP Identity Store in Java SE Applications".

8.4 Querying the Identity Store Programmatically

To programmatically query the LDAP identity store, you use OPSS to obtain the JPS context; this acts like a bridge to obtain the store instance. Subsequently you use the User and Role API to query the store.

Example 8–5 Querying the LDAP Identity Store Programmatically

```
 try {
        //find the JPS context
        JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
        JpsContext ctx = ctxFactory.getContext();

        //find the JPS IdentityStore service instance
        //(assuming the backend is ldap type)
        LdapIdentityStore idstoreService =
(LdapIdentityStore)ctx.getServiceInstance(IdentityStoreService.class)

         //get the User/Role API's Idmstore instance
          oracle.security.idm.IdentityStore idmIdentityStore =
idstoreService.getIdmStore();

        //use the User/Role API to query id store
        //

//alternatively, instead of using IdentityStore, use the
//IdentityDirectory to access LDAP
oracle.igf.ids.IdentityDirectory ids = idstoreService.getIdentityStore();
// ref. chapter "Developing with the Identity Directory API"
// on how to use IdentityDirectory

 } catch (Exception e) {
  e.printStackTrace()
}
```

To see how to enable the '`virtualize`' property in the identity store service, refer to Example 8–4.

For additional information about using MBeans, see Section E.2, "Configuring OPSS Services with MBeans".

8.5 Configuring SSL for the Identity Store Service

Connections between the identity store and an LDAP server can be SSL-enabled. Both the Identity Directory API and the User and Role API can operate in a multi-LDAP identity store configuration (`virtualize = true`, which is the default setting). This section explains how to SSL-enable the connection from the identity store to the LDAP servers in virtualized environment.

This section contains the following topics:

- Connections from Oracle WebLogic Server to Identity Store
- Configuring SSL with Keystore Service
- Configuring SSL with JKS-Based Key Stores

8.5.1 Connections from Oracle WebLogic Server to Identity Store

When the connection to the identity store originates at a client residing in Oracle WebLogic Server, SSL configuration is handled by Oracle WebLogic Server. For details, see Section 9.2.2.

8.5.2 Configuring SSL with Keystore Service

When using multi-LDAP SSL with the Keystore Service, the store certificates are stored centrally across the domain. In the following scenarios, you configure SSL communication from the identity store to the LDAP servers using Keystore Service WLST commands. For more information about the Keystore Service and WLST commands, see Chapter 12.

For more information about configuring SSL with a JKS-based key store, see Section 8.5.3.

This section contains the following procedures:

- Configuring One-way SSL in a Multi-LDAP Scenario with Keystore Service

- Configuring Two-way SSL in a Multi-LDAP Scenario with Keystore Service
- Switching from JKS to KSS

8.5.2.1 Configuring One-way SSL in a Multi-LDAP Scenario with Keystore Service

To enable one-way SSL communication in a multi-LDAP identity store configuration:

1. Export the store certificate from the LDAP directory using the appropriate export command.
2. Use the `importKeyStoreCertificate` command to import the store certificate from the external LDAP server into the shared domain-wide truststore. For example:

   ```
   svc.importKeyStoreCertificate(appStripe='system', name='trust', password='',
   alias='alias', keypassword='', type='TrustedCertificate',filepath='absolute
   _file_path')
   ```

   where `appStripe='system'name='trust'` is the domain truststore.
3. Restart Oracle WebLogic Server.

8.5.2.2 Configuring Two-way SSL in a Multi-LDAP Scenario with Keystore Service

To enable two-way SSL communication in a multi-LDAP identity store configuration:

1. Export the store certificate from the LDAP directory using the appropriate export command.
2. Use the `importKeyStoreCertificate` command to import the certificate from external LDAP server into the shared domain wide truststore. For example:

   ```
   svc.importKeyStoreCertificate(appStripe='system', name='trust', password='',
   alias='alias', keypassword='', type='TrustedCertificate',filepath='absolute
   _file_path')
   ```

   where `appStripe='system'name='trust'` is the domain truststore.
3. Create the virtualized keystore in the Keystore Service using `createKeyStore` command with `appStripe='libovd', name='adapters'`. For example:

   ```
   svc.createKeyStore(appStripe='libovd',name='adapters',password='',permission=tr
   ue)
   ```

4. Run the `generateKeyPair` command to generate a new key pair signed by a CA. For example:

   ```
   svc.generateKeyPair(appStripe='libovd',name='adapters',password='',dn='cn=direc
   tory manager', keysize='1024',alias='alias',keypassword='')
   ```

5. Export this certificate to a file using the `exportKeyStoreCertificate` command. For example:

   ```
   svc.exportKeyStoreCertificate(appStripe='libovd',name='adapters',password='',al
   ias='key1', type='Certificate', filepath='/tmp/cert.txt')
   ```

6. Import the certificate generated in previous step to the backend LDAP server.
7. Change "`property name="enabled" value="true"`" to be "*`KSSKeyManager`*" in the provider.os_xml file under DOMAIN_HOME/config/fmwconfig/ovd/default directory.
8. Restart Oracle WebLogic Server.

8.5.2.3 Switching from JKS to KSS

The following procedure explains how to switch an SSL configuration in an LibOVD adapter using JKS to an SSL configuration using KSS.

To switch to a KSS LibOVD:

1. Import the backend LDAP certificates from existing LibOVD JKS store to the domain truststore using the command `importKeyStore`, as illustrated in the following sample:

   ```
   importKeyStore(appStripe='system', name='trust', password='welcome1',
      aliases='alias_names_from_JKS', keypasswords='',type='JKS',
      permission=true, filepath='DOMAIN_HOME/config/fmwconfig/ovd/<context_
      name>/keystores/adapters.jks')
   ```

2. Open the file `DOMAIN_HOME/config/fmwconfig/ovd/<context_name>/provider.os_xml` and make the following changes so that to enable KSS:

   - Look for "provider name="KSSTrustManager" and change "enabled" to "true".
   - Look for “provider name="FileKeyManager" and change "enabled" to "false".
   - Look for “provider name="FileTrustManager" and change "enabled" to "false".

3. If previously you were using 2-way SSL, then perform the following steps:

   1. Create the LibOVD keystore in KSS using `createKeyStore` command, as illustrated in the following sample:

      ```
      createKeyStore(appStripe='libovd',name='adapters',password='myPass',permiss
      ion=true)
      ```

   2. Import the previously generated key pair from `adapters.jks` to the LibOVD keystore using the command `importKeyStore`, as illustrated in the following sample:

      ```
      importKeyStore(appStripe='libovd', name='adapters', password='myPassw',
         aliases='alias_names_from_JKS', keypasswords='',type='JKS',
         permission=true, filepath='DOMAIN_
      HOME/config/fmwconfig/ovd/default/keystores/adapters.jks')
      ```

   3. Change "enabled" value to "true" for "KSSKeyManager" in Open the file `DOMAIN_HOME/config/fmwconfig/ovd/<context_name>/provider.os_xml`, look for KSSKeyManager, and replace "enabled" for "true".

4. Restart the WebLogic Server.

8.5.3 Configuring SSL with JKS-Based Key Stores

When using multi-LDAP SSL with a JKS-based key store, the store certificates are maintained in multiple locations. For example, in the WebLogic Server truststore when the WLS authenticator is configured and in the adapters.jks file.

Use Table 8–2 to determine the correct procedure to use to configure SSL when using a JKS-based key store.

Table 8–2 SSL With JKS-Based Key Store

Virtualize Flag	Using the User and Role API	Using the Identity Directory Service API
`virtualize=false` (that is, virtualize flag is not set)	Specify truststore using JSSE parameters, as explained in Section 9.2.2. For example: -Djavax.net.ssl.trustStore=*trust_store_path_name*	Use the adapters.jks file as shown in Section 8.5.3.1 and Section 8.5.3.2.
`virtualize=true` (that is, virtualize flag is set). The default setting is `virtualize=true`	Use the adapters.jks file as shown in Section 8.5.3.1 and Section 8.5.3.2.	Use the adapters.jks file as shown in Section 8.5.3.1 and Section 8.5.3.2.

8.5.3.1 Configuring One-way SSL in a Multi-LDAP Scenario with JKS-Based Keystore

To enable one-way SSL communication in a multi-LDAP identity store configuration:

Note: ORACLE_HOME must be set to oracle_common directory before running libovdconfig.sh script.

---

1. Create a keystore to contain the LDAP server certificate(s) for use by the service. You will must provide passwords for the WebLogic Admin Server and the keystore, respectively.

   Create the keystore using the script $MW_HOME/oracle_common/bin/libovdconfig.sh with the "-createKeystore" option:

   ```
   libovdconfig.sh -host wls_host -port wls_adminserver_port -userName
   wls_user_name -domainPath full_path_domain_home -createKeystore
   ```

   where:

   - `host` is the Oracle WebLogic Server host
   - `port` is the Oracle WebLogic Server Admin Server port
   - `username` is the Oracle WebLogic Server admin user name
   - `domainPath` is the complete path to the domain home

2. Export the certificate from the LDAP directory using the appropriate export command.

3. Import this certificate into the keystore you created in Step 1.

   Import the certificate to the keystore using the `keytool` command. The syntax is as follows, for a keystore named `adapters.jks`:

   ```
   $JAVA_HOME/bin/keytool -importcert
   -keystore $DOMAIN_HOME/config/fmwconfig/ovd/default/keystores/adapters.jks
   -storepass keystore_password_used_in_libovdconfig.sh
   -alias alias_name
   -file full_path_to_LDAPCert_file
   -noprompt
   ```

4. Restart Oracle WebLogic Server.

8.5.3.2 Configuring Two-way SSL in a Multi-LDAP Scenario with JKS-Based Keystore

To enable two-way SSL in a multi-LDAP identity store configuration:

1. Perform the procedure described in Section 8.5.3.1.
2. In the keystore that was created by Step 1 of Section 8.5.3.1, generate a new key pair, signed by a CA.
3. Export this certificate to a file.
4. Import the certificate into the server's keystore.

9 Configuring the OPSS Security Store

The OPSS security store is the repository of system and application-specific policies, credentials, keys, and audit metadata.

This chapter explains the features of the OPSS security store in the following sections:

For details about Java EE and WebLogic Security, see section Java EE and WebLogic Security in *Understanding Security for Oracle WebLogic Server*.

Note: When a WebLogic domain is setup to use policies based on the OPSS security store, JACC policies and the Java Security Manager become unavailable on all managed servers in that domain.

Important: All permission classes used in policies in the OPSS security store must be included in the class path, so the policy provider can load them when a service instance is initialized.

9.1 Introduction to the OPSS Security Store

The OPSS security store is the repository of system and application-specific policies, credentials, and keys. This centralization facilitates the administration and maintenance of policy, credential, and key data.

The OPSS security store can be file-, LDAP-, or DB-based depending on the choice of repository type, and it can be reassociated (that is, the repository type can be changed) from file-based to LDAP- or DB-based; from DB-based to LDAP- or DB-based; and from LDAP-based to LDAP- or DB-based. No other reassociation is supported. For details about the tools and procedures available to reassociate the OPSS security store, see sections Reassociating with Fusion Middleware Control and Reassociating with the Script reassociateSecurityStore. Out-of-the-box, the OPSS security store is DB-based for production WebLogic domains.

The security artifacts relevant to a Java EE application are typically packaged with the application and they can be migrated at deploy time to the OPSS security store. For details about the tools and procedures available to migrate to the OPSS security store, see sections Migrating with Fusion Middleware Control and Migrating with the Script migrateSecurityStore.

9.1.1 Multi-Server Environments

Production WebLogic domains with several server instances (administration and managed servers) on the same host or distributed across multiple machines, must use an OID- or an Oracle RDBMS-based OPSS security store.

Note: File-based providers are not supported in production environments.

For details about the properties you can set on policies and credentials, see sections Appendix F.2.2, "Policy Store Properties," and Appendix F.2.3, "Credential Store Properties."

9.2 Using an LDAP-Based OPSS Security Store

An LDAP-based OPSS security store is typically used in production environments. The only LDAP server supported is the Oracle Internet Directory (release 10.1.4.3 or later).

Note: Depending on the version, the following patches to Oracle Internet Directory are required:

- Patch to fix bug 9093298 in Oracle Internet Directory 10.1.4
- Patch to fix bug 8736355 in Oracle Internet Directory 11.1.x
- Patch to fix bug 8426457 in Oracle Internet Directory 11.1.x and 10.1.4.3
- Patch to fix bug 8351672 in Oracle Internet Directory 10.1.4.3
- Patch to fix bug 8417224 in Oracle Internet Directory 10.1.4.3
- Patch to fix bug 13782459 in Oracle Internet Directory 11.1.1.6.0

To apply a patch, proceed as follows:

1. Visit Oracle Automated Release Updates.
2. Click the **Patches** tab.
3. Enter the bug number in the **Request Number** box, and click **Search**.
4. Apply the patch.

To use a domain LDAP-based OPSS security store the domain administrator must configure it, as appropriate, using Oracle Enterprise Manager Fusion Middleware Control or WLST commands.

Important: OPSS does *not* support enabling referential integrity on Oracle Internet Directory servers. The server will not work as expected if referential integrity is enabled.

To disable a server's referential integrity, use Oracle Enterprise Manager Fusion Middleware Control as follows:

1. Select **Administration**, then **Shared Properties** from the Oracle Internet Directory menu, and then select **General**.
2. Select **Disabled** from the Enable referential Integrity list.

For a list of properties that can be specified in a service instance, see Appendix F.2.5, "Properties Common to All OID-Based Instances."

9.2.1 Prerequisites to Using an LDAP-Based Security Store

The only supported LDAP-based OPSS security store is Oracle Internet Directory. In order to ensure the proper access to the Oracle Internet Directory, you must set a node in the server directory as explained below.

Fusion Middleware Control automatically provides bootstrap credentials in the file `cwallet.sso` when you use that control to reassociate to an LDAP-based repository. To specify these required credentials manually, see section Section 24.6.7, "Specifying Bootstrap Credentials Manually."

Setting a Node in an Oracle Internet Directory Server

The following procedure is carried out by an Oracle Internet Directory administrator.

To set a node in an LDAP Oracle Internet Directory, proceed as follows:

1. Create an LDIF file (assumed `jpstestnode.ldif`, for illustration purpose) specifying the following DN and CN entries:

   ```
   dn: cn=jpsroot
   cn: jpsroot
   objectclass: top
   objectclass: OrclContainer
   ```

   The distinguished name of the root node (illustrated by the string *jpsroot* above) must be distinct from any other distinguished name. Some LDAP servers enforce case sensitivity by default. One root node can be shared by multiple WebLogic domains. It is not required that this node be created at the top level, as long as read and write access to the subtree is granted to the Oracle Internet Directory administrator.

2. Import this data into the LDAP server using the command `ldapadd`, as illustrated in the following example (there should be no line break in the command invocation):

   ```
   >ldapadd -h ldap_host -p ldap_port -D cn=orcladmin -w password -v -f
   jpstestnode.ldif
   ```

3. Verify that the node has been successfully inserted using the command `ldapsearch`, as illustrated in the following example (there should be no line break in the command invocation):

   ```
   >ldapsearch -h ldap_host -p ldap_port -D cn=orcladmin -w password -s base
   -b "cn=jpsroot" objectclass="orclContainer"
   ```

4. Run the utility `oidstats.sql` to generate database statistics for optimal database performance, as illustrated in the following example:

   ```
   >$ORACLE_HOME/ldap/admin/oidstats.sql
   ```

   The above utility must be run just once after the initial provisioning. For details about this utility, consult the *Oracle Fusion Middleware Reference for Oracle Identity Management*.

   To reassociate the OPSS security store, see Reassociating the OPSS Security Store.

9.2.2 Setting Up a One- Way SSL Connection to the LDAP

This section describes how to set up a one-way SSL channel between Oracle WebLogic server or a Java SE application and the LDAP Oracle Internet Directory. Such connection may be required, for example, when reassociating to an LDAP-based target store.

Prerequisite: Configuring the Oracle Internet Directory Server

To configure the Oracle Internet Directory server to listen in one-way SSL mode, see section Enabling SSL on Oracle Internet Directory Listeners in *Administering Oracle Fusion Middleware*.

Exporting Oracle Internet Directory's Certificate Authority (CA)

The use of `orapki` to create a certificate is needed *only if* the CA is unknown to the Oracle WebLogic server.

The following sample illustrates the use of this command to create the certificate `serverTrust.cert`:

```
>orapki wallet export -wallet CA -dn "CN=myCA" -cert serverTrust.cert
```

The above invocation prompts the user to enter the keystore password.

Before You Begin

Before configuring SSL, note that:

- The following procedures are required if the type of SSL being established is server-auth, and they are not required in any other case (no-auth or client-auth).
- If the flags specified in the procedures below are used in a multi-application environment, then the trust store must be shared by all those applications.

Setting Up the WebLogic Server in Case of a Java EE Application

The difference in the following procedures is because the identity store service and the OPSS security store service use different socket factories.

To establish a one-way SSL connection between the server and the identity store, proceed as follows (if applicable, the trust CA is assumed exported):

1. If the CA is known to the Oracle WebLogic server, skip this step; otherwise, use the utility `keytool` to import the Oracle Internet Directory's CA into the WebLogic trust store.

   The following invocation, which outputs the file `myKeys.jks`, illustrates the use of this command to import the file `serverTrust.cert`:

   ```
   >keytool -import -v -trustcacerts -alias trust -file serverTrust.cert -keystore
   myKeys.jks -storepass keyStorePassword
   ```

2. Modify the script (typically startWebLogic.sh) that starts the server to include a line like the following, and then restart the server:

   ```
   -Djavax.net.ssl.trustStore=<absolute path name to file myKeys.jks>
   ```

To establish a one-way SSL connection between the server and the OPSS security store, proceed as follows (if applicable, the trust CA is assumed exported):

1. Use the utility `keytool` to import trust CA to the trust key store, as illustrated in the following invocation:

   ```
   >keytool -import -v -trustcacerts -alias trust -file serverTrust.cert -keystore
   myKeys.jks -storepass keyStorePassword
   ```

2. Modify the script (typically startWebLogic.sh) that starts the server to include a line like the following, and then restart the server:

   ```
   -Dweblogic.security.SSL.trustedCAKeyStore=<absolute path name to file
   myKeys.jks>
   ```

3. If the OID server uses a wild card in the SSL certificate, then add the following line to the script that starts the WebLogic server:

   ```
   -Dweblogic.security.SSL.ignoreHostnameVerification=true
   ```

Setting Up the WebLogic Server in Case of a Java SE Application

The setting up in the case of Java SE applications is the identical for both the identity store and the OPSS security store.

1. If the CA is known to the Oracle WebLogic server, skip this step; otherwise, use the utility `keytool` to import the Oracle Internet Directory's CA into the WebLogic trust store.

   The following invocation, which outputs the file `myKeys.jks`, illustrates the use of this command to import the file `serverTrust.cert`:

   ```
   >keytool -import -v -trustcacerts -alias trust -file serverTrust.cert -keystore
   myKeys.jks -storepass keyStorePassword
   ```

2. Modify the script that starts the JMV to include a line like the following:

   ```
   -Djavax.net.ssl.trustStore=<absolute path name to file myKeys.jks>
   ```

9.3 Using a DB-Based OPSS Security Store

A DB-based security store is typically used in production environments. The only supported DB-based security store in production environments is Oracle RDBMS.

For versions supported see Oracle Fusion Middleware Supported System Configurations at `http://www.oracle.com/technetwork/middleware/ias/downloads/fusion-certification-100350.html`.

Note: In development environments only, the OPSS security store can be Oracle RDBMS Express, Oracle RDBMS Standard Edition, and Oracle RDBMS Standard Edition One.

To use a DB-based OPSS security store the domain administrator must configure it, as appropriate, using Oracle Enterprise Manager Fusion Middleware Control or WLST commands. In case any checks are needed before the server completes its initialization, see Section J.4.6, "Permission Failure Before Server Starts." The OPSS and audit schemas both support Edition-Based Redefinition (EBR); see end of Section 7.5.2, "Upgrading a DB-Based Security Store."

For a list of configuring properties, see Appendix F.2, "OPSS Configuration Properties."

This section contains the following topics:

- Prerequisites to Using a DB-Based Security Store
- Maintaining a DB-Based Security Store
- Setting Up an SSL Connection to the DB

9.3.1 Prerequisites to Using a DB-Based Security Store

To use a database repository for the OPSS security store, one must first use Oracle Fusion Middleware Repository Creation Utility (RCU) to create the required schema and to seed some initial data.

For details about this task, see appendix A, Understanding Repository Creation Utility Screens, in *Creating Schemas with the Repository Creation Utility*. To create the OPSS schema, select the following schemas:

- <prefix>_OPSS
- <prefix>_IAU
- <prefix>_IAU_APPEND
- <prefix>_IAU_VIEWER

9.3.2 Maintaining a DB-Based Security Store

This section describes a few tasks that an administrator can follow to maintain a DB-based security store, including changing the OPSS schema password.

A DB-based security store maintains a change log that should be periodically purged. To purge it, an administrator can use the provided SQL script `opss_purge_changelog.sql`, which will purge change logs older than 24 hours, or connect to the database and run SQL `delete` (with the appropriate arguments) as illustrated in the following lines:

```
SQL>delete from jps_changelog where createdate < (select(max(createdate) - 1) from
jps_changelog);
SQL>Commit;
```

If the OPSS management API performs slowly while accessing the DB-based security store, run the `DBMS_STATS` package to gather statistics about the physical storage of a DB table, index, of cluster. This information is stored in the data dictionary and used to optimize the execution plan for SQL statements accessing analyzed objects.

When loading large amount of data into a DB-based security store, such as when creating thousands of new application roles, it is recommended that `DBMS_STATS` be run within short periods and concurrently with the loading activity. Otherwise, when the loading activity is small, `DBMS_STATS` must be run just once and according to your needs.

The following sample illustrates the use of `DBMS_STATS`:

```
EXEC DBMS_STATS.GATHER_SCHEMA_STATS('DEV_OPSS', DBMS_STATS.AUTO_SAMPLE_SIZE,
no_invalidate=>FALSE);
```

where `DEV_OPSS` denotes the name of the DB schema created during the RCU setup. For details about the `DBMS_STATS` package, see the *Oracle Database Administrator's Guide*.

To run `DBMS_STATS` periodically, use a shell script or an SQL script, as described next.

The following sample script runs the command `DBMS_STATS` every 10 minutes:

```
#!/bin/sh
i=1
while [ $i -le 1000 ]
do
echo $i
sqlplus dev_opss/welcome1@inst1 @opssstats.sql
sleep 600
i=`expr $i + 1`
done
```

where `opssstats.sql` contains the following text:

```
EXEC DBMS_STATS.gather_schema_stats('DEV_OPSS',DBMS_STATS.AUTO_SAMPLE_SIZE,
no_invalidate=>FALSE);
QUIT;
```

The following sample SQL script also runs the command `DBMS_STATS` every 10 minutes:

```
variable jobno number;
BEGIN
DBMS_JOB.submit
(job => :jobno,
what =>
'DBMS_STATS.gather_schema_stats(''DEV_OPSS'',DBMS_STATS.AUTO_SAMPLE_SIZE,no_invali
date=>FALSE);',
interval => 'SYSDATE+(10/24/60)');
COMMIT;
END;
/
```

To stop the periodic invocation of `DBMS_STATS` by the above SQL script, first find out its job number by issuing the following commands:

```
sqlplus '/as sysdba'
SELECT job FROM dba_jobs WHERE schema_user = 'DEV_OPSS' AND what =
'DBMS_STATS.gather_schema_stats(''DEV_OPSS'',DBMS_STATS.AUTO_SAMPLE_SIZE,
no_invalidate=>FALSE);';
```

Then issue a command like the following, in which it is assumed that the query above returned the job number 31:

```
EXEC DBMS_JOB.remove(31);
```

To reset the OPSS schema password, proceed as follows:

1. Use the DB command ALTER USER to reset the password in the DB. Remember the new password entered, as it will be used in the next two steps.
2. Using the Oracle WebLogic Administration Console, update the password that the data source the domain uses to connect to the OPSS schema with the new

password. For details, see *Administering JDBC Data Sources for Oracle WebLogic Server*.Administering JDBC Data Sources for Oracle WebLogic Server

3. Using the WLST command `modifyBootStrapCredential`, update the bootstrap wallet with the new password. For details, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

9.3.3 Setting Up an SSL Connection to the DB

Establishing a one- or two-way SSL connection to a DB-Based OPSS security store is optional and explained in section Configuring SSL for the Database in *Administering Oracle Fusion Middleware*.

For additional information about SSL-related topics see the following document:

- *Oracle Database JDBC Developer's Guide*.

9.4 Configuring the OPSS Security Store

For examples of store configurations for Java SE applications, see Section 18.1, "Configuring the Security Store in Java SE Applications."

For examples of store configurations for Java EE applications, see Example 1.

For details about configuring other artifacts, see Configuring Services Providers with Fusion Middleware Control.

9.5 Reassociating the OPSS Security Store

Reassociating the OPSS security store is the process of relocating security artifacts from one repository to another one. The source can be file-, LDAP-, or DB-based; the target can be LDAP- or DB-based. The only type of LDAP target supported is Oracle Internet Directory; the only type of DB target supported is Oracle database.

Reassociation changes the repository while preserving the integrity of the data stored. All security artifacts are migrated from the existing security store to the new target store. This operation can take place at any time after the domain has been created, and it is carried out using either Fusion Middleware Control or `reassociateSecurityStore` as explained in the following sections:

- Reassociating with Fusion Middleware Control
- Reassociating with the Script reassociateSecurityStore

9.5.1 Reassociating with Fusion Middleware Control

Reassociation migrates the OPSS security store (policies, credentials, keys, and audit metadata) from one repository to another and reconfigures the appropriate security store providers. This section explains how to perform reassociation with Fusion Middleware Control pages.

For information about other uses of the **Security Provider Configuration** page, see Configuring Services Providers with Fusion Middleware Control.

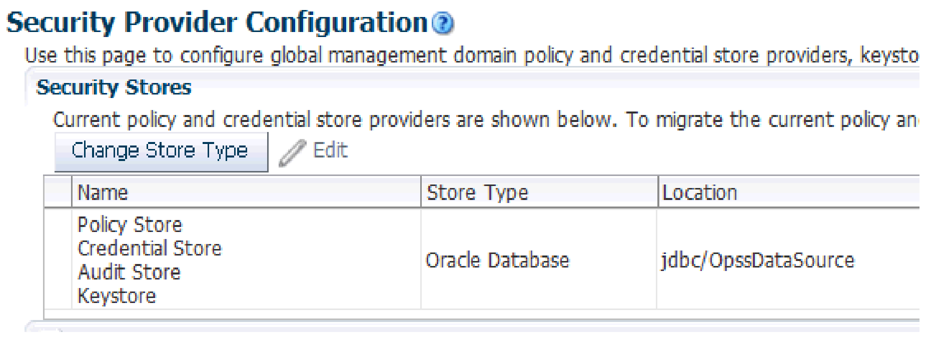
Important Points

- Before reassociating to a target LDAP store, ensure that your setup satisfies the Prerequisites to Using an LDAP-Based Security Store.

- Before reassociating to a target DB store, ensure that your setup satisfies the Prerequisites to Using a DB-Based Security Store.
- If reassociation requires a one-way SSL to a target LDAP, follow the instructions in Setting Up a One- Way SSL Connection to the LDAP *before* reassociating.
- After reassociating to an LDAP store, to secure access to the root node of the Oracle Internet Directory store, follow the instructions in Securing Access to Oracle Internet Directory Nodes.
- Reassociation updates the files `jps-config.xml` and `jps-config-jse.xml` with the appropriate new configuration.

To reassociate the OPSS security store with Fusion Middleware Control, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration**, to display the **Security Provider Configuration** page, partially illustrated in the following graphic:



Security Provider Configuration

Use this page to configure global management domain policy and credential store providers, keysto

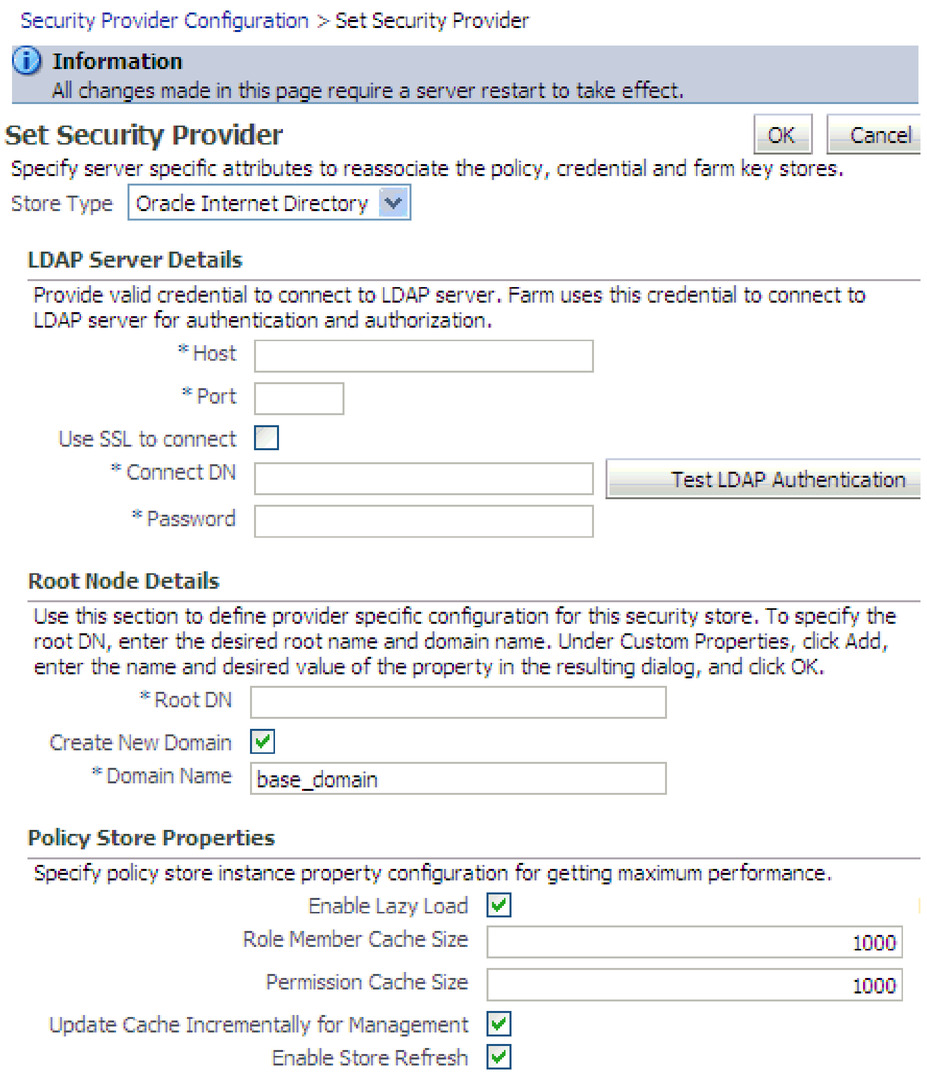
Security Stores

Current policy and credential store providers are shown below. To migrate the current policy an

Name	Store Type	Location
Policy Store Credential Store Audit Store Keystore	Oracle Database	jdbc/OpsDataSource

   The table in the area **Security Stores** shows the characteristics of the current provider configured in the domain.

2. Click the button **Change Association** to display the **Set Security Provider** page, and choose the **Store Type** from the pull-down list. The text displayed on this page depends on the store type selected. The following graphic partially illustrates this page when Oracle Internet Directory is selected.


Security Provider Configuration > Set Security Provider

Information
All changes made in this page require a server restart to take effect.

Set Security Provider

Specify server specific attributes to reassociate the policy, credential and farm key stores.

Store Type: Oracle Internet Directory

LDAP Server Details

Provide valid credential to connect to LDAP server. Farm uses this credential to connect to LDAP server for authentication and authorization.

* Host	
* Port	
Use SSL to connect	☐
* Connect DN	
* Password	

Root Node Details

Use this section to define provider specific configuration for this security store. To specify the root DN, enter the desired root name and domain name. Under Custom Properties, click Add, enter the name and desired value of the property in the resulting dialog, and click OK.

* Root DN	
Create New Domain	✓
* Domain Name	base_domain

Policy Store Properties

Specify policy store instance property configuration for getting maximum performance.

Enable Lazy Load	✓
Role Member Cache Size	1000
Permission Cache Size	1000
Update Cache Incrementally for Management	✓
Enable Store Refresh	✓

3. If you have selected Database, enter the name of the data source in the **Datasource Name** box. This should be the name of the JDBC data source entered when the data source was created. If needed, click **Select** to obtain a list of configured data source names.

4. If you have selected Oracle Internet Directory, in the **LDAP Server Details** area, specify details and connection information about the target LDAP server:

   1. Enter the host name and port number of your target Oracle Internet Directory LDAP server.

   2. Optionally, check the box **Use SSL to Connect** to establish an anonymous SSL transmission to the LDAP server.

      When checking this box, keep in mind the following points:

      The port of the target LDAP server must be configured to handle an anonymous SSL transmission; this port is distinct from the default (non-secure) LDAP server port.

      If the reassociation is to use a one-way SSL to a target LDAP store, be sure to follow the instructions in Setting Up a One- Way SSL Connection to the LDAP *before* completing this step. Among other things, that setup identifies the port to support a one-way SSL channel, and it is that port that should be specified in this step. Reassociation through a two-way SSL channel is not supported in this release.

Fusion Middleware Control modifies the file `weblogic.policy` by adding the necessary grant to support the anonymous SSL connection.

3. In the text box **Connect DN**, enter the full distinguished name, a string containing between 1 and 256 characters. For example, cn=orcladmin,dc=us,dc=oracle,dc=com.

4. In the box **Password**, enter the user password, also a string containing between 1 and 256 characters.

5. To verify that the connection to the LDAP server using the entered data works, click the button **Test LDAP Authentication**. If you run into any connection problem, see Section J.6.5, "Failure to Establish an Anonymous SSL Connection."

5. In the **Root Node Details** area, enter the root DN in the box **Root DN**, which identifies the top of the tree that contains the data in the LDAP repository. The **Domain Name** defaults to the name of the selected domain.

   To solve most common errors arising from the specifications in these two fields, see Section J.3.1, "Reassociation Failure."

6. Optionally, in the **Policy Store Properties** and **Credential Store Properties** areas, enter service instance properties, such as Enable Lazy Load and Role Member Cache Size.

   To add a new property: click **Add** to display the **Add New Property** dialog; in this dialog, enter strings for **Property Name** and **Value**; click **OK**. The added property-value pair is displayed in the table **Custom Properties**.

   These properties are typically used to initialize the instance when it is created.

   A property-value pair you enter modifies the domain configuration file `jps-config.xml` by adding a `<property>` element in the configuration of the LDAP service instance.

   To illustrate how a service instance is modified, suppose you enter the property name `foo` and value `bar`; then the configuration for the LDAP service instance changes to contain a `<property>` element as illustrated in the following excerpt:

```
<serviceInstance name="myNewLDAPprovider" provider="someProvider"
  ...
  <property name="foo" value="bar"/>
  ...
</serviceInstance>
```

7. When finished entering your data, click **OK** to return to the **Security Provider Configuration** page. The system displays a dialog notifying the status of the reassociation. The table in the **Security Stores** area is modified to reflect the provider you have specified.

8. Restart the application server. Changes do not take effect until it has been restarted.

Reassociation modifies the domain configuration file `$DOMAIN_HOME/config/fmwconfig/jps-config.xml`: it deletes any configuration for the old store provider, inserts a configuration for the new store provider, and moves the policy and credential information from the source to the destination store.

If the destination store is LDAP-based, the information is stored under the domain DN according to the following format:

```
cn=<domain_name>,cn=JpsContext,<JPS ROOT DN>
```

As long as the configuration of the installation relies upon the above domain DN, that node should not be deleted from the LDAP Server.

9.5.1.1 Securing Access to Oracle Internet Directory Nodes

The procedure explained in this section is optional and performed only to enhance the security to access an Oracle Internet Directory.

An access control list (ACL) is a list that specifies who can access information and what operations are allowed on the Oracle Internet Directory directory objects. The control list is specified at a node, and its restrictions apply to all entries in the subtree under that node.

You can use ACL to control the access to policy and credential data stored in an LDAP Oracle Internet Directory repository, and it is, typically, specified at the top, root node of the store.

To specify an ACL at a node in an Oracle Internet Directory repository, proceed as follows:

1. Create an LDIF file with a content that specifies the ACL:

   ```
   dn: <storeRootDN>
   changetype: modify
   add: orclACI
   access to entry by dn="<userDN>" (browse,add,delete) by * (none)
   access to attr=(*) by dn="<userDN>" (search,read,write,compare) by * (none)
   ```

   where storeRootDN stands for a node (typically the root node of the store), and userDN stands for the DN of the administrator data (the same userDN that was entered to perform reassociation).

2. Use the Oracle Internet Directory utility `ldapmodify` to apply these specifications to the Oracle Internet Directory.

Here is an example of an LDIF file specifying an ACL:

```
dn: cn=jpsRootNode
changetype: modify
add: orclACI
access to entry by dn="cn=myAdmin,cn=users,dc=us,dc=oracle,dc=com"
(browse,add,delete) by * ( none )
access to attr=(*) by dn="cn=myAdmin,cn=users,dc=us,dc=oracle,dc=com"
(search,read,write,compare) by * (none)
```

For more information about access control lists and the command `ldapmodify`, see chapter 18 in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*.

9.5.2 Reassociating with the Script reassociateSecurityStore

The OPSS store can be reassociated with the WLST command `reassociateSecurityStore`. For details, see Section 10.4.1, "reassociateSecurityStore."

9.6 Migrating the OPSS Security Store

Applications can specify their own policies, and these are stored as application policies in the appropriate stripe in the domain security store when the application is deployed to a server. All applications deployed or redeployed to a domain use a common security store, the domain security store. This store is logically partitioned in stripes, one for each application name; in a file-based store these partitions are specified in the file `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` under the element `<applications>`.

Migrating the OPSS security store is the process that relocates the policies, credentials, audit metadata, and keys from one repository to another one. The source can be file-, LDAP-, or DB-based; the target can be LDAP- or DB-based. The OPSS binaries and the target OPSS security store must have compatible versions; for details, see Section J.9.1, "Incompatible Versions of Binaries and Policy Store."

During application development, an application specifies its own policies, credentials, and audit components, and these can be migrated to the OPSS security store when the application is deployed with Fusion Middleware Control. Policies can also be migrated manually; in addition, each application component can specify the use of anonymous user and role, authenticated role, and JAAS mode.

The configuration of the OPSS security store is performed by an administrator.

These topics are explained in the following sections:

- Migrating with Fusion Middleware Control
- Migrating with the Script migrateSecurityStore

Note: Use the system property `jps.deployment.handler.disabled` to disable the migration of application policies and credentials for applications deployed on a WebLogic Server.

When this system property is set to TRUE, the migration of policies and credentials at deployment is disabled for *all* applications regardless of the particular application settings in the application file `weblogic-application.xml`.

9.6.1 Migrating with Fusion Middleware Control

Application policies are specified in the application file `jazn-data.xml` and can be migrated to the OPSS security store when the application is deployed to a server in the WebLogic environment with Fusion Middleware Control; they can also be removed from the OPSS security store when the application is undeployed or be updated when the application is redeployed.

All three operations, the migration, the removal, and the updating of application policies, can take place regardless of the type of policy repository, but they do require particular configurations.

For details, see procedure in Section 6.6.2, "Migrating Policies and Credentials."

9.6.2 Migrating with the Script migrateSecurityStore

Application-specific security artifacts, such as policies, system policies, and credentials, can be migrated from a source repository to a target repository using the WLST command `migrateSecurityStore`.

This script is offline, that is, it does not require a connection to a running server to operate; therefore, the configuration file passed to the argument `configFile` need not be an actual domain configuration file, but it can be assembled *just* to specify the source and destination repositories of the migration.

Note: Since the script `migrateSecurityStore` recreates GUIDs and takes a long time to migrate large volume of data, you may want to consider migrating stores with an alternate procedure that uses Oracle Internet Directory bulk operations. For details, see Section 7.6.2, "Backing Up and Recovering an OID-Based Security Store."

For platform-specific requirements to run a WLST command, see note in Section 10.4, "Managing Application Policies with WLST commands." For usage details, see Examples of Use.

To migrate *all* policies (system *and* application-specific, for all applications) on WebLogic use the script (first) or interactive (second) syntax (arguments are written in separate lines for clarity):

```
migrateSecurityStore.py -type policyStore
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-skip trueOrfalse]
                        [-overwrite trueOrfalse]

migrateSecurityStore(type="policyStore", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext" [,skip="trueOrfalse"]
[,overwrite="trueOrfalse"])
```

The meanings of the arguments (all required) are as follows:

- `configFile` specifies the location of a configuration file relative to the directory where the script is run. This configuration file should be specially assembled and must contain the following elements:
  - a context specifying the source store
  - a context specifying the destination store
  - a context specifying the bootstrap credentials

  The bootstrap context refers to the location of a `cwallet.sso` file where the keys needed to access the source and destination stores, and to decrypt and encrypt security data in them are expected to be.

  For information about extracting keys used by a domain, see exportEncryptionKey; for information about storing a key into a wallet, see importEncryptionKey.

  For information about creating a wallet, see section Common Wallet Operations in *Administering Oracle Fusion Middleware*.

- `src` specifies the name of a jps-context in the configuration file passed to the argument `configFile`. The case of the string passed must match the case of the context in the configuration file.

- `dst` specifies the name of another jps-context in the configuration file passed to the argument `configFile`. The case of the string passed must match the case of the context in the configuration file.
- `skip` specifies whether the migration should skip migrating incompatible artifacts or to terminate upon encountering an incompatible artifact in the source repository; set to true to skip migrating incompatible artifacts and not to terminate; set to false to terminate if an incompatible artifact is detected. Optional; if unspecified, it defaults to false.
- `overwrite` specifies whether to overwrite existing data in the destination store; set to true to overwrite existing destination data; set to false to not overwrite existing destination data. Optional; default is false.

The contexts passed in `src` and `dst` must be defined in the passed configuration file, must have distinct names, and the case of the passed contexts must match the case of the contexts in the configuration file. From these two contexts, the script determines the locations of the source and the target repositories involved in the migration.

To migrate *just* system policies on WebLogic, use the script (first) or interactive (second) syntaxes (arguments are written in separate lines for clarity):

```
migrateSecurityStore.py -type globalPolicies
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-overwrite trueOrfalse]

migrateSecurityStore(type="globalPolicies", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext" [,overwrite="trueOrfalse"])
```

The meanings of the arguments (all required) are identical to the previous case.

To migrate *just* application-specific policies on WebLogic, for one application, use the script (first) or interactive (second) syntaxes (arguments are written in separate lines for clarity):

```
migrateSecurityStore.py -type appPolicies
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        -srcApp srcAppName
                       [-dstApp dstAppName]
                       [-overWrite trueOrfalse]
                       [-migrateIdStoreMapping trueOrfalse]
                       [-mode laxOrstrict]
                       [-skip trueOrfalse]

migrateSecurityStore(type="appPolicies", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext", srcApp="srcAppName",
[dstApp="dstAppName"], [overWrite="trueOrfalse"],
[migrateIdStoreMapping="trueOrfalse"], [mode="strict"], skip="trueOrfalse")
```

The meanings of the arguments `configFile`, `src`, `dst` and `skip` are identical to the previous cases. The meaning of other five arguments is as follows:

- `srcApp` specifies the name of the source application, that is, the application whose policies are being migrated.
- `dstApp` specifies the name of the target application, that is, the application whose policies are being written. If unspecified, it defaults to the name of the source application.

- `migrateIdStoreMapping` specifies whether enterprise policies should be migrated. The default value is True. To filter out enterprise policies from the migration, that is, to migrate *just* application policies, set it to False.
- `overWrite` specifies whether a target policy matching a source policy should be overwritten by or merged with the source policy. Set to true to overwrite the target policy; set to false to merge matching policies. Optional. If unspecified, defaults to false.
- `mode` specifies whether the migration should stop and signal an error upon encountering a duplicate principal or a duplicate permission in an application policy. Either do not specify or set to lax to allow the migration to continue upon encountering duplicate items, to migrate just one of the duplicated items, and to log a warning to this effect.

If the input does not follow the syntax requirements above, the script execution fails and returns an error. In particular, the input must satisfy the following requisites: (a) the file `jps-config.xml` is found in the passed location; (b) the file `jps-config.xml` includes the passed jps-contexts; and (c) the source and the destination context names are distinct.

To migrate *all* credentials use either of the script (first) or interactive (second) syntaxes (arguments are written in separate lines for clarity):

```
migrateSecurityStore.py -type credStore
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-skip trueOrfalse]
                        [-overwrite trueOrfalse]

migrateSecurityStore(type="credStore", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext", [skip="trueOrfalse"],
[overwrite="trueOrfalse"])
```

The meanings of the arguments `configFile`, `src`, `dst`, `skip`, and `overwrite` are identical to the previous case.

To migrate *just* one credential map, use either of the script (first) or interactive (second) syntaxes (arguments are written in separate lines for clarity):

```
migrateSecurityStore.py -type folderCred
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-srcFolder map1]
                        [-dstFolder map2]
                        [-srcConfigFile alternConfigFileLocation]
                        [-overWrite trueOrFalse]
                        [-skip trueOrFalse]

migrateSecurityStore(type="folderCred", configFile="jpsConfigFileLocation",
src="srcJpsContext", dst="dstJpsContext", [srcFolder="map1"],
[dstFolder="map2"], [srcConfigFile="alternConfigFileLocation"],
[overWrite="trueOrFalse"], [skip="trueOrFalse"])
```

The meanings of the arguments `configFile`, `src`, `dst` and `skip` are identical to the previous case. The meanings of the last four arguments (all optional) are as follows:

- `srcFolder` specifies the name of the map containing the credentials to be migrated. Optional. If unspecified, the credential store is assumed to have only one map and the value of this argument defaults to the name of that map.
- `dstFolder` specifies the map where the source credentials are migrated. Optional; if unspecified, it defaults to the map passed to `srcFolder`.
- `srcConfigFile` specifies the location of an alternate configuration file, and it is used in the special case in which credentials are not configured in the file passed to `configFile`. Optional; if unspecified, it defaults to the value passed to `configFile`; if specified, the value passed to `configFile` is ignored.
- `overWrite` specifies whether a target credential matching a source credential should be overwritten by or merged with the source credential. Set to true to overwrite target credentials; set to false to merge matching credentials. Optional. If not specified, defaults to false. When set to false, if a matching is detected, the source credential is disregarded and a warning is logged.

9.6.2.1 Migrating Audit Metadata

You can use the script `migrateSecurityStore` to migrate audit metadata into a domain security store, or migrate audit metadata into an XML file.

See Section 6.6.3 for details.

9.6.2.2 Examples of Use

For complete examples illustrating the use of this script, see the following sections:

- Section 6.6.2.1, "Migrating Policies with migrateSecurityStore"
- Section 6.6.2.2, "Migrating Credentials with migrateSecurityStore"
- Section 6.6.4, "Migrating Keystore Service Artifacts"

9.7 Configuring Services Providers with Fusion Middleware Control

This section explains how to use Fusion Middleware Control to configure several service providers and the security store in the following sections:

- Configuring the Security Store
- Configuring the Identity Store Provider
- Configuring the Single Sign-On Provider
- Configuring the Trust Service Provider
- Configuring Properties and Property Sets

To configure policies, credentials, keys, and audit, see the following sections:

- Section 10.3, "Managing Policies with Fusion Middleware Control"
- Section 11.3, "Managing Credentials with Fusion Middleware Control"
- Section 12.2, "Keystore Management with the Keystore Service"
- Section 14.1, "Understanding Audit Administration Tasks"

9.7.1 Configuring the Security Store

To configure the security store, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration** to display the **Security Provider Configuration** page.
2. The **Security Stores** area of the page displays the characteristics of the policy store, the credential store, the audit store, and the keystore.
3. To change the store type, click the tab **Change Store Type**, so that the **Configure Security Stores** page is displayed.

   This page allows setting the details of the store, such as, type, server details, root node details, and properties.
4. If joining to an existing security store, *uncheck* the box **Create New Domain** in the **Root Node Details** area. Thenyou are required to enter the location and password for the encryption key that has been exported from the domain being joined.
5. To save your input, click **OK**; to revert to the original setting, clik **Cancel**.

9.7.2 Configuring the Identity Store Provider

To configure the parameters used by the User and Role API that interact with the identity store, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration** to display the **Security Provider Configuration** page.
2. Expand, if necessary, the area **Identity Store Provider**, and click **Configure** to display the page **Identity Store Configuration**.
3. Manage custom properties, as appropriate, using the buttons **Add** and **Delete**.
4. When finished, click **OK** to save your settings and to return to the **Security Provider Configuration** page.

9.7.3 Configuring the Single Sign-On Provider

To configure the SSO provider used by a domain, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration** to display the **Security Provider Configuration** page.
2. In that page, click the **Configure** in the Single Sign-On Provider area to display the **Single Sign-On Provider** page.
3. In that page, check the box **Configure Single Sign-On**, to allow entering data for the provider. All boxes are grayed out until this box is checked.
4. Select the **Store Type** from the pull-down list, and enter the corresponding data for the selected provider (the data required changes with the type selected).
5. Enter the **Login URL**, **Autologin URL**, and **Logout URL** in the text boxes.
6. Select the **Authentication Level** from the pull-down list.
7. Optionally, manage the provider **Custom Properties** using the buttons **Add**, **Edit**, and **Delete**, at the bottom of the page.
8. When finished, click **OK** to save the entered data.

9.7.4 Configuring the Trust Service Provider

To configure the trust service provider used by a domain, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration** to display the **Security Provider Configuration** page.
2. In that page, the **Trust Service Provider** area shows whether the service is configured; to configure the service, click the **Configure** to display the **Trust Service Provider** page.
3. That page contains the following areas:
   - **Provider**, where the name of the chosen provider is displayed.
   - **Trust Store**, where you select a stripe and a trust store, from the available ones.
   - **Keystore and Alias**, where you select a certificate by picking a stripe and a keystore, and an alias from the list of available trust aliases. The read-only entries at the bottom of the area (Issuer Name and Expiration Date) give information about the certificate selected.
   - **Trust Service Configuration**, where you enter the following values:
     – Clock Skew, the number of seconds that the clocks of the two systems (involved in the use of the trust service) may differ; the default is 60.
     – Token Validity Period, the number of seconds that the token is valid; the default is 1800.
     – Include Certificate, a boolean stating whether the certificate should be included in the token.
   - **Codesource Permission**, specifying the URL of the client code (that will access the trust store); no default value provided; this information translates into a codesource permission granted to the specified code.
4. Click **OK** to save the entered data and to return to the Security Provider Configuration page; the values you entered are saved in the file `$DOMAIN_HOME/config/fmwconfig/jps-config.xml`.

9.7.5 Configuring Properties and Property Sets

A property set is collection of properties typically used to define the properties of a service instance or generic properties of the domain.

For a list of OPSS configuration properties, see Appendix F.2, "OPSS Configuration Properties."

The elements `<property>` and `<properySet>` in the file `$DOMAIN_HOME/config/fmwconfig/jps-config.xml` are used to define property and property sets. Property sets are referenced by the element `<propertySetRef>`.

To define a property or a property set, proceed as follows:

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Security Provider Configuration** to display the **Security Provider Configuration** page.
2. Expand, if necessary, the area **Advanced Properties**, and click **Configure** to display the **Advanced Properties** page.

3. To enter a property, click **Add** in the **Properties** area to display the dialog **Add New Property**, and enter a property name and value. When finished, click **OK**. The entered property appears on the **Properties** table.
4. To enter a property set, click **Add Property Set** in the **Property Sets** area to display the dialog **Add Property Set**, and enter the property set name.
5. To enter a property in a property set, select a property set from the existing ones, then click **Add Property** to display the dialog **Add New Property**, and then enter a property name and value. The entered property is added to the list of properties in the selected property set.
6. Use the button **Delete** to remove a selected item from any table. When finished entering or editing properties and property sets, click **OK**.
7. Restart the Oracle WebLogic Server. Changes do not take effect until the server has been restarted.

The addition or deletion of property sets modifies the domain configuration file `$DOMAIN_HOME/config/fmwconfig/jps-config.xml`; the changes do not take effect until the server is restarted.

The elements `<property>` and `<propertySet>` added by the previous procedure are inserted directly under the element `<jpsConfig>`.

10 Managing the Policy Store

This chapter explains how a security administrator can manage policies using either Fusion Middleware Control, WLST commands, or Oracle Entitlements Server.

These topics are presented in the following sections:

- Determining the Domain Security Store Characteristics
- Managing the Policy Store
- Managing Policies with Fusion Middleware Control
- Managing Application Policies with WLST commands
- Caching and Refreshing the Cache
- Granting Policies to Anonymous and Authenticated Roles with WLST commands
- Application Stripe for Versioned Applications in WLST commands
- Managing Application Policies with Oracle Entitlements Server

10.1 Determining the Domain Security Store Characteristics

The offline WLST command `listSecurityStore` can be used by an administrator to determine the several attributes of the domain security store. For details about this command, see listSecurityStoreInfo.

10.2 Managing the Policy Store

Only a user with the appropriate permissions, such as the domain administrator, can access data in the policy store.

The following sections explain how an administrator can manage policies using either Fusion Middleware Control, WLST commands, or Oracle Entitlements Server. Typical operations include:

- Managing Policies with Fusion Middleware Control
- Managing Application Policies with WLST commands
- Managing Application Policies with Oracle Entitlements Server

To avoid unexpected authorization failures and to manage policies effectively, note the following important points:

Important Point 1: Before deleting a user, revoke all permissions, application roles, and enterprise groups that have been granted to the user. If you fail to remove all security artifacts referencing a user to be deleted, they are left dangling and, potentially, be inadvertently inherited if another user with the same name or uid is created at a later time.

Similar considerations apply to when a user name or uid is changed: all policies (grants, permissions, groups) referring to old data must be updated so that it works as expected with the changed data.

See Section J.5.4, "User Gets Unexpected Permissions."

Important Point 2: Policies use case sensitivity in names when they are applied. The best way to avoid possible authorization errors due to case in user or group names is to use the spelling of those names exactly as specified in the identity store.

It is therefore recommended that:

- When provisioning a policy, the administrator spell the names of users and groups used in the policy *exactly* as they are in the identity repository. This guarantees that queries into the policy store (involving a user or group name) work as expected.
- When entering a user name at run-time, the end-user enter a name that matches *exactly* the case of a name supplied in the identity repository. This guarantees that the user is authorized as expected.

See Section J.5.2, "Failure to Grant or Revoke Permissions - Case Mismatch."

Important Point 3: The name of a resource type, a resource, or an entitlement can contain printable characters only and it cannot start or end with a white space.

For other considerations regarding the use of characters in policies, in particular in role names, see Section J.10.5, "Characters in Policies."

Important Point 4: Authorization failures are not visible, by default, in the console. To have authorization failures sent to the console you must set the system variable jps.auth.debug as follows: `-Djps.auth.debug=true`

In particular, to have `JpsAuth.checkPermission` failures sent to the console, you must set the variable as above.

10.3 Managing Policies with Fusion Middleware Control

Fusion Middleware Control allows managing system and application policies in a WebLogic domain, regardless of the type of policy store provider used in the domain, as explained in the following sections:

- Managing Application Policies

- Managing Application Roles
- Managing System Policies

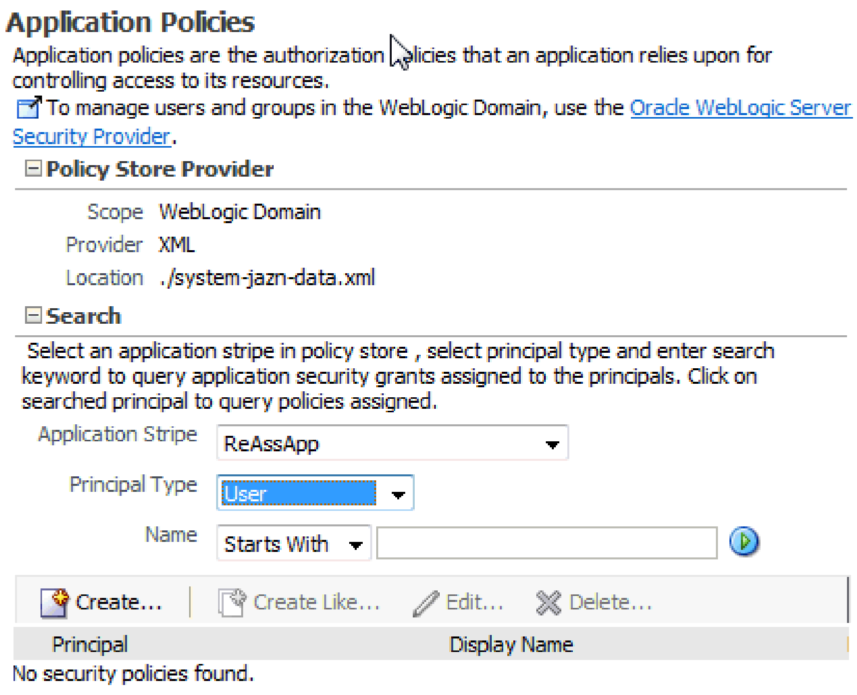
For procedures to manage other service providers, see Section 9.7, "Configuring Services Providers with Fusion Middleware Control."

10.3.1 Managing Application Policies

This section explains how to use Fusion Middleware Control to manage application policies.

> **Note:** If multiple applications are to share a permission and to prevent permission check failures, the corresponding permission class must be specified in the system class path.

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Application Policies** to display the **Application Policies** page, partially illustrated in the following graphic:



Application Policies

Application policies are the authorization policies that an application relies upon for controlling access to its resources.

To manage users and groups in the WebLogic Domain, use the Oracle WebLogic Server Security Provider.

Policy Store Provider

Scope	WebLogic Domain
Provider	XML
Location	./system-jazn-data.xml

Search

Select an application stripe in policy store , select principal type and enter search keyword to query application security grants assigned to the principals. Click on searched principal to query policies assigned.

Application Stripe	ReAssApp
Principal Type	User
Name	Starts With

Principal	Display Name

No security policies found.

   The area **Policy Store Provider** is read-only; when expanded, it displays the policy store provider currently in use in the domain.

2. To display policies in an application matching a given principal or permission, expand the **Search** area, select the application stripe to search, enter a string to match (a principal name, principal group, or application role), and click the blue button. The results of the search are displayed in the table at the bottom of the page.

3. To create an application policy for the selected application stripe, click **Create** to display the **Create Application Grant** page where you add principals and permissions for the grant being created.

    1. To add permissions, click **Add** in the **Permissions** area to display the **Add Permission** dialog.

        In the **Search** area of that dialog, first select **Permissions** or **Resource Types**; if Permissions was selected, then identify permissions matching a class or resource name, and determine the **Permission Class** and **Resource Name**; if Resource Types was selected, then identify the resource types matching a type name, and determine a type; then click **OK** to return to the **Create Application Grant** page. The permission you selected is displayed in the table in the **Permissions** area.

    2. To add principals, click the button **Add** in the **Grantee** area to display the dialog **Add Principal**.

        In the **Search** area of that dialog, select a **Type**, enter strings to match principal names and display names, and click the blue button; the result of the query is displayed in the **Searched Principals** table; then select one or more rows from that table, and click **OK** to return to the **Create Application Grant** page. The principals you selected are displayed in the table in the **Grantee** area

    3. At any point you can remove an item from the table in the Grantee area by selecting it and clicking the **Delete** button; similarly, you can modify an item from that table by selecting it and clicking the **Edit** button.

    4. When finished, click **OK** to return to the **Application Policies** page. The principal and permissions of the policy created are displayed in the table at the bottom of the page.

4. To create an application policy based on an existing one:

    1. Select an existing policy from the table.

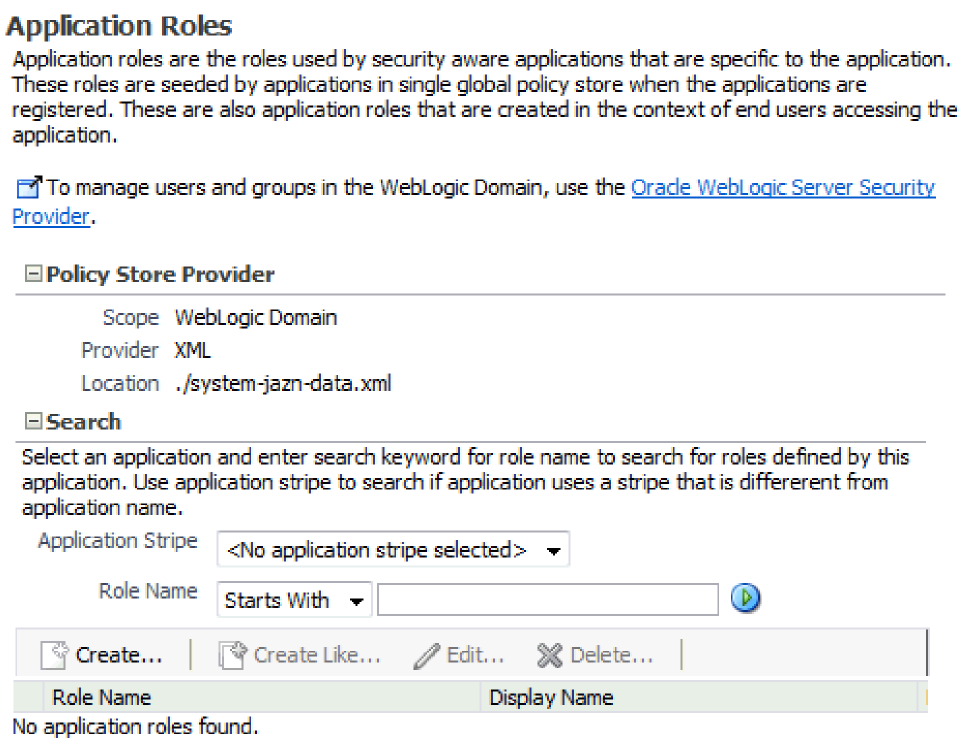
    2. Click **Create Like**, to display the **Create Application Grant Like** page. Notice that in this page the table of permissions is automatically filled in with the data extracted from the policy you selected.

    3. Modify those values, as appropriate, as explained in the substeps of step 3 above, and then click **OK**.

10.3.2 Managing Application Roles

This section explains how to use Fusion Middleware Control to manage application roles.

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Application Roles** to display the **Application Roles** page partially illustrated in the following graphic:



Application Roles

Application roles are the roles used by security aware applications that are specific to the application. These roles are seeded by applications in single global policy store when the applications are registered. These are also application roles that are created in the context of end users accessing the application.

To manage users and groups in the WebLogic Domain, use the Oracle WebLogic Server Security Provider.

Policy Store Provider

Scope	WebLogic Domain
Provider	XML
Location	./system-jazn-data.xml

Search

Select an application and enter search keyword for role name to search for roles defined by this application. Use application stripe to search if application uses a stripe that is differerent from application name.

Application Stripe	<No application stripe selected>
Role Name	Starts With

Role Name	Display Name

No application roles found.

The area **Policy Store Provider** is read-only; when expanded, it displays the policy store provider used in the domain.

2. To display roles in an application, expand the **Search** area, choose an application stripe to search, enter the data to match role names, and click the blue button. The results of the search are displayed in the table at the bottom of the page.

3. To create an application role, click **Create** to display the **Create Application Role** page. You need not enter data in all areas at once; for example, you could create a role by entering the role name and display name, save your data, and later on specify the members in it; similarly, you could enter data for role mapping at a later time.

   In the area **General**, specify the following attributes of the role being created:

   1. The name of the role, in the text box **Role Name**.
   2. Optionally, the name to display for the role, in the text box **Display Name**.
   3. Optionally, a description of the role, the text box **Description**.

   In the area **Members**, specify the users, groups, or other application roles, if any, into which the role being created is mapped.

   To add application roles to the application role being created:

   1. Click **Add**, to display the **Add Principal** dialog.
   2. In this dialog, select a **Type** (application role, group, or user), enter a string to match principal names, and click the blue button; the result of the search is displayed in the **Searched Principals** table; select one or more principals from that table.
   3. When finished, click **OK** to return to the **Create Application Role** page. The selected application roles are displayed in the table **Members**.

4. At any point you can remove an item from the Members table by selecting it and clicking the **Delete** button; similarly, you can modify an item from the table by selecting it and clicking the **Edit** button.

5. Click **OK** to effect the role creation (or updating) and to return to the **Application Roles** page. The role just created is displayed in the table at the bottom of that page.

6. To create an application role based on an existing one:

   1. Select an existing role from the table.

   2. Click **Create Like**, to display the **Create Application Role Like** page. Notice that in this page some data is automatically filled in with the data extracted from the role you selected.

   3. Modify the list of roles and users, as appropriate, and then click **OK**.

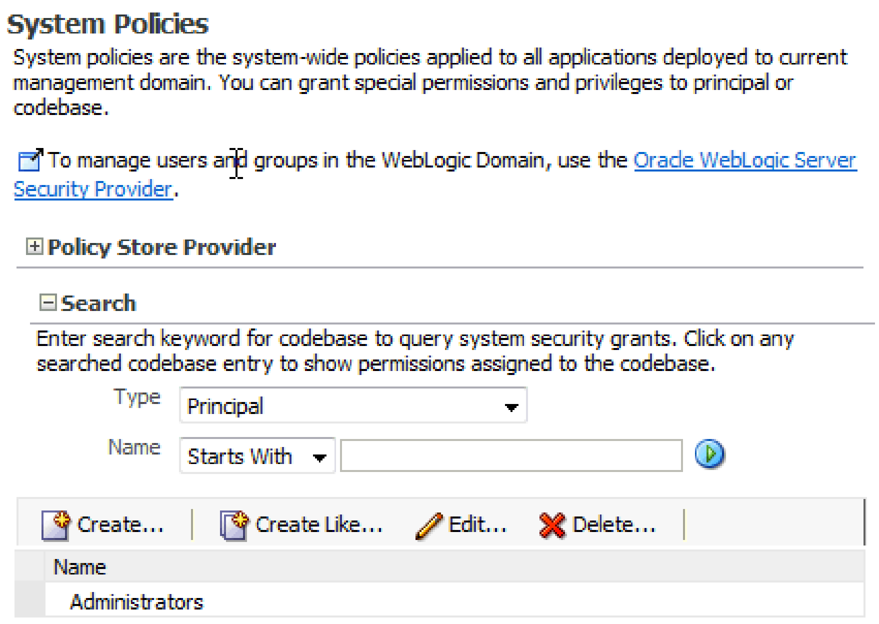
To understand how permissions are inherited in a role hierarchy, see Section 2.2.1, "Permission Inheritance and the Role Hierarchy."

10.3.3 Managing System Policies

This section explains how to use Fusion Middleware Control to manage system policies for an Oracle WebLogic Server domain.

The procedure below enables creating two types of system policies: principal policies and codebase policies. A principal policy grants permissions to a list of users or groups. A codebase policy grants permissions to a piece of code, typically a URL or a JAR file; for example, an application using the Credential Store Framework requires an appropriate codebase policy. Wildcard and patterns are not supported in codebase URLs.

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **System Policies** to display the **System Policies** page partially illustrated in the following graphic:



System Policies

System policies are the system-wide policies applied to all applications deployed to current management domain. You can grant special permissions and privileges to principal or codebase.

To manage users and groups in the WebLogic Domain, use the Oracle WebLogic Server Security Provider.

Policy Store Provider

Search

Enter search keyword for codebase to query system security grants. Click on any searched codebase entry to show permissions assigned to the codebase.

Type: Principal

Name: Starts With

Name
Administrators

The area **Policy Store Provider** is read-only; when expanded, it displays the policy store provider used in the domain.

2. To display system policies matching a given type, name, and permission, expand the **Search** area, enter the data to match, and click the blue button. The results of the search are displayed in the table at the bottom of the page.
3. At any point, you can edit the characteristics of a selected policy by clicking the **Edit** button, or remove it from the list by clicking the **Delete** button.

To create a system policy:

1. Click **Create** to display the **Create System Grant** page.
2. Select type of policy to create: Principal or Codebase. The UI differs slightly depending on the type chosen; the steps below assume the selection **Principal**.
3. To add permissions, click the button **Add** in the **Permissions** area to display the **Add Permission** dialog and choose a permission to add to the policy being created.
    1. Use the **Search** area to query permissions matching a type, principal name, or permission name. The result of the search is display in the table in the **Search** area.
    2. To choose the permission to add, select a permission from the table; when a permission is selected, its details are rendered in the read-only **Customize** area.
    3. Click **OK** to return to the **Create System Grant** page. The selected permission is added to the table **Permissions**.
4. At any point, you can select a permission from the Permissions table and use the button **Edit** to change the characteristics of the permission, or the button **Delete** to remove it.
5. Click **OK** to return to the System Policies page.
6. The table in the area **Permissions for Codebase** is read-only and it shows the resource name, actions, and permission class associated with a codebase system policy.

10.4 Managing Application Policies with WLST commands

An **online** command is a command that requires a connection to a running server. Unless otherwise stated, scripts listed in this section are online scripts and operate on the OPSS security store, regardless of its type(file, OID, or DB). There are a few scripts that are **offline**, that is, they do not require a server to be running to operate.

Read-only scripts can be performed only by users in the following WebLogic groups: Monitor, Operator, Configurator, or Admin. Read-write scripts can be performed only by users in the following WebLogic groups: Admin or Configurator. All WLST commands are available out-of-the-box with the installation of the Oracle WebLogic Server.

WLST commands can be run in interactive mode or in script mode. In interactive mode, you enter the command at a command-line prompt. In script mode, you write scripts in a text file (with a py file name extension) and run it without requiring input, much like the directives in a shell script.

WASAdmin scripts can be run in interactive mode only.

Note: Before invoking a WLST command you must run one of the scripts below to ensure that the required JARs are added to the class path.

```
>sh $ORACLE_HOME/common/bin/wlst.sh
```

To run an online command, you must connect to a WebLogic server as follows:

```
>java weblogic.WLST
>connect('userName', 'userPassword', 'url', 'adminServerName')
```

In regards to configuration files and JVMs, see note in Section 5.6, "Typical Security Practices with WLST Commands."

OPSS provides the following commands to administer application policies (all commands are **online**, unless otherwise stated):

- listAppStripes
- listCodeSourcePermissions
- createAppRole
- deleteAppRole
- grantAppRole
- revokeAppRole
- listAppRoles
- listAppRolesMembers
- grantPermission
- revokePermission
- listPermissions
- deleteAppPolicies
- createResourceType
- getResourceType
- deleteResourceType
- createResource
- deleteResource
- listResources
- listResourceActions
- createEntitlement
- getEntitlement
- deleteEntitlement
- addResourceToEntitlement
- revokeResourceFromEntitlement
- listEntitlements

- grantEntitlement
- revokeEntitlement
- listResourceTypes
- reassociateSecurityStore
- migrateSecurityStore. see Section 9.6.2, "Migrating with the Script migrateSecurityStore"

All class names specified in the above scripts must be fully qualified path names. The argument `appStripe` refers to the application stripe (typically, identical to the application name) and identifies the subset of policies pertaining to a particular application.

For important information about the authenticated and the anonymous roles and WLST commands, see Section 10.6, "Granting Policies to Anonymous and Authenticated Roles with WLST commands."

For the correct usage of the application stripe in versioned applications, see Section 10.7, "Application Stripe for Versioned Applications in WLST commands."

10.4.1 reassociateSecurityStore

The command `reassociateSecurityStore` migrates the OPSS security store from a source store to a target store and resets service configurations in the files `jps-config.xml` and `jps-config-jse.xml` to the target repository.

The source store can be a file-, OID-, or DB-based store. The target store can be a brand new store or an existing store in some other domain (see optional argument `join` below), and, in case of an existing store, one can specify whether to append the source store data to the joined target store (see optional argument `migrate` below).

The version of the source store must be equal or later than the version of the target store. If the versions are unequal (that is, the version of the source is later than the version of the target), then the command runs a compatibility check between the source and the target security artifacts; if the check fails on some artifacts, the command allows for skipping the migration of incompatible artifacts by setting the argument `skip` to true; if this argument is not set to true and incompatible artifacts are detected, the command aborts.

The command resets the bootstrap credentials (see arguments `admin` and `password` below); for an alternate way to reset bootstrap credentials, see the WLST commands `modifyBootStrapCredential` and `addBootStrapCredential`.

For recommendations involving reassociation, see Important Points. This command is supported in only the interactive mode.

Note: In a given domain and for a given major release (11g or 12c), the OPSS binaries version must be equal to or later than the OPSS security store version. See Section J.9.1, "Incompatible Versions of Binaries and Policy Store."

Interactive Mode Syntax

The command syntax varies slightly according to the type of the target store. When the target is an OID-based store, use the following syntax (arguments are displayed in separate lines for clarity only):

```
reassociateSecurityStore(domain="domainName",
servertype="OID",
ldapurl="hostAndPort",
jpsroot="cnSpecification",
admin="cnSpecification",
password="passWord",
[join="trueOrfalse"] [,keyFilePath="dirLoc", keyFilePassword="password"]]
[migrate="trueOrfalse"]
[skip="trueOrfalse"])
```

When the target is a DB-based store, use the following syntax (arguments are displayed in separate lines for clarity only):

```
reassociateSecurityStore(domain="domainName",
       servertype="DB_ORACLE",
       datasourcename="datasourceName",
       jpsroot="jpsRoot",
       jdbcurl="jdbcURL",
       jdbcdriver="jdbcDriverClass",
       dbUser="dbUserName",
       dbPassword="dbPassword",
       [admin="adminAccnt", password="passWord",]
       [,join="trueOrfalse"
          [,keyFilePath="dirLoc", keyFilePassword="password"]
          [,migrate="trueOrfalse" [,skip="trueOrfalse"]]]])
       [odbcdsn="odbcDsnSting"]
       [migrate="trueOrfalse"]
       [skip="trueOrfalse"])
```

The main points regarding the use of the arguments `join`, `migate`, and `skip` are next summarized:

- The argument `migrate` is relevant only when `join` is set to true; otherwise it is ignored. Therefore, if migration is desired then both `join` and `migrate` must be set to true.
- The arguments `keyFilePath` and `keyFilePassword` are required when `join` and `migrate` are both set to true.
- When `join` and `migrate` are both set to true then: if `skip` is set to true, the migration of incompatible artifacts with the target store are skipped; if `skip` is set to false (default value), then the command aborts if any incompatible artifacts are encountered. In this release, the skipping of only generic credentials is supported.

The meaning of the arguments is as follows:

- `domain`: on WebLogic, specifies the domain name where the target store is located.
- `admin` specifies, in case of an OID target, the administrator's user name on the target server, and the format is `cn=usrName`.

  In case of a DB target, it is required only when the DB has a protected data source (that is, protected with user/password); in this case, it specifies the user name that was set to protect the data source when the data source was created; that user and password must be present in the bootstrap credential store. If specified, then `password` must also be specified.
- `password` specifies the password associated with the user specified for the argument `admin`. It is required in case of an OID target.

In case of a DB target, it is required only when the DB has a protected data source; in this case, it specifies the password associated with the user specified in the argument `admin`. If specified, then `admin` must also be specified.

- `ldapurl` specifies the URI of the OID server. The format is `ldap//host:port`, if you are using the default port, or `ldaps://host:port`, if you are using an anonymous SSL or one-way SSL transmission. The secure port must be configured to handle the desired SSL connection mode, and must be distinct from the default (non-secure) port.
- `servertype` specifies the kind of the target store. The only valid types are `OID` or `DB_ORACLE`.
- `jpsroot` specifies the root node in the target OID repository under which all data is migrated. The format is `cn=nodeName`.
- `join` specifies whether the target store is an existing store in some other domain. Optional. Set to true to share an existing target store in some other domain; set to false otherwise. If unspecified, it defaults to false. The use of this argument allows multiple WebLogic domains to point to the same logical OPSS security store.

Note 1: When an OPSS security store is reassociated to another one with `join` set to true, OPSS encryption keys must be exported from one domain and imported into the other domain.

This note explains a way to accomplished this task; for an alternate way, see note in the description of argument `keyFilePath`.

Specifically, assume that Domain1 has a security store and Domain2 has reassociated to Domain1's security store using `join` set to true; then proceed as follows:

1. Use the WLST command `exportEncryptionKey` to extract the key from Domain1 into the file `ewallet.p12`; the value of the argument `keyFilePassword` passed must be used later when importing that key into the second domain.
2. Use the WLST command `importEncryptionKey` to import the extracted `ewallet.p12` into Domain2; the value of the argument `keyFilePassword` must be identical to the one used when the file `ewallet.p12` was generated.
3. Restart Domain2's server.

For details about the commands to import or export keys, see exportEncrytionKey and importEncryptionKey.

- `migrate` is meaningful only if `join` is set to true, otherwise ignored; specifies whether the data in the source store should be appended to the joined store; set to true to append source data to the target store; set to false to join to the target store without any appending source data. Optional. If unspecified, defaults to false.
- `skip` is meaningful only if both `join` and `migrate` are set to true, otherwise it is ignored; specificies whether the migration of incompatible artifacts should be skipped; set to true to skip appending incompatible artifacts to the target store and to not abort the command; set to false to abort the command upon encountering an incompatible artifact in the source store. Optional. If unspecified, it defaults to false.
- `datasourcename` specifies the JNDI name of the JDBC data source; this should be identical to the value of the JNDI name data source entered when the data source was created.

- `keyFilePath` specifies the directory where the file `ewallet.p12` for the target domain is located; the content of this file is encrypted and secured by the value passed to `keyFilePassword`. Required only if `join` is set to true.

> **Note 2:** When an OPSS security store is reassociated to another one with `join` set to true, OPSS encryption keys must be exported from one domain and imported into the other domain.
>
> This note explains a way to accomplished this task; for an alternate way, see note in the description of the argument `join`.
>
> When using the arguments `keyFilePath` and `keyFilePassword`, the export-import of keys is automatically done by the command `reassociateSecurityStore`.
>
> Specifically, assume that Domain1 has a security store and Domain2 is to reassociate to Domain1's security store using `join` set to true and the keyFile arguments; then proceed as follows:
>
> 1. Run the command `reassociateSecurityStore` with the appropriate argument values.
> 2. Restart Domain2's server.

- `keyFilePassword` specifies the password to secure the file `ewallet.p12`. Required only if `join` is set to true.
- `jdbcurl` specifies the jdbc URL use by a Java SE application to connect to the database. Applies only to Java SE applications. Required. Must be used with arguments `jdbcdriver`, `dbUser`, and `dbPassword`.
- `jdbcdriver` specifies the class of the jdbc driver used to connect to the database. Required. Must be used with argument `jdbcurl`.
- `dbUser` specifies the database user in the credential store to be added to the bootstrap credentials. Required. Must be used with argument `jdbcurl` .
- `dbPassword` specifies specifies the password of the user specified by `dbUser`. Required. Must be used with argument `jdbcurl`.
- `odbcdsn` specifies the odbc DSN name used by the C CSF API. Applies only to C programs.

Examples of Use

Here is an example invocation to reassociate to a DB-based security store:

```
reassociateSecurityStore(domain="farm", servertype="DB_ORACLE",
jpsroot="cn=jpsroot", datasourcename="jdbc/opssds",
jdbcurl="jdbc:oracle:thin:@myhost.oracle.com:5555:testdb",
dbUser="test_opss", dbPassword="mypass",
jdbcdriver="oracle.jdbc.xa.client.OracleXADataSource")
```

To share the OPSS security store in `otherDomain` *without* migrating the contents of the source security store, invoke the command as in the following sample:

```
reassociateSecurityStore(domain="otherDomain", servertype="DB_ORACLE",
jpsroot="cn=jpsroot", datasourcename="jdbc/opssds",
jdbcurl="jdbc:oracle:thin:@myhost.oracle.com:5555:testdb",dbUser="test_opss",
dbPassword="mypass", jdbcdriver="oracle.jdbc.xa.client.OracleXADataSource",
join="true", keyFilePath="/tmp/myFileDirectory", keyFilePassword="password")
```

To share the OPSS security store in `otherDomain` and *to migrate* the contents of the source security store to the target DB-based security store skipping over incompatible artifacts, invoke the command as in the following sample:

```
reassociateSecurityStore(domain="otherDomain", servertype="DB_ORACLE",
jpsroot="cn=jpsroot", datasourcename="jdbc/opssds",
jdbcurl="jdbc:oracle:thin:@myhost.oracle.com:5555:testdb",dbUser="test_opss",
dbPassword="mypass", jdbcdriver="oracle.jdbc.xa.client.OracleXADataSource",
join="true", migrate="true", skip="true",
keyFilePath="/tmp/myFileDirectory", keyFilePassword="password")
```

10.5 Caching and Refreshing the Cache

This topic applies to OID- and DB-based OPSS security stores only. In case of a file-based store, the cache is updated after a few seconds (file-based stores are not recommended in production systems.)

OPSS optimizes the authorization process by caching security artifacts. When an application policy (or some other security artifact) is modified, the change becomes effective at different times depending on where the application and the tool used to modified the artifact are running:

- If both the application and the tool are running on the same host and in the same domain, the change becomes effective immediately.
- Otherwise, if the application and the tool are running on different hosts or in different domains, the change becomes effective *after* the store cache is refreshed. The frequency of the cache refresh is determined by the value of the property `oracle.security.jps.ldap.policystore.refresh.interval`. The default value is 10 minutes.

  Within a domain, any changes effected via a WLST command or Fusion Middleware Control are first accounted on the administration server only; those changes are pushed to all managed servers in the domain *only when* the server is restarted.

10.5.1 An Example

The following use case illustrates the authorization behavior in four scenarios when (from a different domain or host) Oracle Entitlements Server is used to modify security artifacts, and the property `oracle.security.jps.ldap.policystore.refresh.interval` is set to 10 minutes.

This case assumes that:

- A user is member of an enterprise role.
- That enterprise role is included as a member of an application role.
- The application role is granted a permission that governs some application functionality.

Under the above assumptions, we now examine the authorization result in the following four scenarios.

Scenario A

1. The user logs in to the application.
2. The user accesses the functionality secured by the application role.

3. From another host (or domain), Oracle Entitlements Server is used to remove the enterprise role from the application role.
4. The user logs out from the application, and *immediately* logs back in.
5. The user is still able to access the functionality secured by the application role.

The reason for this outcome is that the policy cache has not yet been refreshed with the change introduced in step 3 above.

Scenario B

1. The user logs in to the application.
2. The user accesses the functionality secured by the application role.
3. From another host (or domain), Oracle Entitlements Server is used to remove the enterprise role from the application role.
4. The user logs out from the application, and logs back in *after 10 minutes*.
5. The user is not able to access the functionality secured by the application role.

The reason for this outcome is that the policy cache has been refreshed with the change introduced in step 3 above.

Scenario C

1. The user logs in to the application.
2. The user accesses the functionality secured by the application role.
3. From another host (or domain), Oracle Entitlements Server is used to remove the enterprise role from the application role.
4. The user does not log out and remains able to access the functionality secured by the application role *within 10 minutes*.

The reason for this outcome is that the policy cache has not yet been refreshed with the change introduced in step 3 above.

Scenario D

1. The user logs in to the application.
2. The user accesses the functionality secured by the application role.
3. From another host (or domain), Oracle Entitlements Server is used to remove the enterprise role from the application role.
4. The user does not log out, waits *more than 10 minutes*, and then attempts to access the functionality secured by the application role: the access is denied.

The reason for this outcome is that the policy cache has been refreshed with the change introduced in step 3 above.

10.6 Granting Policies to Anonymous and Authenticated Roles with WLST commands

Several WLST commands require the specification of the principal name and the principal class for a role involved in the operation.

For example, the following invocation adds a principal to the role with application stripe `myApp` and name `myAppRole`:

```
grantAppRole.py -appStripe myApp -appRoleName myAppRole
```

```
            -principalClass myPrincipalClass -principalName myPrincipal
```

When in such scripts the principal refers to the authenticated role or the anonymous role, the principal names and principal classes are fixed and *must* be one of the following pairs:

- Authenticated role
  - Name: `authenticated-role`
  - Class: `oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl`
- Anonymous role
  - Name: `anonymous-role`
  - Class: `oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl`

The list of WLST commands that required the above principal name and class specification are the following:

- `grantAppRole`
- `revokeAppRole`
- `grantPermission`
- `revokePermission`
- `listPermissions`

10.7 Application Stripe for Versioned Applications in WLST commands

Several WLST commands require the specification of an application stripe. If the application is not versioned, the application stripe defaults to the application name. Otherwise, if the application is versioned, the application name and the application stripe are not identical.

For example, the name of a versioned application with name `myApp` and version 1 is displayed `myApp(v1.0)` in Fusion Middleware Control pages, but the application stripe of this application is `myApp#v1.0`.

In general, an application with display name `appName(vers)` has application stripe `appName#vers`. It is this last string that should be passed as the application stripe in WLST commands, as illustrated in the following invocation:

```
>listAppRoles myApp#v1.0
```

The list of WLST commands that can use stripe specification are the following:

- `createAppRole`
- `deleteAppRole`
- `grantAppRole`
- `revokeAppRole`
- `listAppRoles`
- `listAppRoleMembers`

- `grantPermission`
- `revokePermission`
- `listPermissions`
- `deleteAppPolicies`

10.8 Managing Application Policies with Oracle Entitlements Server

Oracle Entitlements Server allows managing and searching application policies and other security artifacts in a WebLogic domainOracle Internet Directory.

For details, see the following topics in *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*:

- Querying Security Artifacts
- Managing Policies and Roles

11 Managing the Credential Store

A credential can hold user names, passwords, and tickets; credentials can be encrypted. Credentials are used during authentication, when principals are populated in subjects, and, further, during authorization, when determining what actions the subject can perform.

Oracle Platform Security Services includes the Credential Store Framework (CSF), a set of APIs that applications can use to create, read, update, and manage credentials securely. A typical use of the credential store is to store user names and passwords to access some external system, such as a database or an LDAP-based repository.

This chapter is divided into the following sections:

11.1 Credential Types

OPSS supports the following types of credentials according to the data they contain:

- A *password* credential encapsulates a user name and a password.
- A *generic* credential encapsulates any customized data or arbitrary token, such as a symmetric key.

A credential is uniquely identified by a map name and a key name. A map can hold several keys and, typically, the map name corresponds with the name of an application; all credentials with the same map name define a logical group of credentials, such as the credentials used by the application. The pair of map and key names must be unique for all entries in a credential store.

There is no limit to the number or kind of characters you can set in a password, except that it must be non-empty and non-null.

Oracle Wallet is the default file-based credential store, and it can store X.509 certificates; production environments typically use either an Oracle Internet Directory LDAP-based or a DB-based credential store.

11.2 Encrypting Credentials

OPSS supports storing encrypted data in file- and LDAP-based credential stores. OPSS uses an encryption key to encrypt and decrypt data when it is read from or written to the credential store. This key has domain scope, that is, there is exactly on (current) key per domain. To enable the encryption of credentials in a file- or LDAP-based store, set the following property in the credential store service instance of the file `jps-config.xml`:

```
<property name="encrypt" value="true" />
```

In case of DB-based credential stores, data is always encrypted using a client-side key.

The Domain Encryption Key

When the property `encrypt` is set to true, OPSS uses an encryption key to encrypt new credentials entered in the credential store. This encryption key is a 128-bit AES key randomly generated when the domain is started for the very first time and is valid for the entire domain; eventually, the domain encryption key may require being rolled over periodically; rolling over a key generates a new key and archives the previous one; archived keys are used to decrypt old data, and the new key is used to encrypt and decrypt new data.

When a new domain encryption key is produced, data in the credential store is not immediately re-encrypted with the new key. Instead, data is encrypted (with the new key) only when it is written. This implies that to get all data to use the same encryption key, all credentials must be read and written.

Domains Sharing a Credential Store

If two or more domains share a credential store and encryption is enabled in that store, then each of those domains must use the same encryption key. To ensure this, OPSS provides offline scripts to export, import, and restore keys in the domain bootstrap wallet, so that an encryption key generated in one domain can be carried over to all other domains sharing the credential store. For details about these commands, see Managing Credentials with WLST commands.

The following scenarios illustrate how to set credential encryption in a cluster of two domains, Domain1 and Domain2. (In case of more than two domains, treat each additional domain as Domain2 in the illustration below.)

Note: The following scenarios assume that the credential store is LDAP-based, but the use of `importEncryptionKey` and `exportEncryptionKey` to import and export keys across domains applies also to DB-based credential stores (in which data is always encrypted).

First Scenario

Assume that Domain1 has reassociated to an LDAP-based credential store, and Domain2 has *not yet joined* to that store. Then, to enable credential encryption on that store, proceed as follows:

1. Set the property `encrypt` to true in Domain1's `jps-config.xml` file and restart the domain.
2. Use the WLST command `exportEncryptionKey` to extract the key from Domain1 into the file `ewallet.p12`; note that the value of the argument `keyFilePassword` passed to the script must be used later when importing that key into another domain.
3. Set the property `encrypt` to true in Domain2's `jps-config.xml` file.

At this point you can complete the procedure in one of two ways; both of them use the WLST command `reassociateSecurityStore`, but with different syntaxes. For details about this script, see Section 10.4.1, "reassociateSecurityStore."

The first approach is as follows:

1. Use the WLST command `reassociateSecurityStore` to reassociate Domain2's credential store to that used by Domain1; use the argument `join` and *do not use* the arguments `keyFilePassword` and `keyFilePath`.
2. Use the WLST command `importEncryptionKey` to import the extracted `ewallet.p12` into Domain2; note that the value of the argument `keyFilePassword` must be identical to the one used when the file `ewallet.p12` was generated.
3. Restart Domain2's server.

The second approach is as follows:

1. Use the WLST command `reassociateSecurityStore` to reassociate Domain2's credential store to that used by Domain1; use the arguments `join`, `keyFilePassword`, and `keyFilePath`.
2. Restart Domain2's server.

Second Scenario

Assume that Domain1 has reassociated to an LDAP-based credential store and Domain2 has *already joined* to that store. Then, to enable credential encryption on that store, proceed as follows:

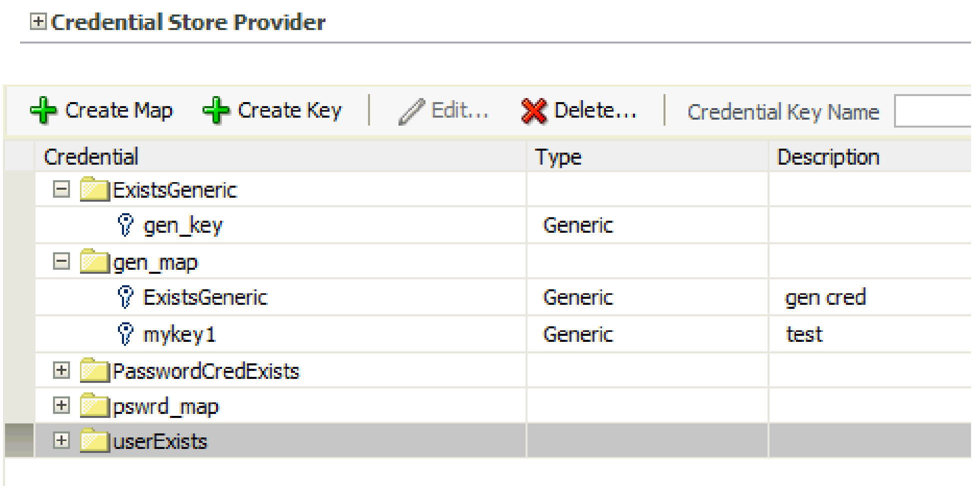
1. Set the property `encrypt` to true in Domain1's `jps-config.xml` file and restart the domain.
2. Use the WLST command `exportEncryptionKey` to extract the key from Domain1 into the file `ewallet.p12`; note that the value of the argument `keyFilePassword` passed to the script must be used later when importing that key into another domain.
3. Set the property encrypt to true in Domain2's `jps-config.xml` file.
4. Use the WLST command `importEncryptionKey` to write the extracted `ewallet.p12` into Domain2; note that the value of the argument `keyFilePassword` must be identical to the one used when the file `ewallet.p12` was generated.
5. Restart Domain2's server.

Important Note: In case of multiple domains sharing a credential store in which encryption has been enabled, every time a roll-over key is generated in one of those domains, the administrator *must* import that key to each of the other domains in the cluster using the WLST commands `exportEncryptionKey` and `importEncryptionKey`.

11.3 Managing Credentials with Fusion Middleware Control

The following procedure explains how to use Fusion Middleware Control to manage credentials. For procedures to manage other service providers, see Section 9.7, "Configuring Services Providers with Fusion Middleware Control."

1. Log in to Fusion Middleware Control and navigate to *Domain* > **Security** > **Credentials** to display the **Credentials** page partially illustrated in the following graphic:



Credential Store Provider

Credential	Type	Description
ExistsGeneric		
gen_key	Generic	
gen_map		
ExistsGeneric	Generic	gen cred
mykey1	Generic	test
PasswordCredExists		
pswrd_map		
userExists		

The area **Credential Store Provider** is read-only; when expanded, it displays the credential store provider currently in use in the domain.

2. To display credentials matching a given key name, enter the string to match in the **Credential Key Name** box, and then click the blue button. The result of the search is displayed in the table at the bottom of the page.

3. At any point, you can remove an item by selecting it and clicking the **Delete** button; similarly, you can modify an item from the table by selecting it and clicking **Edit** button. Note that deleting a credential map, deletes all keys in it.

To create a credential map:

1. Click **Create Map** to display the **Create Map** dialog.

2. In this dialog, enter the name of the map for the credential being created.

3. Click **OK** to return to the **Credentials** page. The new credential map name is displayed with a map icon in the table.

To add a key to a credential map:

1. Click **Create Key** to display the **Create Key** dialog.

2. In this dialog, select a map from the menu **Select Map** for the key being created, enter a key in the text **Key** box, and select a type (Password or Generic) from the pull-down menu **Type**. The dialog display changes according the type selected.

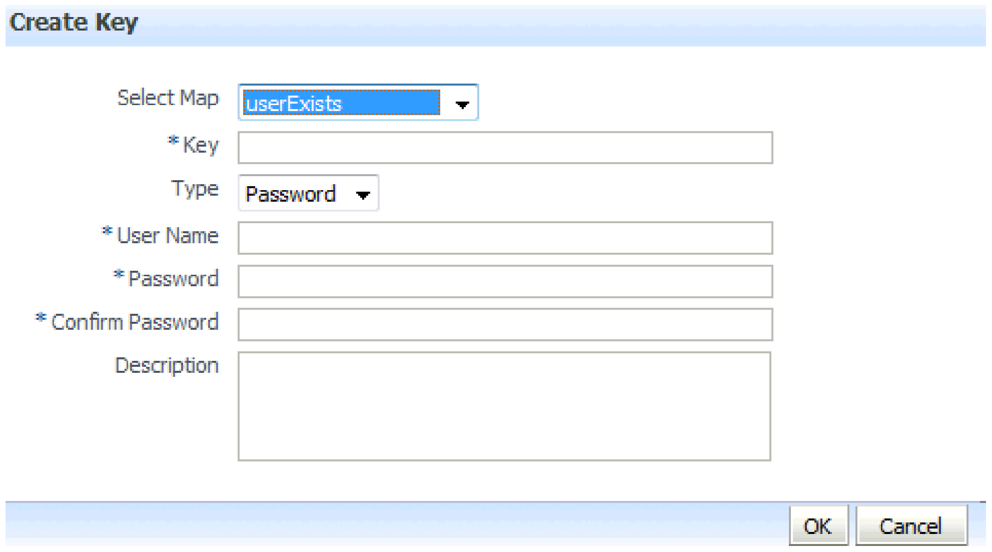
   If Password was selected, enter the required fields (Key, User Name, Password, Confirm Passwords).

   If Generic was selected, enter the required field Key and the credential information either as text (select **Enter as Text** radio button), or as a list of key-value pairs (select **Enter Map of Property Name and Value Pairs** radio button); to add a key-value pair, click **Add Row**, and then enter the Property Name, Value, and Confirm Value in the added arrow.

   Figure 11–1 illustrates the dialog used to create a password key.

3. Click **OK** to return to the **Credentials** page. The new key is displayed under the map icon corresponding to the map you selected.

Figure 11–1 The Create Key Dialog


Create Key

Select Map	userExists
* Key	
Type	Password
* User Name	
* Password	
* Confirm Password	
Description	

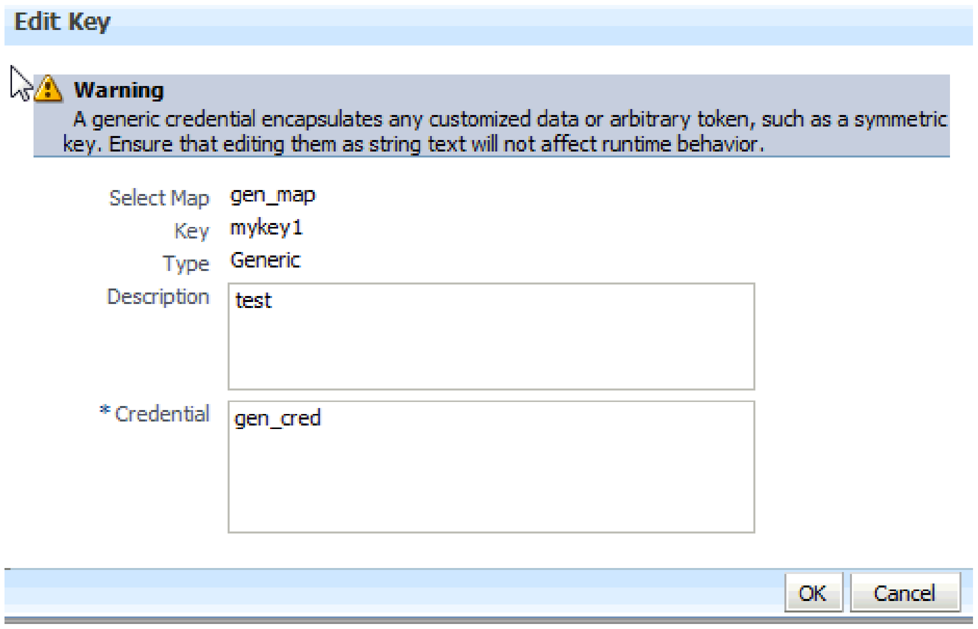
To edit a key:

1. Select a key from the table.
2. Click **Edit** to bring up the **Edit Key** dialog.
3. In that dialog, modify the key data as appropriate. In case of editing a generic key, use the red X next to a row to delete the corresponding property-value pair.

   Figure 11–2 illustrates the dialog used to edit a generic key.
4. Click **OK** to save your changes and return to the **Credentials** page.

For specific considerations that apply to ADF applications only, see section How to Edit Credentials Deployed with the Application in *Administering Oracle ADF Applications*.

Figure 11–2 The Create Key Dialog



Warning
A generic credential encapsulates any customized data or arbitrary token, such as a symmetric key. Ensure that editing them as string text will not affect runtime behavior.

Select Map	gen_map
Key	mykey1
Type	Generic
Description	test
* Credential	gen_cred

11.4 Managing Credentials with WLST commands

An **online** script is a script that requires a connection to a running server. Unless otherwise stated, scripts listed in this section are online scripts and operate on a policy store, regardless of whether it is file-, LDAP-, or DB-based. There are a few scripts that are **offline**, that is, they do not require a server to be running to operate.

Read-only scripts can be performed only by users in the following WebLogic groups: Monitor, Operator, Configurator, or Admin. Read-write scripts can be performed only by users in the following WebLogic groups: Admin or Configurator. All WLST commands are available out-of-the-box with the installation of the Oracle WebLogic Server.

WLST commands can be run in interactive mode or in script mode. In interactive mode, you enter the script at a command-line prompt; in script mode, you write scripts in a text file (with a py file name extension) and run it without requiring input, much like the directives in a shell script.

For platform-specific requirements to run a WLST command, see note in section Section 10.4, "Managing Application Policies with WLST commands."

In regards to configuration files and JVMs, see note in Section 5.6, "Typical Security Practices with WLST Commands."

OPSS provides the following scripts on all supported platforms to administer credentials (all scripts are **online**, unless otherwise stated):

- updateCred
- createCred
- deleteCred
- modifyBootStrapCredential
- addBootStrapCredential

- exportEncryptionKey
- importEncryptionKey
- restoreEncryptionKey
- rollOverEncryptionKey

Note: To retrieve a credential with a given map and key, use scripting to invoke the MBean operation `JpsCredentialMXBean.getPortableCredential(map, key).`

---

12 Managing Keys and Certificates with the Keystore Service

This chapter explains how to use the Keystore Service to administer keys and certificates.

This chapter contains these topics:

- About the Keystore Service
- Keystore Management with the Keystore Service
- Certificate Management with the Keystore Service
- How Oracle Fusion Middleware Components Use the Keystore Service
- About Keystore Service Commands
- Getting Help for Keystore Service Commands
- Keystore Service Command Reference

12.1 About the Keystore Service

The OPSS Keystore Service enables you to manage keys and certificates for SSL, message security, encryption, and related tasks. You use the Keystore Service to create and maintain keystores that contain keys, certificates, and other artifacts. You can also import or export to Oracle Wallet, a feature useful for system components such as Oracle HTTP Server that use wallets at runtime.

The following topics introduce Keystore Service concepts:

- Structure of the Keystore Service
- Types of Keystores
- Domain Trust Store
- Keystores for Domains with Multiple Servers
- Troubleshooting Keystore Service

12.1.1 Structure of the Keystore Service

Each keystore created with the Keystore Service is uniquely referenced by an application stripe and keystore.

- Application Stripe

Keys and certificates created in the keystore reside in an application stripe or product, and each stripe in a domain is uniquely named.

- Keystore

  The keystore name is unique within an application stripe. Each product or application is allowed to create more than one keystore within its application stripe.

Thus `(appstripe1, keystoreA)` is unique and distinct from `(appstripe1, keystoreB)`, which is distinct from `(appstripe2, keystoreA)`.

In turn, each keystore may contain the following entries, referenced by an alias that is unique within the keystore:

- Asymmetric Keys - These include the public key and the corresponding private key, and are typically used for SSL communication. The public key is wrapped in a certificate.
- Symmetric Keys - These keys are generally used for encryption.
- Trusted Certificates - These certificates are typically used to establish trust with an SSL peer.

12.1.2 Types of Keystores

The Keystore Service lets you create two types of keystores:

- Keystores protected by Permission

  These types of keystores are protected by authorization policies and any access to them by runtime code is protected by code source permissions. The key data in the backend is encrypted using an encryption key that is generated uniquely for each domain.

- Keystores protected by Password

  These types of keystores are protected by keystore and/or key passwords. Any access to them by runtime code requires access to the keystore and key password (if different from the keystore password). The key data in the back-end is encrypted using the keystore/key password through password-based encryption (PBE).

It is recommended that you use permission-protected keystores for applications. If you require high security and are willing to manage passwords, however, consider using keystores that are password-protected.

Note: The Keystore Service does not manage passwords for keystore or keys. The product or application is responsible for managing them in an appropriate repository. For example, you may choose to store the passwords for your applications in a credential store.

Export/Import Between Keystore and Wallet

You can export or import a keystore to/from Oracle Wallet, a feature useful for system components such as Oracle HTTP Server that use wallets at runtime.

For details, see Section 12.2.2.7 and Section 12.2.2.8.

12.1.3 Domain Trust Store

Caution: The Demo CA has a well known hard-coded private key, Care should be taken not to trust the certificates signed by the Demo CA. As such, the Demo CA certificate in the trust store should not be used in production. It should be removed from the domain trust store in production.

Although each application may configure multiple keystores for its SSL usage, a domain-level trust store comes pre-configured for all products and applications to use for trust management.

This domain trust store contains the trusted certificates of most well-known third-party Certificate Authorities (CAs) as well as the trusted certificate of the demo CA that is configured with the Keystore Service. Each application can simply point to this domain trust store for its SSL needs, eliminating the need to create a dedicated trust store for this task.

One-Way SSL

For one-way SSL, applications can simply use the domain trust store and do not need to create any keystore or trust store.

Two-Way SSL

For two-way SSL, applications should create only the keystore containing their identity certificate and use the domain trust store for trust.

Note: The domain trust store is a shared store for all products and applications in a domain. The decision to add or remove trust should not be taken lightly since it may affect all other products in the domain.

Consider creating a custom trust store only if a product's trust management requirements are not met by the domain trust store.

12.1.4 Keystores for Domains with Multiple Servers

For a domain with multiple servers, the only recommended store types are LDAP or DB. Do not configure a file-based store in such an environment.

12.1.5 Troubleshooting Keystore Service

Troubleshooting topics related to the keystore service are addressed in Appendix J. Currently, the following topic may be of interest:

- Section J.10.6, "Invalid Key Size".

12.2 Keystore Management with the Keystore Service

This section describes the typical lifecycle of keystores and certificates, and how to use the keystore service to create and maintain keystores and certificates. It includes these topics:

- About the Keystore Lifecycle

- Common Keystore Operations

12.2.1 About the Keystore Lifecycle

Typical lifecycle events for a KSS keystore are as follows:

- The keystore is created in the context of an application stripe. Keystores can be created directly or by importing a keystore file from the file system.
- The list of available keystores is viewed and specific keystores selected for update.
- Keystores are updated or deleted. For a password-protected keystore, update operations require that the keystore password be entered.
- The keystore password can be changed.
- The keystore can be deleted.
- Keystores can be exported and imported. KSS supports migration for these certificate formats:
  - JKS
  - JCEKS
  - Oracle Wallet

12.2.2 Common Keystore Operations

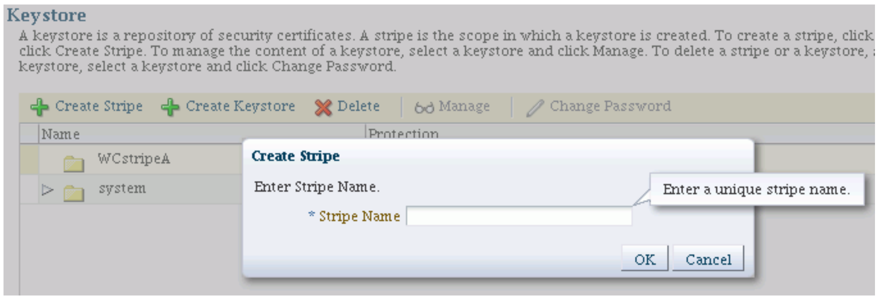
This section explains the following keystore operations that you can perform through Fusion Middleware Control or at the command line:

- Creating a Keystore with Fusion Middleware Control
- Creating a Keystore at the Command Line
- Deleting a Keystore with Fusion Middleware Control
- Deleting a Keystore at the Command Line
- Changing Keystore Password with Fusion Middleware Control
- Changing Keystore Password at the Command Line
- Exporting a Keystore at the Command Line
- Importing a Keystore at the Command Line

12.2.2.1 Creating a Keystore with Fusion Middleware Control

Take these steps to create a keystore:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Select the stripe in which the keystore is to be created. If necessary, create a stripe as follows:
   a. Click **Create Stripe**. The Create Stripe dialog appears.



Keystore

A keystore is a repository of security certificates. A stripe is the scope in which a keystore is created. To create a stripe, click
click Create Stripe. To manage the content of a keystore, select a keystore and click Manage. To delete a stripe or a keystore,
keystore, select a keystore and click Change Password.

Create Stripe

Enter Stripe Name.

* Stripe Name

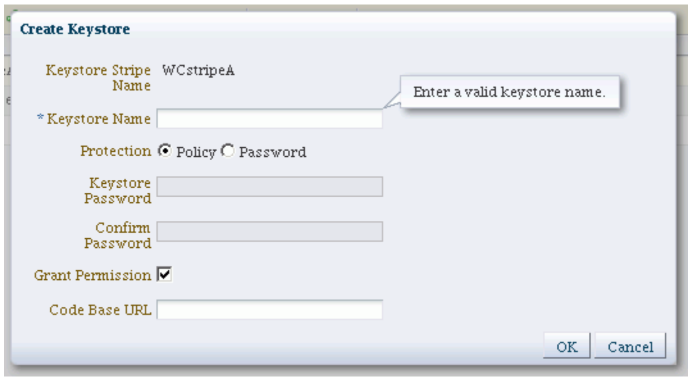
Enter a unique stripe name.

b. Provide a unique stripe name. Any combination of characters is possible, but it is recommended that you do not use the forward slash (/) in the name.

c. Click **Submit**. The new stripe appears in the list of stripes and you can select it for keystore creation.

5. Click **Create Keystore**.

6. The Create Keystore dialog appears.



Create Keystore

Keystore Stripe Name: WCstripeA

* Keystore Name

Enter a valid keystore name.

Protection: [✓ Policy] Password

Keystore Password

Confirm Password

Grant Permission: ✓

Code Base URL

7. Complete the dialog form as follows:

   - Keystore Name: Enter a unique keystore name. Do not use any special (non-ascii) characters or characters from a different encoding or locale.
   - Protection Type: Select the protection mechanism for the keystore; choose between Policy and Password.
   - For a password-protected keystore, enter a valid password.
   - Grant Permission: Check this box to grant permissions using code URL.

8. Click **OK**. The new keystore appears under the appropriate stripe.

12.2.2.2 Creating a Keystore at the Command Line

You can create a keystore using the `createKeyStore` script at the command line. For example, assuming the stripe name is `teststripe1`, use this command to create a permission-based keystore:

```
svc.createKeyStore(appStripe='teststripe1', name='keystore1',
password='password',permission=true)
```

where *`password`* is the password for this keystore. Section 12.5 explains how to obtain the OPSS service command object.

Any combination of characters is possible when specifying a new stripe, but it is recommended that you do not use the forward slash (/) in the name.

Enter a unique keystore name. Do not use any special (non-ascii) characters or characters from a different encoding or locale.

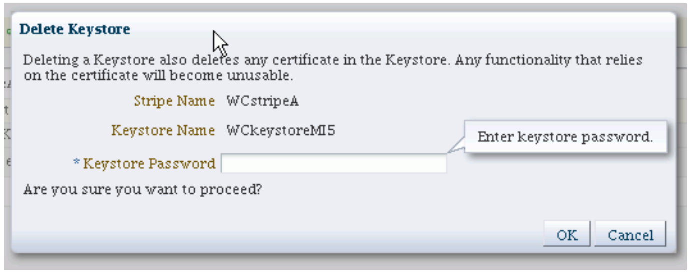
> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*

12.2.2.3 Deleting a Keystore with Fusion Middleware Control

When you delete a keystore, be aware that all certificates in the keystore are also deleted. If any functions rely on these certificates, they will be rendered unusable as a result.

Take these steps to delete a keystore:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Delete**.
6. The Delete Keystore dialog appears.



Delete Keystore

Deleting a Keystore also deletes any certificate in the Keystore. Any functionality that relies on the certificate will become unusable.

Stripe Name WCstripeA

Keystore Name WCkeystoreMI5

* Keystore Password

Enter keystore password.

Are you sure you want to proceed?

7. If this is a password-protected keystore, enter the keystore password.
8. Click **OK**.

12.2.2.4 Deleting a Keystore at the Command Line

You can delete a keystore using the `deleteKeyStore` script at the command line. For example, assuming the stripe is named `appstripe1`, use this command to delete a keystore:

```
svc.deleteKeyStore(appStripe='appstripe1', name='keystore1', password='password')
```

where *`password`* is the password for this keystore. Section 12.5 explains how to obtain the OPSS service command object.

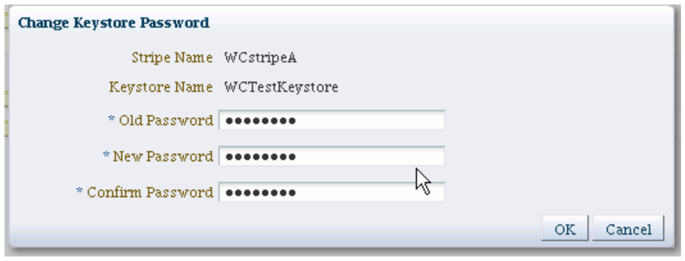
> **See Also:** Oracle Fusion Middleware Infrastructure Security WLST Command Reference

12.2.2.5 Changing Keystore Password with Fusion Middleware Control

To change the password for a password-protected keystore:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.

3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Change Password**.
6. The Change Keystore Password dialog appears.


Change Keystore Password

Stripe Name	WCstripeA
Keystore Name	WCTestKeystore
* Old Password	••••••••
* New Password	••••••••
* Confirm Password	••••••••

7. Enter the old and new passwords.
8. Click **OK**.

12.2.2.6 Changing Keystore Password at the Command Line

You can change a keystore's password using the `changeKeyStorePassword` script at the command line. For example, assuming the instance name is system, use this command to change the keystore password:

```
svc.changeKeyStorePassword(appStripe='system', name='keystore2',
currentpassword='currentpassword', newpassword='newpassword')
```

where `currentpassword` is the current password for this keystore, and `newpassword` is the new password. Section 12.5 explains how to obtain the OPSS service command object.

12.2.2.7 Exporting a Keystore at the Command Line

You can export a keystore using the `exportKeyStore` script at the command line.

Exporting a Single Key

For example, assuming stripe name mystripe, alias named myorakey to be exported, alias password keypassword1, the following command exports the keystore to a file:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myorakey', keypasswords='keypassword1',
type='JKS',filepath='/tmp/file.jks')
```

where *`password`* is the password for this keystore. Section 12.5 explains how to obtain the OPSS service command object.

Exporting Multiple Keys

To export multiple keys using this command, specify a comma-separated list of aliases and key passwords.

Exporting a Symmetric Key

When exporting a keystore containing a symmetric key, use the `JCEKS` type. For example:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myorakey', keypasswords='keypassword1',
type='JCEKS',filepath='/tmp/file.jks')
```

Exporting to an Oracle Wallet

To export a keystore to an Oracle Wallet, use the type 'OracleWallet'. For example:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore3',
password='mypassword',aliases='myorakey1,myorakey2', keypasswords='',
type='OracleWallet',path='/tmp')
```

See Also: Oracle Fusion Middleware Infrastructure Security WLST Command Reference

12.2.2.8 Importing a Keystore at the Command Line

You can import a keystore using the `importKeyStore` script at the command line.

Importing a Single Key

For example, assuming stripe name mystripe, alias named myorakey to be imported, alias password keypassword1, the following command imports a keystore from an operating system file:

```
svc.importKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myorakey', keypasswords='keypassword1', type='JKS',
permission=true, filepath='/tmp/file.jks')
```

where *`password`* is the keystore password. Section 12.5 explains how to obtain the OPSS service command object.

Importing Multiple Keys

To import multiple keys using this command, specify a comma-separated list of aliases and key passwords.

Importing an Oracle Wallet

To import one or more keys from an Oracle Wallet, use the type 'OracleWallet'. For example:

```
svc.importKeyStore(appStripe='mystripe', name='keystore4',
password='owPwd1234',aliases='myorakey1,myorakey2', keypasswords='',
type='OracleWallet', permission=true, filepath='/tmp')
```

See Also: Oracle Fusion Middleware Infrastructure Security WLST Command Reference

12.3 Certificate Management with the Keystore Service

This section explains how to manage certificates with the Keystore Service. It contains these topics:

- About the Certificate Lifecycle
- Common Certificate Operations

12.3.1 About the Certificate Lifecycle

Typical lifecycle events for a certificate residing in a Keystore Service (KSS) keystore are as follows:

- A self-signed certificate is automatically created for the keypair.
- A certificate signing request (CSR) is generated for the certificate, and can be saved to a file.
- The CSR is sent to a certificate authority, who verifies the sender, signs and returns the signed certificate.
- Certificates are imported into the keystore. A certificate can either be pasted into a text box or imported from the file system. You can import both user certificates and trusted certificates (also known as CA certificates) in this way.

Note: Keystore Service supports import of PEM/BASE64-encoded certificates only. You cannot import DER-encoded certificates or trusted certificates into a KSS keystore.

- Certificates or trusted certificates are exported from the keystore out to a file.
- Certificates or trusted certificates are deleted from the keystore.

12.3.2 Common Certificate Operations

This section explains the common certificate operations:

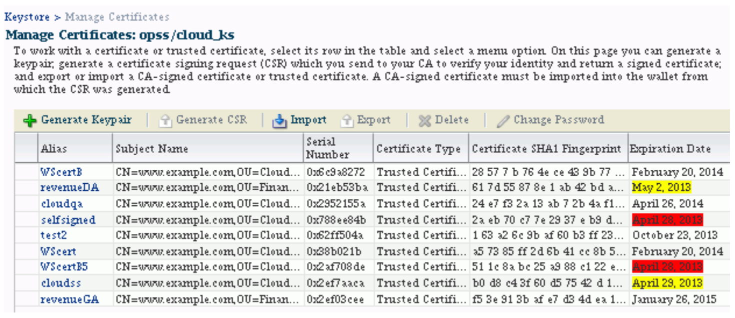
- Generating a Keypair with Fusion Middleware Control
- Generating a Keypair at the Command Line
- Generating CSR for a Certificate with Fusion Middleware Control
- Generating CSR for a Keypair at the Command Line
- Importing a Certificate or Trusted Certificate with Fusion Middleware Control
- Importing a Certificate at the Command Line
- Exporting a Certificate or Trusted Certificate with Fusion Middleware Control
- Exporting a Certificate or Trusted Certificate at the Command Line
- Deleting a Certificate with Fusion Middleware Control
- Deleting a Certificate at the Command Line
- Changing Certificate Password with Fusion Middleware Control
- Changing Certificate Password at the Command Line

12.3.2.1 Generating a Keypair with Fusion Middleware Control

To generate a certificate with an associated keypair:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.

5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears:



Keystore > Manage Certificates

Manage Certificates: opss/cloud_ks

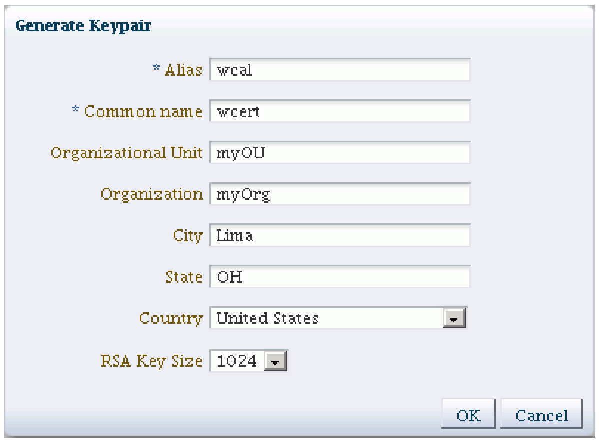
To work with a certificate or trusted certificate, select its row in the table and select a menu option. On this page you can generate a keypair; generate a certificate signing request (CSR) which you send to your CA to verify your identity and return a signed certificate; and export or import a CA-signed certificate or trusted certificate. A CA-signed certificate must be imported into the wallet from which the CSR was generated.

Alias	Subject Name	Serial Number	Certificate Type	Certificate SHA1 Fingerprint	Expiration Date
WScertB	CN=www.example.com,OU=Cloud...	0x6c9a8272	Trusted Certifi...	28 57 7 b 76 4e ce 43 9b 77 ...	February 20, 2014
revenueDA	CN=www.example.com,OU=Finan...	0x21eb53ba	Trusted Certifi...	61 7d 55 87 8e 1 ab 42 bd a...	May 2, 2013
cloudqa	CN=www.example.com,OU=Cloud...	0x2952155a	Trusted Certifi...	24 e7 f3 2a 13 ab 7 2b 4a f1...	April 26, 2014
selfsigned	CN=www.example.com,OU=Cloud...	0x788ee84b	Trusted Certifi...	2a eb 70 c7 7e 29 37 e b9 d...	April 28, 2013
test2	CN=www.example.com,OU=Cloud...	0x62ff504a	Trusted Certifi...	1 63 a2 6c 9b af 60 b3 ff 23...	October 23, 2013
WScert	CN=www.example.com,OU=Cloud...	0x38b021b	Trusted Certifi...	a5 73 85 ff 2d 6b 41 cc 8b 5...	February 20, 2014
WScertB5	CN=www.example.com,OU=Cloud...	0x2af708de	Trusted Certifi...	51 1c 8a bc 25 a9 88 c1 22 e...	April 28, 2013
cloudss	CN=www.example.com,OU=Cloud...	0x2ef7aaca	Trusted Certifi...	b0 d8 c4 3f 60 d5 75 42 d 1...	April 29, 2013
revenueGA	CN=www.example.com,OU=Finan...	0x2ef03cee	Trusted Certifi...	f5 3e 91 3b af e7 d3 4d ea 1...	January 26, 2015

The Expiration Date column enables you to quickly assess which certificates are expiring or already expired.

Click **Generate Keypair**.

8. The Generate Keypair dialog appears:

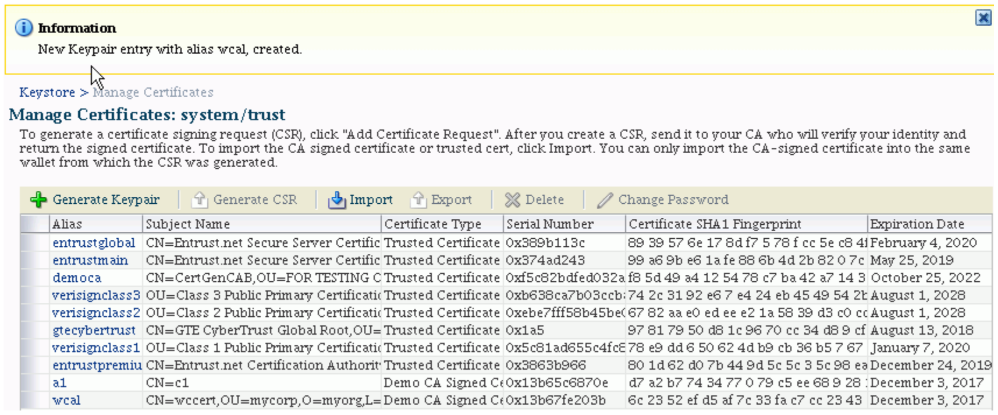


Generate Keypair

* Alias	wcal
* Common name	wcert
Organizational Unit	myOU
Organization	myOrg
City	Lima
State	OH
Country	United States
RSA Key Size	1024

9. Provide the following information:
   - Alias (required)
   - Common Name (required)
   - Organizational Unit
   - Organization
   - City
   - State
   - Country: Choose from the drop-down box.
   - RSA Key Size: Choose from the drop-down box. Default is 1024 bytes.
10. Click **OK**.

    The new certificate appears in the list of certificates.



Information
New Keypair entry with alias wcal, created.

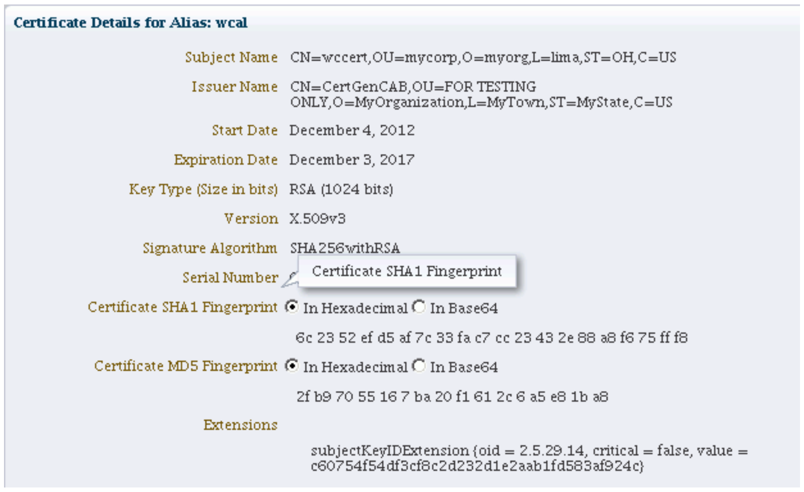
Keystore > Manage Certificates

Manage Certificates: system/trust

To generate a certificate signing request (CSR), click "Add Certificate Request". After you create a CSR, send it to your CA who will verify your identity and return the signed certificate. To import the CA signed certificate or trusted cert, click Import. You can only import the CA-signed certificate into the same wallet from which the CSR was generated.

Alias	Subject Name	Certificate Type	Serial Number	Certificate SHA1 Fingerprint	Expiration Date
entrustglobal	CN=Entrust.net Secure Server Certific	Trusted Certificate	0x389b113c	89 39 57 6e 17 8d f7 5 78 f cc 5e c8 4f	February 4, 2020
entrustmain	CN=Entrust.net Secure Server Certific	Trusted Certificate	0x374ad243	99 a6 9b e6 1a fe 88 6b 4d 2b 82 0 7c	May 25, 2019
democa	CN=CertGenCAB,OU=FOR TESTING C	Trusted Certificate	0xf5c82bdfed032a	f8 5d 49 a4 12 54 78 c7 ba 42 a7 14 3	October 25, 2022
verisignclass3	OU=Class 3 Public Primary Certificati	Trusted Certificate	0xb638ca7b03ccb	74 2c 31 92 e6 7 e4 24 eb 45 49 54 2b	August 1, 2028
verisignclass2	OU=Class 2 Public Primary Certificati	Trusted Certificate	0xebe7fff58b45be	67 82 aa e0 ed ee e2 1a 58 39 d3 c0 cc	August 1, 2028
gtecybertrust	CN=GTE CyberTrust Global Root,OU=	Trusted Certificate	0x1a5	97 81 79 50 d8 1c 96 70 cc 34 d8 9 cf	August 13, 2018
verisignclass1	OU=Class 1 Public Primary Certificati	Trusted Certificate	0x5c81ad655c4fc8	78 e9 dd 6 50 62 4d b9 cb 36 b5 7 67	January 7, 2020
entrustpremiu	CN=Entrust.net Certification Authorit	Trusted Certificate	0x3863b966	80 1d 62 d0 7b 44 9d 5c 5c 3 5c 98 ea	December 24, 2019
a1	CN=c1	Demo CA Signed Ce	0x13b65c6870e	d7 a2 b7 74 34 77 0 79 c5 ee 68 9 28	December 3, 2017
wcal	CN=wccert,OU=mycorp,O=myorg,L=	Demo CA Signed Ce	0x13b67fe203b	6c 23 52 ef d5 af 7c 33 fa c7 cc 23 43	December 3, 2017

11. You can view the certificate details by clicking on the certificate alias:



Certificate Details for Alias: wcal

Subject Name	CN=wccert,OU=mycorp,O=myorg,L=lima,ST=OH,C=US
Issuer Name	CN=CertGenCAB,OU=FOR TESTING ONLY,O=MyOrganization,L=MyTown,ST=MyState,C=US
Start Date	December 4, 2012
Expiration Date	December 3, 2017
Key Type (Size in bits)	RSA (1024 bits)
Version	X.509v3
Signature Algorithm	SHA256withRSA
Serial Number	
Certificate SHA1 Fingerprint	[✓ In Hexadecimal] In Base64 6c 23 52 ef d5 af 7c 33 fa c7 cc 23 43 2e 88 a8 f6 75 ff f8
Certificate MD5 Fingerprint	[✓ In Hexadecimal] In Base64 2f b9 70 55 16 7 ba 20 f1 61 2c 6 a5 e8 1b a8
Extensions	subjectKeyIDExtension {oid = 2.5.29.14, critical = false, value = c60754f54df3cf8c2d232d1e2aab1fd583af924c}

The generated keypair is wrapped in a CA signed certificate (using a Demo CA). To use this certificate for SSL or where trust needs to be established, applications must either use the domain trust store as their trust store (since it contains the Demo CA certificate) or import the certificate to a custom application-specific trust store.

12.3.2.2 Generating a Keypair at the Command Line

You can generate a keypair for a keystore using the `generateKeyPair` script at the command line. For example, assuming an application stripe named appstripe2, the following command creates a keypair with alias myalias:

```
svc.generateKeyPair(appStripe='appstripe2', name='keystore2', password='password',
dn='cn=www.example.com', keysize='1024', alias='myalias',
keypassword='keypassword')
```

where *`password`* is the keystore password and *`keypassword`* is the password of the alias. Section 12.5 explains how to obtain the OPSS service command object.

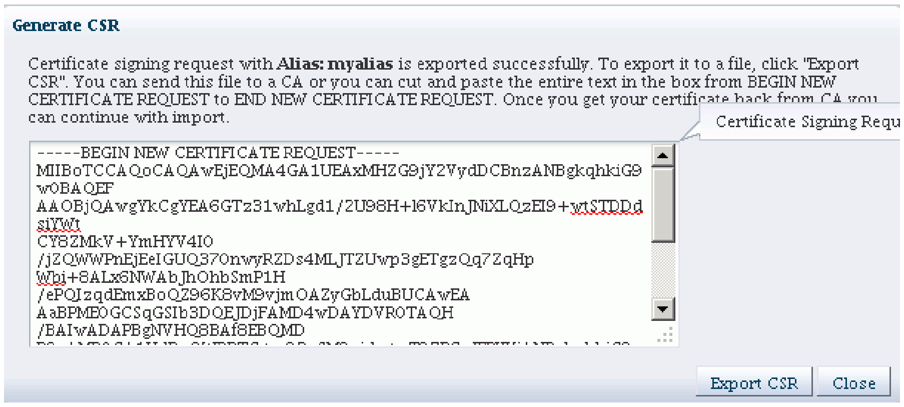
> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference.*

The generated keypair is wrapped in a CA signed certificate (using a Demo CA). To use this certificate for SSL or where trust needs to be established, applications must either use the domain trust store as their trust store (since it contains the Demo CA certificate) or import the certificate to a custom application-specific trust store.

12.3.2.3 Generating CSR for a Certificate with Fusion Middleware Control

To generate a CSR for a certificate or trusted certificate:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears. Select the row corresponding to the certificate and click **Generate CSR**.
8. The Generate CSR dialog appears:



Certificate signing request with **Alias: myalias** is exported successfully. To export it to a file, click "Export CSR". You can send this file to a CA or you can cut and paste the entire text in the box from BEGIN NEW CERTIFICATE REQUEST to END NEW CERTIFICATE REQUEST. Once you get your certificate back from CA you can continue with import.

```
-----BEGIN NEW CERTIFICATE REQUEST-----
MIIBoTCCAQoCAQAwEjEQMA4GA1UEAxMHZG9jY2VydDCBnzANBgkqhkiG9
w0BAQEF
AAOBjQAwgYkCgYEA6GTz31whLgd1/2U98H+l6VkInJNiXLQzEI9+wtSTDDd
siYWt
CY8ZMkV+YmHYV4IO
/jZQWWPnEjEeIGUQ370nwyRZDs4MLJTZUwp3gETgzQq7ZqHp
Wbi+8ALx6NWAbJhOhbSmP1H
/ePQIzqdEmxBoQZ96K8vM9vjmOAZyGbLduBUCAwEA
AaBPME0GCSqGSIb3DQEJDjFAMD4wDAYDVR0TAQH
/BAIwADAPBgNVHQ8BAf8EBQMD
```

You can:

- Copy and paste the entire CSR into a text file, and click Close.

  or

- Click Export CSR to automatically save the CSR to a file.

You can send the resulting certificate request to a certificate authority (CA) which will return a signed certificate.

12.3.2.4 Generating CSR for a Keypair at the Command Line

You can generate a CSR for a keypair using the `exportKeyStoreCertificateRequest` script at the command line. For example, assuming an application stripe stripe1, the following command generates a CSR from the keypair testalias:

```
svc.exportKeyStoreCertificateRequest(appStripe='stripe1', name='keystore1',
password='password', alias='testalias', keypassword='keypassword',
filepath='/tmp/csr-file')
```

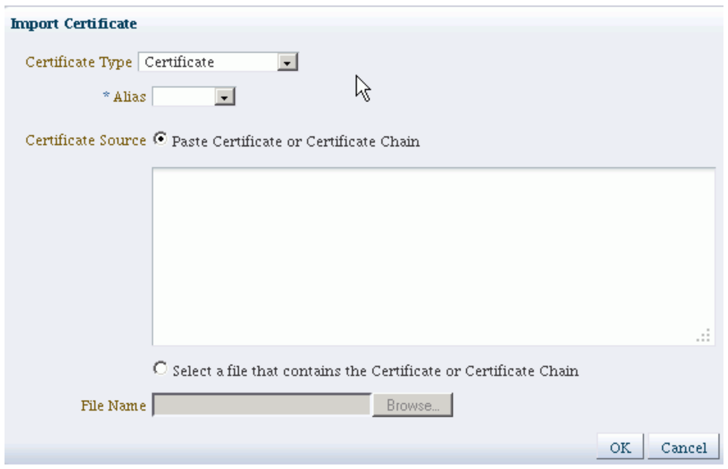
where *`password`* is the keystore password and *`keypassword`* is the password of the alias. The CSR is exported to an operating system file. Section 12.5 explains how to obtain the OPSS service command object.

> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

12.3.2.5 Importing a Certificate or Trusted Certificate with Fusion Middleware Control

To import a certificate into a password-protected keystore:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears. Click **Import**.
8. The Import Certificate dialog appears:


Import Certificate
Certificate Type
Certificate
* Alias
Certificate Source
Paste Certificate or Certificate Chain
Select a file that contains the Certificate or Certificate Chain
File Name
Browse...
OK
Cancel

   Complete this form as follows:

   - Select the certificate type, either Certificate or Trusted Certificate, from the drop-down.
   - Select the alias from the drop-down.
   - Specify the certificate source. If using the Paste option, copy and paste the certificate directly into the text box. If using the Select a file option, click **Browse** to select the file from the operating system.
   - Click **OK**. The imported certificate or trusted certificate appears in the list of certificates.

9. Click **OK**.

   The certificate appears in the list of certificates.

12.3.2.6 Importing a Certificate at the Command Line

You can import a certificate using the importKeyStoreCertificate script at the command line. For example, assuming an application stripe appstripe1, the following command imports a certificate with alias `mykey` from an operating system file:

```
svc.importKeyStoreCertificate(appStripe='appstripe1', name='keystore2',
password='password', alias='mykey', keypassword='keypassword', type='Certificate',
filepath='/tmp/cert.txt')
```

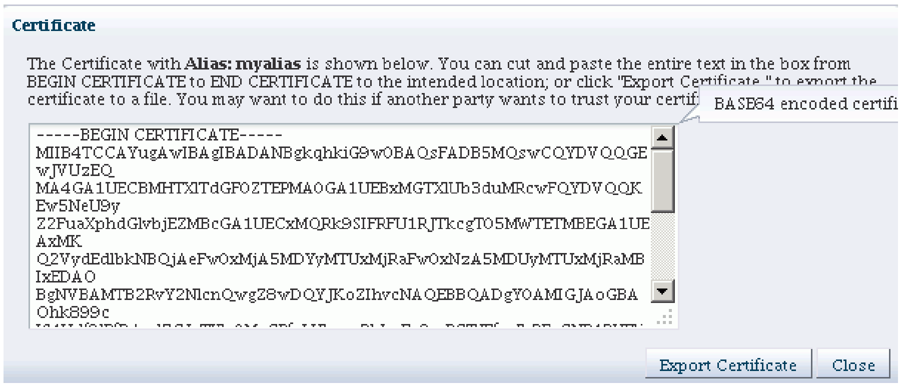
where *password* is the keystore password and *keypassword* is the password of the alias. Section 12.5 explains how to obtain the OPSS service command object.

> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference.*

12.3.2.7 Exporting a Certificate or Trusted Certificate with Fusion Middleware Control

Take these steps to export a certificate or trusted certificate:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears. Select the row corresponding to the certificate and click **Export**.
8. The certificate export dialog appears:



Certificate

The Certificate with **Alias: myalias** is shown below. You can cut and paste the entire text in the box from BEGIN CERTIFICATE to END CERTIFICATE to the intended location; or click "Export Certificate " to export the certificate to a file. You may want to do this if another party wants to trust your certif

BASE64 encoded certifi

```
-----BEGIN CERTIFICATE-----
MIIB4TCCAYugAwIBAgIBADANBgkqhkiG9w0BAQsFADB5MQswCQYDVQQGE
wJVUzEQ
MA4GA1UECBMHTXlTdGF0ZTEPMA0GA1UEBxMGTXlUb3duMRcwFQYDVQQK
Ew5NeU9y
Z2FuaXphdGlvbjEZMBcGA1UECxMQRk9SIFRFU1RJTkcgT05MWTETMBEGA1UE
AxMK
Q2VydEdlbkNBQjAeFw0xMjA5MDYyMTUxMjRaFw0xNzA5MDUyMTUxMjRaMB
IxEDAO
BgNVBAMTB2RvY2NlcnQwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBA
Ohk899c
```

You can:

- Copy and paste the entire certificate into a text file, and click **Close**.

  or

- Click **Export Certificate** to automatically save the certificate to a file.

12.3.2.8 Exporting a Certificate or Trusted Certificate at the Command Line

You can export a certificate using the exportKeyStoreCertificate script at the command line. For example, assuming an application stripe appstripe1, the following command exports a certificate with alias `mykey` to an operating system file:

```
svc.exportKeyStoreCertificate(appStripe='appstripe1', name='keystore2',
password='password', alias='mykey', keypassword='keypassword', type='Certificate',
filepath='/tmp/cert.txt')
```

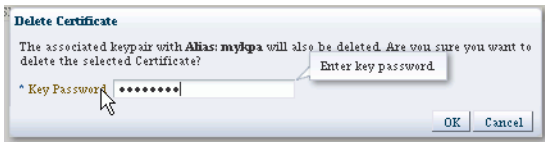
where `password` is the keystore password and `keypassword` is the password of the alias. Section 12.5 explains how to obtain the OPSS service command object.

> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference.*

12.3.2.9 Deleting a Certificate with Fusion Middleware Control

Take these steps to delete a certificate:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.
4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears. Select the row corresponding to the certificate and click **Delete**.
8. The Delete Certificate dialog appears:



Delete Certificate

The associated keypair with **Alias: mykpa** will also be deleted. Are you sure you want to delete the selected Certificate?

* Key Password ••••••••

   You are asked to confirm deletion. Click OK.

12.3.2.10 Deleting a Certificate at the Command Line

You can delete a certificate from a keystore using the deleteKeyStoreEntry script at the command line. For example, assuming an application stripe appstripe, the following command deletes a certificate with alias orakey:

```
ssvc.deleteKeyStoreEntry(appStripe='appstripe', name='keystore2',
password='password', alias='orakey', keypassword='keypassword')
```

where `password` is the keystore password and `keypassword` is the password of the alias. Section 12.5 explains how to obtain the OPSS service command object.

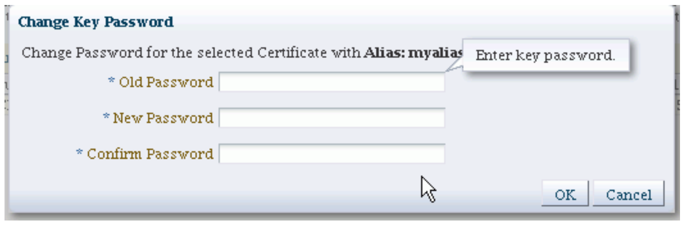
> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference.*

12.3.2.11 Changing Certificate Password with Fusion Middleware Control

Take these steps to change certificate password:

1. Log in to Fusion Middleware Control.
2. From the navigation pane, locate the domain of interest.
3. Navigate to **Security**, then **Keystore**. The Keystore page appears.

4. Expand the stripe in which the keystore resides. Select the row corresponding to the keystore.
5. Click **Manage**.
6. If the keystore is password-protected, you are prompted for a password. Enter the keystore password and click **OK**.
7. The **Manage Certificates** page appears. Select the row corresponding to the certificate and click **Change Password**.
8. The Change Key Password dialog appears:



Change Key Password

Change Password for the selected Certificate with **Alias: myalias**

Enter key password.

* Old Password

* New Password

* Confirm Password

9. Enter the old and new passwords and click **OK**.

12.3.2.12 Changing Certificate Password at the Command Line

You can change a certificate password using the changeKeyPassword script at the command line. For example, assuming an application stripe system1, the following command deletes a certificate with alias testkey:

```
svc.changeKeyPassword(appStripe='system1', name='keystore', password='password',
alias='testkey', currentkeypassword='currentkeypassword',
newkeypassword='newkeypassword')
```

where *password* is the keystore password and *keypassword* is the password of the certificate alias. Section 12.5 explains how to obtain the OPSS service command object.

> **See Also:** *Oracle Fusion Middleware Infrastructure Security WLST Command Reference.*

12.4 How Oracle Fusion Middleware Components Use the Keystore Service

This section explains how components and services make use of the Keystore Service to manage keys and certificates. Topics include:

- Using the syncKeyStores Command
- Integrating with Oracle WebLogic Server
- Integrating with Node Manager
- Integrating with the Identity Store Service

12.4.1 Using the syncKeyStores Command

In the Oracle WebLogic Server environment, the "source of truth" for keystores and truststores is the OPSS security store, which acts as the central security store. Oracle recommends that all Oracle Fusion Middleware stack components use this central store; however, certain infrastructure components (like Oracle WebLogic Server and Node Manager) require bootstrapping for which the local keystore instance is used.

The WLST `syncKeyStores` command is used to create the local file instance from the central security store. Synchronization is a one-way procedure in which key data is read from the database security store to synchronize the local file instance.

For example, when you update the domain trust store in the central security store, the local keystore copy used by Oracle Weblogic Server can be synchronized with the central store using this command.

- syncKeyStores Command Usage
- When to Use the syncKeyStores Command

12.4.1.1 syncKeyStores Command Usage

The syntax of the syncKeyStores command is as follows:

```
syncKeyStores ()
```

This command looks for the "system" stripe in the central repository and download its contents into a file named `keystores.xml` under the `DOMAIN_HOME/config/fmwconfig` directory. It also downloads the contents of the domain truststore into the same file.

12.4.1.2 When to Use the syncKeyStores Command

Use the following guidelines to determine when you should use the `syncKeyStores` command:

- If the keystore being updated belongs to Oracle WebLogic Server, such a keystore would likely be present under the "system" stripe; for example "demoidentity" keystore.
- If the domain trust store is updated (either a trusted certificate is added or removed), then invoke the synchronization command for Oracle WebLogic Server.
- Updating the keystore for any layered component like Web Services, Oracle Web Services Manager, or JavaEE applications does **not** require executing the synchronization command. Such components directly access their key material from the central security store.

12.4.2 Integrating with Oracle WebLogic Server

When configuring keystores for use with Oracle WebLogic Server, you can use Keystore Service to generate the keys and certificates.

For details about keystore and SSL configuration in Oracle WebLogic Server see:

- "Configure Keystores" in the Oracle WebLogic Server Administration Console Online Help
- "Configuring SSL" in *Administering Security for Oracle WebLogic Server*.

12.4.3 Integrating with Node Manager

You can configure Node Manager to use the Keystore Service. For details, see "Configuring Node Manager to Use the OPSS Keystore Service" in *Administering Oracle Fusion Middleware*.

For more information about Node Manager, see *Administering Node Manager for Oracle WebLogic Server*.

12.4.4 Integrating with the Identity Store Service

You can configure SSL communication from the identity store to LDAP servers. For details, see Section 8.5.2.

12.5 About Keystore Service Commands

The Keystore Service uses a dedicated set of command-line commands for keystore operations such as creating and managing keystores, exporting certificates, and generating key pairs. While their usage is similar, these commands are distinct from other OPSS commands.

The starting point for using the Keystore Service command set is `getOpssService`, which gets an OPSS service command object that enables you to:

- execute commands for the service
- obtain command help

The general syntax is:

```
variable = getOpssService(name='service_name')
```

where

- the `variable` stores the command object
- the service name refers to the service whose command object is to be obtained. The only valid value is '`KeyStoreService`'.

For example:

```
svc = getOpssService(name='KeyStoreService')
```

12.6 Getting Help for Keystore Service Commands

To obtain help for any Keystore Service command, start by obtaining a service command object as explained in Section 12.5. Use this object in conjunction with the help command and the command in question.

To obtain a list of all Keystore Service commands, enter:

```
svc.help()
```

To obtain help for a specific command, enter:

```
svc.help('command-name')
```

For example, the following returns help for the `exportKeyStore` command:

```
svc.help('exportKeyStore')
```

12.7 Keystore Service Command Reference

For syntax and reference information about the Keystore Service commands, see "OPSS Keystore Service Commands" in the *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

13 Introduction to Oracle Fusion Middleware Audit Service

This chapter introduces the Audit Service in Oracle Fusion Middleware.

In Oracle Fusion Middleware, auditing provides a measure of accountability and answers the "who has done what and when" types of questions. This chapter contains the following topics:

- Benefits and Features of the Oracle Fusion Middleware Audit Framework
- Overview of Audit Features
- Understanding Oracle Fusion Middleware Audit Framework Concepts
- About the Audit Metadata Model
- About Audit Definition Files

13.1 Benefits and Features of the Oracle Fusion Middleware Audit Framework

This section contains these topics:

- What are the Objectives of Auditing?
- Understanding Today’s Audit Challenges
- About the Oracle Fusion Middleware Audit Framework

13.1.1 What are the Objectives of Auditing?

With compliance becoming an integral part of any business requirement, audit support is also becoming a focus in enterprise deployments. Customers are looking for application vendors to provide out-of-the-box audit support. In addition, middleware customers who are deploying custom applications would like to centralize the auditing of their deployed applications wherever audit is appropriate.

IT organizations are looking for several key audit features driven by compliance, monitoring, and analytics requirements.

Compliance

Compliance is obviously a major requirement in the enterprise. With regulations such as Sarbanes-Oxley (financial) and Health Insurance Portability and Accountability Act (healthcare), many customers must now be able to audit on identity information and user access on applications and devices. These include events like:

- User profile change
- Access rights changes
- User access activity
- Operational activities like starting and stopping applications, upgrades, and backups

This allows compliance officers to perform periodic reviews of compliance policies.

Monitoring

The audit data naturally provides a rich set of data for monitoring purpose. In addition to any log data and component metrics that are exposed, audit data can be used to create dashboards and to build Key Performance Indicators (KPIs) for alerts to monitor the health of the various systems on an ongoing basis.

Analytics

Audit data can also be used in assessing the efficacy of controls through analysis on the audit data. The data can also be used for risk analysis. Based on historical data, a risk score can be calculated and assigned to any user. Any runtime evaluation of user access can include the various risk scores as additional criteria to protect access to the systems.

13.1.2 Understanding Today's Audit Challenges

To satisfy the audit requirements, IT organizations often battle with the deficiencies in audit support for their deployed applications. There is no reliable standard for:

- Audit Record Generation
- Audit Record Format and Storage
- Audit Policy Definition

As a result, today's audit solutions suffer from a number of key drawbacks:

- There is no centralized audit framework.
- The quality of audit support is inconsistent from application to application.
- Audit data is scattered across the enterprise.
- Complex data correlation is required before any meaningful cross-component analysis can be conducted.
- Audit policies and their configurations are also scattered.

These factors are costing IT organization considerable amount of time and resources to build and maintain any reasonable audit solutions. With the data scattered among individual silos, and the lack of consistency and centralization, the audit solutions also tend to be fragile with idiosyncrasies among applications from different vendors with their current audit capabilities.

13.1.3 About the Oracle Fusion Middleware Audit Framework

Oracle Fusion Middleware Audit Framework is designed to provide a centralized audit framework for the middleware family of products. The framework provides an audit service for the following:

- Middleware Platform - This includes Java components such as Oracle Platform Security Services (OPSS) and Oracle Web Services. These components are leveraged by applications deployed on the middleware. Indirectly, all the deployed applications leveraging these Java components benefit from the auditing events that happen at the platform level.

- Java EE applications - The objective is to provide a service for Java EE applications, including Oracle's own Java EE-based components. Java EE applications are able to create application-specific audit events.
- System Components - For system components like Oracle HTTP Server, the audit service also provides an end-to-end structure similar to that for Java components, including an audit API for C/C++ based applications.

See Also: Understanding Key Oracle Fusion Middleware Concepts in the *Administering Oracle Fusion Middleware*.

13.2 Overview of Audit Features

Key features of the Oracle Fusion Middleware Audit Framework and audit service include:

- A uniform system for administering audits across a range of Java components, system components, and applications
- Extensive support for Java component auditing, which includes:
  - support for Oracle Platform Security Services auditing for non-audit-aware applications
  - the ability to search for audit data at any application level
- Capturing authentication history/failures, authorization history, user management, and other common transaction data
- Flexible audit policies
  - pre-seeded audit policies, capturing customers' most common audit events, are available for ease of configuration
  - tree-like policy structure simplifies policy setup
- Ability to write your own reports based on the published audit schema. See Chapter 15, "Using Audit Analysis and Reporting" for details.
- Audit record storage

  Audit data store (database) and files (bus-stop) are available. Maintaining a common location for all audit records simplifies maintenance.

  Using an audit data store lets you generate reports with Oracle Business Intelligence Publisher.
- Common audit record format

  Highlights of the audit trail include:
  - baseline attributes like outcome (status), event date-time, user, and so on
  - event-specific attributes like authentication method, source IP address, target user, resource, and so on
  - contextual attributes like the execution context ID (ECID), session ID, and others
- Common mechanism for audit policy configuration

Oracle Fusion Middleware Audit Framework offers a unified method for configuring audit policies in the domain.

- Leverages the Oracle Fusion Middleware infrastructure
  - is usable across Oracle Fusion Middleware components and services
  - integrates with Oracle Enterprise Manager Fusion Middleware Control for UI-based configuration and management
  - integrates with `wlst` for command-line, script-based configuration
  - leverages the SPI infrastructure of Oracle Platform Security Services
- Utilizes a dynamic metadata model to enable applications to integrate with the audit framework:
  - applications can register with the audit service at any time
  - simplifies the ability of applications to leverage the audit framework to define and log audit events
  - provides versioning of event definitions and enables audit clients to upgrade definitions independent of release cycles.

13.3 Understanding Oracle Fusion Middleware Audit Framework Concepts

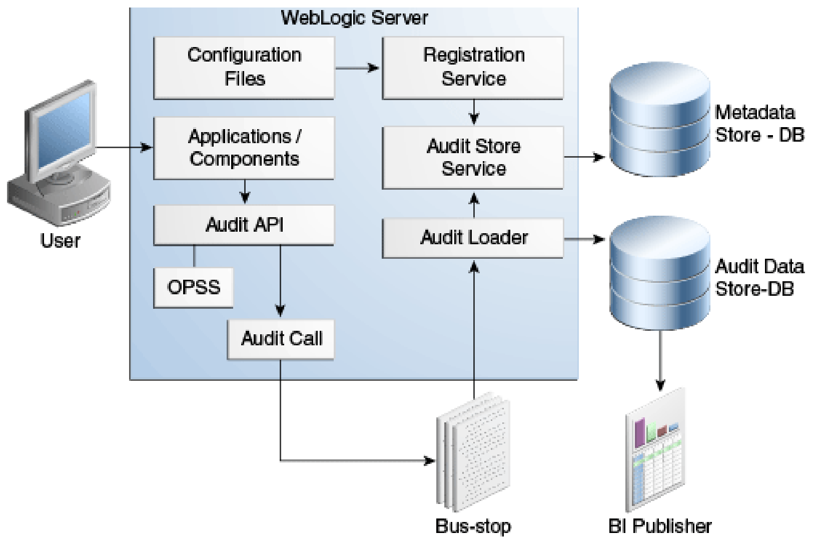
This section introduces basic principles of the Oracle Fusion Middleware Audit Framework:

- About the Audit Architecture
- Understanding Key Technical Concepts of Auditing
- About the Audit Metadata Store
- How Audit Data is Stored
- About Audit Analytics

13.3.1 About the Audit Architecture

Figure 13–1 shows the architecture of the Oracle Fusion Middleware Audit Framework:

Figure 13–1 Audit Architecture



The figure illustrates essential elements of the architecture, including development and run-time services, the audit event flow, and key features, which are described in the subsequent section.

13.3.1.1 The Audit Service Model

At the heart of the framework is the audit service model, which like other OPSS services, provides a standards-based, integrated framework for Java EE and SE applications and Oracle components alike.

Used across Oracle Fusion Middleware, this model enables you to conveniently leverage the platform's audit features, and supports key requirements of the enterprise.

Dynamic Audit Metadata

The audit service features a dynamic audit metadata model that allows audit clients to manage audit event definitions and make version changes independent of build and release cycles. Audit event definitions can be dynamically updated at redeployment.

Application Lifecycle Support

The model supports all aspects of the application lifecycle from design to development to deployment.

Application Registration Service

A versatile registration service enables you to register your applications to the audit service:

- declaratively; Java EE applications do so by packaging an XML configuration in ear files in the META-INF directory at deployment

  See Also: Section 23.4.1.

- programatically, by invoking the audit registration API
- at the command line, through WLST commands

Seel Also: "registerAudit" in the *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

- through bundling of security artifacts in a product template

See Also: Section 7.1

Distributed Environments

The audit service supports distributed environments with multiple servers, that is, an administration server and one or more managed servers. The audit service monitors the audit metadata store so changes in audit policy at one server are synchronized with the other servers without the need for any manual action.

Consider, for example, a distributed environment consisting of an administration server and three managed servers. A single security store that includes an audit metadata store supports all the servers. If you change audit policy at the administration server through Fusion Middleware Control, the changes are automatically propagated to synchronize with the remaining servers.

13.3.1.2 About the Audit APIs

Audit APIs are provided by the audit framework for any audit-aware components integrating with the Oracle Fusion Middleware Audit Framework. During runtime, applications may call these APIs to manage audit policies and to audit the necessary information about a particular event happening in the application code. The interface allows applications to specify event details such as username and other attributes needed to provide the context of the event being audited.

The audit framework provides these APIs:

- audit service API
- audit client API
- audit administration service API

See Chapter 23 for details.

13.3.1.3 Understanding Run-time Support and Audit Event Flow

The process can be illustrated by looking at the actions taken in the framework when an auditable event (say, login) occurs within an application server instance:

Note: The architecture shown in Figure 13–1 contains an audit data store; if your site did not configure an audit data store, the audit records reside in the bus-stop files.

1. During application deployment or audit service start-up, a client such as a Java EE application or Oracle component registers with the audit service.
2. The service reads the application’s pre-configured audit definition file and updates the metadata store with the audit definitions.
3. When a user accesses the component or application, an audit API function is called to audit the event.
4. The audit framework checks if events of this type, status, and with certain attributes need to be audited.

5. If so, the audit function is invoked to create the audit event structure and collect event information like the status, initiator, resource, ECID, and so on.
6. The event is stored on a local file in an intermediate location known as the bus-stop; each component has its own bus-stop.
7. If a database is configured for an audit store, the audit loader pulls the events from the bus-stops, uses the application's metadata to format the data, and moves the data to the audit store.
8. Reports can also be generated from the audit data. (See Chapter 15, "Using Audit Analysis and Reporting".)

Application Behavior in Case of Audit Failure

It is important to note that an application does not stop execution if it is unable to record an audit event for any reason.

13.3.2 Understanding Key Technical Concepts of Auditing

This section introduces key concepts in the Oracle Fusion Middleware Audit Framework.

Audit-Aware Components

The term "audit-aware" refers to components that are integrated with the Oracle Fusion Middleware Audit Framework so that audit policies can be configured and events can be audited for those components. Oracle HTTP Server is an example of an audit-aware component.

Audit Metadata Store

The audit metadata store contains audit event definitions for components and for applications integrated with the audit framework.

For details, see Section 13.3.3.

Audit Data Store

The audit data store contains audit event records. It is a database that contains a pre-defined Oracle Fusion Middleware Audit Framework schema, created by Repository Creation Utility (RCU). Once the audit data store is configured, the audit loader is aware of the store and uploads data to it periodically. The audit data in the store is expected to be cumulative and will grow over time. Ideally, this should not be an operational database used by any other applications - rather, it should be a standalone database used for audit purposes only.

The audit database can store audit events generated by Oracle components as well as user applications integrated with the audit framework.

Note: The audit metadata store is separate from the audit data store.

For details about the data store, see Section 13.3.4. For details about using RCU to create the audit schema, see Section 14.2.1.

Audit Definition Files

Applications use audit definition files to specify to the audit service the auditing rules (events, filters, and so on) that will govern audit actions.

For details, see Section 13.5.

Audit Events

The Oracle Fusion Middleware Audit Framework provides a set of generic events for convenient mapping to application audit events. Some of these include common events such as authentication. The framework also allows applications to define application-specific events.

These event definitions and configurations are implemented as part of the audit service in Oracle Platform Security Services. Configurations can be updated through Enterprise Manager (UI) and WLST (command-line tool).

Audit Loader

The Audit Loader is a module of the Oracle WebLogic Server instance and supports audit activity in that instance. When an audit data store is configured, audit loader is responsible for collecting the audit records for all components running in that instance and loading them to the data store.

For Java components, the audit loader is started as part of the container start-up.

System components can either register to the audit store (using the `registerAudit()` WLST command) to upload their events through the container audit loader, or use the command-line stand-alone audit loader.

> **See Also:** Section 14.2.4.2 for details about the stand-alone audit loader.

Audit Policy

An audit policy is a declaration of the type of events to be captured by the audit framework for a particular component. The audit policy can be defined at either the component level (so that it applies to a specific component such as Oracle HTTP Server), or the domain level (so that it applies to specified activity in the domain).

Oracle Fusion Middleware Audit Framework provides several pre-defined policy types:

- None (audit is disabled for the component)
- Low
- Medium
- High
- Custom (implements filters to narrow the scope of audited events); applies at the component level only.

Audit Policy Component Type

This refers to the component type to be audited; for example, Oracle HTTP Server is a source of auditable events during authentication.

For lists of the events that can be audited for each component, see Section C.1, "Audit Events".

Audit Service

The audit service model provides a standards-based, integrated framework for Java EE and SE applications to leverage audit features.

Bus-stop

Bus-stops are local files containing audit data records before they are pushed to the audit data store. In the event that no audit data store is configured, audit data remains in these bus-stop files. The bus-stop files are simple text files that can be queried easily to look up specific audit events. When an audit data store is in place, the bus-stop acts as an intermediary between the component and the audit data store. The local files are periodically uploaded to the audit data store based on a configurable time interval.

A key advantage of the audit data store is that audit data from multiple components can be correlated and combined in reports, for example, authentication failures in all middleware components, instances and so on.

Event Filters

Certain audit events implement filters to control when the event is logged. For example, a successful login event for the Oracle Internet Directory component may be filtered for specific users.

For details, see Section 14.3, "Managing Audit Policies".

Configuration MBeans

All audit configuration is managed through audit configuration MBeans. For Java components and applications, these MBeans are present in the domain administration server and the audit configuration is centrally managed. For system components, separate MBean instances are present for every component instance. Enterprise Manager UI and command-line tools manage Audit configuration using these MBeans.

Oracle Platform Security Services

Oracle Platform Security Services, a key component of Oracle Fusion Middleware, is the Oracle Fusion Middleware security implementation for Java features such as Java Authentication and Authorization Service (JAAS) and Java EE security.

For more information about OPSS, see Section 1.1, "What is Oracle Platform Security Services?".

13.3.3 About the Audit Metadata Store

About the Audit Metadata Store

The audit metadata store provides the storage for artifacts in the metadata model and contains component audit definitions, NLS translation entries, runtime policies, and database mapping tables.

Note: The audit metadata store is separate from the audit data store, which contains the actual audit records.

The audit metadata store supports several critical auditing functions:

- The audit registration service:
  - creates, modifies, and deletes event definition entries
  - creates attribute database mappings to store audit data
- The audit runtime service retrieves event definitions and runtime policies.

- Audit MBean commands look up and modify component audit definitions and runtime policies.

Audit Metadata Store Requires Oracle Database

The audit framework requires Oracle Database as the storage for the metadata.

When a new application registers to the audit service, the following audit artifacts are stored in the audit store:

- audit event definitions including custom attribute group, categories, events, and filter preset definitions
- localized translation entries
- custom attribute-database column mapping table
- run-time audit policies

13.3.4 How Audit Data is Stored

As shown in Figure 13–1, audit data can reside in two types of storage:

- bus-stop files for intermediate storage of audit data. Each component instance writes to its own bus-stop.

  There is a separate bus-stop file for each component.

  Bus-stop files are text-based and easy to query. For further details, see Section 13.3.1, "About the Audit Architecture"
- permanent storage in a database; this is known as the audit data store.

  If using a database, audit records generated by all components in all Oracle Fusion Middleware instances in the domain are written to the same store.

You can migrate from a file-based security store to a database store. This requires a specific configuration procedure. See Section 7.5.5 for details.

Advantages of Using a Database Store

Having the audit records in the bus-stop files has some practical limitations:

- you cannot view domain-level audit data
- reports are difficult to obtain

Thus, there are certain advantages to using a database audit data store:

- You can write audit reports.
- The database store centralizes records from all components in the domain, whereas the bus-stop stores audit records on a per-instance basis.
- performance may be improved compared to file-based storage

Thus, a database is the default audit data store out-of-the-box.

13.3.5 About Audit Analytics

The Common Audit Framework enables audit data to be retrieved efficiently. You can use the audit schema details to create custom audit reports as needed.

For further details, see Section C.2

13.3.6 Understanding the Audit Lifecycle

For an administrator auditing into an environment consisting of Oracle Fusion Middleware components and applications, the key phases of implementing a functional audit system are as follows:

- Understand the architecture of the Common Audit Framework.

  For a review of the essential elements of the framework, the flow of actions, and the features of the audit service model, see Section 13.3.1.

- Integrate applications with the audit framework.

  For details about integration steps, see Section 23.2.

- Create the audit definition files that describe your application's audit events and how they map to the audit schema.

  For details on this topic, see Section 23.3.

- Register the application with the audit service.

  There are various ways to enable registration. For details, see Section 23.4.

- Migrate audit information from test to production.

  See Section 6.6.3.

- Generate audit reports.

  Descriptions of the tools and techniques of audit reporting appear in Chapter 15 and Appendix C.

13.4 About the Audit Metadata Model

The audit framework supports a metadata model which enables applications to specify their audit artifacts in a flexible manner. Applications can dynamically define attribute groups, categories, and events.

Topics include:

- Naming Conventions for Audit Artifacts
- About Attribute Groups
- Understanding Event Categories and Events

13.4.1 Naming Conventions for Audit Artifacts

The following naming restrictions apply to category, event and attribute names:

- They must be in English only
- They must be less than 26 characters
- They can contain only the characters a through z, A through Z, and numbers 0 through 9
- They must start with a letter.

13.4.2 About Attribute Groups

Attribute groups provide broad classification of audit attributes and consist of three types:

- The common attribute group contains all the system attributes common to all applications, such as component type, system IP address, hostname, and others.

  The IAU_COMMON database table contains attributes in this group.
- Generic attribute groups contain attributes for audit application areas such as authentication and user provisioning.
- Custom attribute groups are those defined by an application to meet its specific needs. Attributes in a custom group are limited to that component scope.

See Also: Table C–5 for a list of common OPSS attributes.

13.4.2.1 About Audit Attribute Data Types

Table 13–1 shows the supported attribute data types and the corresponding Java object types:

Table 13–1 Audit Attribute Data Types

Attribute Data Type	Java Object Type	Notes
Integer	Integer	
Long	Long	
Float	Float	
Double	Double	
Boolean	Boolean	
DateTime	java.util.Date	
String	String	Maximum length 2048 bytes
LongString	String	Unlimited length
Binary	byte[]	

13.4.2.2 About Common Attribute Groups

The common attribute group is stored in the `IAU_COMMON` database table.

For details, see Table C–5.

13.4.2.3 About Generic Attribute Groups

A generic attribute group is defined with a namespace, a version number, and one or more attributes. This example defines an attribute group with namespace `authorization` and version 1.0:

```
<AuditConfig xmlns="http://xmlns.oracle.com/ias/audit/audit-2.0.xsd" >
    <Attributes ns="authorization" version="1.0">
        <Attribute displayName="CodeSource" maxLength="2048" name="CodeSource"
type="string"/>
        <Attribute displayName="Principals" maxLength="1024" name="Principals"
type="string"/>
        <Attribute displayName="InitiatorGUID" maxLength="1024"
name="InitiatorGUID" type="string"/>
        <Attribute displayName="Subject" maxLength="1024" name="Subject"
type="string">
          <HelpText>Used for subject in authorization</HelpText>
        </Attribute>
    </Attributes>
……
```

Your application can reference the `CodeSource` attribute like this:

```
<Attribute name="CodeSource" ns="authorization" version="1.0" />
```

Each generic attribute group is stored in a dedicated database table. The naming conventions are:

- `IAU_`*`GENERIC_ATTRIBUTE_GROUP_NAME`* for table names
- `IAU_`*`ATTRIBUTE_NAME`* for table columns

For example, the attribute group `authorization` is stored in database table `IAU_AUTHORIZATION` with these columns:

- `IAU_CODESOURCE` as string
- `IAU_PRINCIPALS` as string
- `IAU_INITIATORGUID` as string

13.4.2.4 About Custom Attribute Groups

A custom attribute group is defined with a namespace, a version number, and one or more attributes.

Attributes consist of:

- attribute name
- data type
- attribute-database column mapping order - This property specifies the order in which an attribute is mapped to a database column of a specific data type in the custom attribute table.
- help text (optional)
- maximum length
- display name
- size - This property denotes how many values of the same data type the attribute can have. The default size value is 1. A size greater than 1 denotes a multi-valued attribute which can have two or more values of the same data type. The multi-value attribute supports all data types except for binary.

This example defines attribute group `Accounting` with namespace `accounting` and version 1.0:

```
<Attributes ns="accounting" version="1.0">

            <Attribute name="TransactionType" displayName="Transaction Type"
type="string" order="1"/>
            <Attribute name="AccountNumber" displayName="Account Number"
type="int" order="2">
                <HelpText>Account number.</HelpText>
            </Attribute>
            ……
        </Attributes>
```

The following example shows a multi-valued attribute named AccountBalance:

```
<Attribute order="3" displayName="AccountBalance" type="double" name="Balance"
size="2" sinceVersion="1.1">
            <MultiValues>
```

```
                <MultiValueName displayName="Previous Balance" index="0">
                     <HelpText>the previous account balance</HelpText>
                </MultiValueName>
                <MultiValueName displayName="Current Balance" index="1"/>
            </MultiValues>
</Attribute>
```

Custom attribute groups and attributes are stored in the IAU_CUSTOM*n* table, where n=1,2,3...

13.4.3 Understanding Event Categories and Events

An audit event category contains related events in a functional area. For example, a session category could contain login and logout events that are significant in a user session's lifecycle.

An event category does not itself define attributes. Instead, it references attributes in component and system attribute groups.

There are two types of event categories:

- System Categories
- Component and Application Categories

See Also: Table C–1 for definitions of system categories and events.

13.4.3.1 About System Categories and Events

A system category references common and generic attribute groups and contains audit events. System categories are the base set of component event categories and events. Applications can reference them directly, log audit events, and set filter preset definitions.

See Also: Table C–1.

The following example shows several elements of the metadata model:

- common attribute group
- generic attribute groups `identity` and `authorization`
- system category `UserSession` with an attribute referencing to a common attribute AuthenticationMethod
- audit events such as UserLogin and UserLogout

```
<SystemComponent major="1" minor="0">
+<Attributes ns="common" version ="1.0"></Attributes>
+<Attributes ns="identity" version ="1.0"></Attributes>
+<Attributes ns="authorization" version ="1.0"></Attributes>
-<Events>
 -<Category name="UserSession" displayName="User Sessions">
  -<Attributes>
     <Attribute name="AuthenticationMethod" ns="common" version ="1.0" />
   </Attributes>
  -<HelpText></HelpText>
  -<Event name="UserLogin" displayName="User Logins" shortName="uLogin"></Event>
  -<Event name="UserLogout" displayName="User Logouts" shortName="uLogout"
    xdasName="terminateSession"></Event>
  -<Event name="Authentication" displayName="Authentication"></Event>
  -<Event name="InternalLogin" displayName="Internal Login" shortName="iLogin"
```

```
    xdasName="CreateSession"></Event>
  -<Event name="InternalLogout" displayName="Internal Logout" shortName="iLogout"
    xdasName="terminateSession"></Event>
  -<Event name="QuerySession" displayName="Query Session"
    shortName="qSession"></Event>
  -<Event name="ModifySession" displayName="Modify Session"
    shortName="mSession"></Event>
  </Category>
 +<Category displayName="Authorization" name="Authorization"></Category>
 +<Category displayName="ServiceManagement" name="ServiceManagement"></Category>
 </Events>
</SystemComponent>
```

13.4.3.2 About Component/Application Categories

A component or application can extend system categories or define new component event categories. In this example, a transaction category references attributes AccountNumber, Date, and Amount from the `accounting` attribute group, and includes events 'purchase' and 'deposit':

```
 <Category displayName="Transaction" name="Transaction">
    <Attributes>
       <Attribute name="AccountNumber" ns="accounting" version="1.0"/>
       <Attribute name="Date" ns="accounting" version="1.0" />
       <Attribute name="Amount" ns="accounting" version="1.0" />
 </Attributes>

    <Event displayName="purchase" name="purchase"/>
    <Event displayName="deposit" name="deposit">
       <HelpText>depositing funds.</HelpText>
    </Event>
……
 </Category>
```

You extend system categories by creating category references in your application audit definitions. List the system events that the system category includes, and add new attribute references and events to it.

In this example, a new category references a system category `ServiceManagement` with a new attribute reference `ServiceTime`, and a new event `restartService`:

```
<CategoryRef name="ServiceManagement" componentType="SystemComponent">
                <Attributes>
                    <Attribute name="ServiceTime" ns="accounting" version="1.0" />
                </Attributes>

                <EventRef name="startService"/>
                <EventRef name="stopService"/>

                <Event displayName="restartService" name="restartService">
                    <HelpText>restart service</HelpText>
                </Event>

</CategoryRef>
```

13.5 About Audit Definition Files

There are two types of definition files:

- component_events.xml definition file
- translation files

13.5.1 About the component_events.xml File

The component_events.xml file specifies the properties the audit service needs to log audit events, including:

- basic properties
  - the component type, which is the property that applications use to register with the audit service and obtain runtime auditor instances
  - Major and minor version of the application.
- at most one custom attribute group
- event categories with attribute references and events
- component level filter definitions
- runtime policies, which include:

Here is an example component_events.xml file:

```
<?xml version="1.0"?>
<AuditConfig xmlns="http://xmlns.oracle.com/ias/audit/audit-2.0.xsd">
    <AuditComponent componentType="ApplicationAudit" major="1" minor="0">
        <Attributes ns="accounting" version="1.0">

            <Attribute name="TransactionType" displayName="Transaction Type"
type="string" order="1">
               <HelpText>Transaction type.</HelpText>
           </Attribute>
            <Attribute name="AccountNumber" displayName="Account Number"
type="int" order="2">
                <HelpText>Account number.</HelpText>
            </Attribute>
            <Attribute name="Date" displayName="Date" type="dateTime" order="3"/>
            <Attribute name="Amount" displayName="Amount" type="float" order="4">
                <HelpText>Transaction amount.</HelpText>
            </Attribute>
            <Attribute name="Status" displayName="Account Status" type="string"
order="5">
                <HelpText>Account status.</HelpText>
            </Attribute>

        </Attributes>

        <Events>
            <Category displayName="Transaction" name="Transaction">
                <Attributes>
                    <Attribute name="AccountNumber" ns="accounting" version="1.0"
/>
                    <Attribute name="Date" ns="accounting" version="1.0" />
                    <Attribute name="Amount" ns="accounting" version="1.0" />
                </Attributes>

                <Event displayName="purchase" name="purchase">
                    <HelpText>direct purchase.</HelpText>
                </Event>
                <Event displayName="deposit" name="deposit">
```

```
                    <HelpText>depositing funds.</HelpText>
                </Event>
                <Event displayName="withdrawing" name="withdrawing">
                    <HelpText>withdrawing funds.</HelpText>
                </Event>
                <Event displayName="payment" name="payment">
                    <HelpText>paying bills.</HelpText>
                </Event>
            </Category>
            <Category displayName="Account" name="Account">
                <Attributes>
                    <Attribute name="AccountNumber" ns="accounting" version="1.0"
/>
                    <Attribute name="Status" ns="accounting" version="1.0" />
                </Attributes>

                <Event displayName="open" name="open">
                    <HelpText>Open a new account.</HelpText>
                </Event>
                <Event displayName="close" name="close">
                    <HelpText>Close an account.</HelpText>
                </Event>
                <Event displayName="suspend" name="suspend">
                    <HelpText>Suspend an account.</HelpText>
                </Event>
            </Category>
        </Events>
        <FilterPresetDefinitions>
            <FilterPresetDefinition displayName="Low" helpText="" name="Low">
                <FilterCategory enabled="partial"
name="Transaction">deposit.SUCCESSESONLY(HostId -eq
&quot;NorthEast&quot;),withdrawing</FilterCategory>
                <FilterCategory enabled="partial"
name="Account">open.SUCCESSESONLY,close.FAILURESONLY</FilterCategory>
            </FilterPresetDefinition>
            <FilterPresetDefinition displayName="Medium" helpText=""
name="Medium">
                <FilterCategory enabled="partial"
name="Transaction">deposit,withdrawing</FilterCategory>
                <FilterCategory enabled="partial"
name="Account">open,close</FilterCategory>
            </FilterPresetDefinition>
            <FilterPresetDefinition displayName="High" helpText="" name="High">
                <FilterCategory enabled="partial"
name="Transaction">deposit,withdrawing,payment</FilterCategory>
                <FilterCategory enabled="true" name="Account"/>
            </FilterPresetDefinition>
        </FilterPresetDefinitions>

        <Policy filterPreset="Low">
            <CustomFilters>
                <FilterCategory enabled="partial"
name="Transaction">purchase</FilterCategory>
            </CustomFilters>

        </Policy>

    </AuditComponent>
</AuditConfig>
```

About Runtime Properties

Supplementing the properties described above is a set of runtime properties, which are created through Fusion Middleware Control, WLST commands, or by defaults set during registration. These properties include:

- filterPreset - specifies the audit filter level
- specialUsers - specifies the users to always audit
- maxBusstopFileSize

For details about run-time policies, see Section 14.3

13.5.2 Overview of Translation Files

Translation files are used to display audit definition in different languages.

For details, see Section 23.3.2.

13.5.3 About Mapping and Versioning Rules

The registration service uses certain rules to create the audit metadata for the application. This metadata is used to maintain different versions of the audit definition, and to load audit data and generate reports.

13.5.3.1 What are Version Numbers?

Each audit definition must have a major and a minor version number, which are integers, for example, major = 1 minor=3. Any change to an audit event definition requires that the version ID be modified by changing the minor and/or major number.

Version numbers are used by the audit registration service to determine the compatibility of event definitions and attribute mappings between versions.

Note: These version numbers have no relation to Oracle Fusion Middleware version numbers.

Versioning for Oracle Components

When registering an Oracle component, the audit registration service checks if this is a first-time registration or an upgrade.

For a new registration, the service:

1. retrieves the component audit and translation information.
2. parses and validates the definition, and stores it in the audit metadata store.
3. generates the attribute-column mapping table, and saves this in the audit metadata store.

For upgrades, the current major and minor numbers for the component in the metadata store are compared to the new major and minor numbers to determine whether to proceed with the upgrade.

Versioning for JavaEE Applications

When modifying your application’s audit definition, it is recommended that you set the major and minor numbers as follows:

- Only increase the minor version number when making version-compatible changes, meaning changes in an audit definition such that the attribute database mapping table generated from the new audit definition should still work with the audit data created by the previous attribute database mapping table.

For example, suppose the current definition version is major=2 and minor=1. When adding a new event that does not affect the attribute database mapping table, you can change the minor version to 2 (minor=2), while the major version remains unchanged (major=2). Likewise, you can increase the minor version after adding a new attribute.

- Increase major version number when making version changes where the new mapping table is incompatible with the previous table.

The change becomes effective after server restart.

13.5.3.2 About Custom Attribute to Database Column Mappings

When registering a new component or application, the registration service creates an attribute-to-database column mapping table from the component's custom attributes, and then saves this table to the audit metadata store.

Attribute-database mapping tables are required to ensure unique mappings between your application's attribute definitions and database columns. The audit loader uses mapping tables to load data into the audit store; the tables are also used to generate audit reports from custom database table IAU_CUSTOM.

Viewing Audit Definitions for your Component

You can use the WLST command `createAuditDBView` to create a database view of the audit definitions of your component.

For details about command syntax, see Appendix C, "Oracle Fusion Middleware Audit Framework Reference.".

Using the `createAuditDBView` command requires an understanding of how your component's attributes are mapped to database columns; see the discussion below for details.

Understanding the Mapping Table for your Component

A custom attribute-database column mapping has properties of attribute name, database column name, and data type.

Each custom attribute must have a mapping order number in its definition. Attributes with the same data type are mapped to the database column in the sequence of attribute mapping order. For example, if the definition file looks like this:

```
<Attributes ns="accounting" version="1.1">
   <Attribute name="TransactionType" type="string" maxLength="0"
    displayName="Transaction Type" order="1"/>
   <Attribute name="AccountNumber" type="int" displayName="Account Number"
    order="2">
   <Attribute name="Date" type="dateTime" displayName="Date" order="3"/>
   <Attribute name="Amount" type="float" displayName="Amount" order="4"/>
   <Attribute name="Status" type="string" maxLength="0" displayName="Account
    Status" order="5"/>
   <Attribute name="Balance" type="float" displayName="Account Balance"
    order="6"/>
</Attributes>
```

then the mapping is as follows:

```
<AttributesMapping ns="accounting" tableName="IAU_CUSTOM" version="1.1">
   <AttributeColumn attribute="TransactionType" column="IAU_STRING_001"
```

```
    datatype="string"/>
   <AttributeColumn attribute="AccountNumber" column="IAU_INT_001"
    datatype="int"/>
   <AttributeColumn attribute="Date" column="IAU_DATETIME_001"
    datatype="dateTime"/>
   <AttributeColumn attribute="Amount" column="IAU_FLOAT_001" datatype="float"/>
   <AttributeColumn attribute="Status" column="IAU_STRING_002" datatype="string"/>
   <AttributeColumn attribute="Balance" column="IAU_FLOAT_002" datatype="float"/>
</AttributesMapping>
```

The version ID of the attribute-database column mapping table matches the version ID of the custom attribute group. This allows your application to maintain the backward compatibility of attribute mappings across audit definition versions. For more information about versioning, see Section 13.5.3.1.

14 Configuring and Managing Auditing

This chapter explains how to perform day-to-day audit administration tasks.

> **See Also:** Chapter 13, "Introduction to Oracle Fusion Middleware Audit Service" for background information about auditing in Oracle Fusion Middleware.

14.1 Understanding Audit Administration Tasks

As audit administrator, you should plan the site's audit setup carefully by following the steps in these areas:

- Implementation Planning

  This includes planning the type of store to use for audit records, data store configuration details, and so on.

  See Section 14.2, "Managing the Audit Data Store" for details.

- Policy Administration

  You must configure the appropriate audit policies to ensure that the required audit events are generated.

  This is an ongoing activity since the audit policies must be able to reflect changes to the application environment, addition of components and users, and so on.

  See Section 14.3, "Managing Audit Policies" for details.

- Reports Management

  This includes planning for and configuring audit reports and queries.

  See Chapter 15, "Using Audit Analysis and Reporting" for details.

- Data Administration

This includes planning/increasing the database size required to store the audit data generated, backing up the audit data and, purging the audit data based on company policy.

See Section 14.6, "Advanced Administration of the Database Store" for details about audit data store administration.

14.2 Managing the Audit Data Store

In a production environment, a database audit data store provides the scalability and high-availability needed to store audit records.

In addition, an audit data store residing in a database allows the audit data to be viewed through query and reporting tools.

This section explains these audit data store management tasks in detail:

- About the Audit Schema
- About Audit Data Sources
- Tuning the Bus-stop Files
- Configuring the Stand-alone Audit Loader

14.2.1 About the Audit Schema

The audit schema (IAU schema) is a database structure for storing audit records, and is created along with the security store (OPSS schema).

For details of security schema creation, see Chapter 9, "Configuring the OPSS Security Store".

Note: The bus-stop files store audit records in the absence of database storage.

Once the database schema is created, you can update the domain configuration to switch the audit data store for audit records.

14.2.2 About Audit Data Sources

In WebLogic Server, you configure database connectivity by adding data sources to your WebLogic domain. WebLogic JDBC data sources provide database access and database connection management.

As explained in Section 14.2.1, domain creation generates the audit schema, a data structure required to store audit records in a database. It also sets up an Oracle WebLogic Server audit data source that points to the audit schema. The audit data source thus enables you to directly query and run reports against the audit records in the database.

For details about domain creation in the OPSS context, see Section 7.3, "Creating an FMW Domain".

14.2.3 Tuning the Bus-stop Files

Once the bus-stop file reaches a certain size and all the data is uploaded to the database, the audit loader deletes the file from the operating system. Tuning the bus-stop files enables you to efficiently manage bus-stop files.

This section contains topics related to maintaining the bus-stop files that store audit records, including:

- bus-stop file locations
- file size

> **Note:** Manually purging audit files to free up space is not recommended. Instead, use file sizing features to control space, as described below.

14.2.3.1 Locating the Bus-Stop Files

Bus-stop files for Java components are located in:

```
$DOMAIN_HOME/servers/$SERVER_NAME/logs/auditlogs/Component_Type
```

Bus-stop files for system components are located in:

```
$ORACLE_INSTANCE/auditlogs/Component_Type/Component_Name
```

14.2.3.2 Sizing Bus-Stop Files

Java Components

The size of a file for the file storage mode can be managed using the `audit.maxFileSize` property described in the configuration file `jps-config.xml`. This property controls the maximum size of a bus-stop file for Java components.

Specify the sizes in bytes.

System Components

The size of a bus-stop file can be set in the `auditconfig.xml` file. For example:

```
<serviceInstance name="audit" provider="audit.provider">
      <property name="audit.maxFileSize" value="10240" />
      <property name=" audit.loader.repositoryType " value="Db" />
   </serviceInstance>
```

> **Note:** If you switch from file to database store for audit data, all the events collected in the audit files are pushed into the database tables and the audit files are deleted.

14.2.4 Configuring the Stand-alone Audit Loader

As shown in Figure 13–1, Common Audit Framework's audit loader moves records from bus-stop files to the audit data store. The mechanism driving the audit loader depends on the application environment:

- Java EE components and applications deployed on Oracle WebLogic Server use the audit loader functionality provided through the application server. Although you can run the stand-alone audit loader, it is not needed in the JavaEE environment.
- System components and non-Java applications use the audit loader functionality provided through the StandAloneAuditLoader java command.

- Java SE applications, which run outside an application server container, also use the stand-alone audit loader.

This section explains how to set up and execute the stand-alone audit loader:

- Configuring the Environment
- Running the Stand-Alone Audit Loader

14.2.4.1 Configuring the Environment

Before you can run the stand-alone audit loader, you must a) configure certain properties, and b) ensure that the password for the database schema user exists in the secret store.

14.2.4.1.1 Configuring Properties for the Audit Loader

You must configure the following properties:

- `ORACLE_HOME` environment variable. Set to the full path of the Oracle home.
- `COMMON_COMPONENTS_HOME` environment variable. Set to the location containing the Java Required Files (JRF).
- `ORACLE_INSTANCE` environment variable. Set to the full path of an Oracle instance.
- `auditloader.jdbcString` system property. Set to the JDBC connection string for the database where audit data resides.
- `auditloader.username` system property. Set to the username of a user who can run the audit loader.

14.2.4.1.2 Storing the Password for the Database Schema User The password for the database schema user is kept in the secret store. Storing the password is a one-time operation for which you use the java `StandAloneAuditLoader` command with the `-Dstore.password=true` property.

Issue the `StandAloneAuditLoader` command to store the password as follows:

```
$JDK_HOME/bin/java
-classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.1.3/jps-manifest.jar
-Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
-Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
-Dauditloader.username=username
oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

You will be prompted for the password.

Note: There should not be any spaces in the classpath specification.

This command generates a cwallet.sso file.

14.2.4.2 Running the Stand-Alone Audit Loader

Issue the `StandAloneAuditLoader` command to load audit records as follows:

```
$JDK_HOME/bin/java
-classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.1.3/jps-manifest.jar
-Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
-Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
-Dauditloader.username=username
```

```
oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

You can schedule this command through a batch or kron job so that audit records are periodically uploaded to the audit data store.

14.3 Managing Audit Policies

What is an Audit Policy?

An audit policy is a declaration of the type of events to be captured by the audit framework for a particular component. For Java components, the audit policy is defined at the domain level. For system components, the audit policy is managed at the component instance level.

For example, an audit policy could specify that all authentication failures should be audited for an Oracle Internet Directory instance.

How Policies are Configured

Oracle Fusion Middleware Audit Framework lets you configure audit policies and provides highly granular controls over the types of events and data being audited. You can configure policies through the Oracle Enterprise Manager Fusion Middleware Control UI tool or with WLST commands.

Policy changes do not require server or instance restart.

The remainder of this section explains how to view, and update audit policy:

- Managing Audit Policies for Java Components with Fusion Middleware Control
- Managing Audit Policies for System Components with Fusion Middleware Control
- Managing Audit Policies at the Command Line (WLST)
- Managing Audit Policies through API Programming

See Also:

- Section 13.3.2, "Understanding Key Technical Concepts of Auditing" for additional background.
- Appendix C for a list of Java components and system components.

14.3.1 Managing Audit Policies for Java Components with Fusion Middleware Control

The domain Audit Policy Settings page manages audit events for all Java components such as Oracle Access Manager, and system libraries like Oracle Platform Security Services.

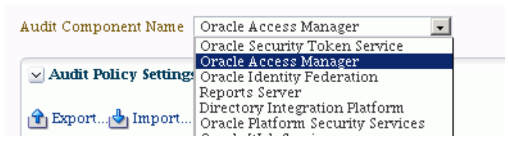
Notes:

- Audit policy for system components is managed in the component home pages. See Section 14.3.2, "Managing Audit Policies for System Components with Fusion Middleware Control"
- See the note at the beginning of Section 14.3, "Managing Audit Policies" titled "Policy Changes Require Server or Instance Restart".

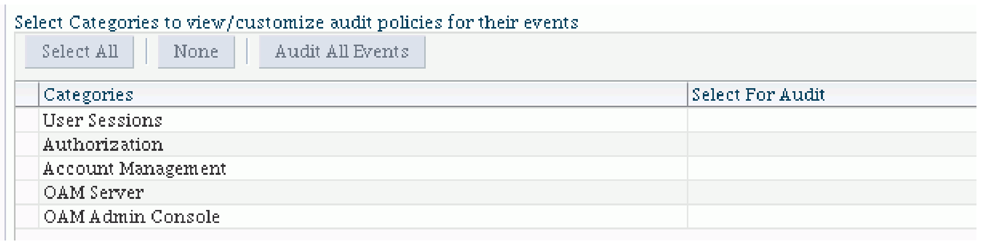
> **See Also :** Section C.1.1, "What Components Can be Audited?" for the list of auditable components.

Use these steps to view and update the currently configured audit policies:

1. Log in to Fusion Middleware Control.
2. Using the topology panel to the left, navigate to the domain of interest under "WebLogic Domain".
3. From the domain menu, navigate to `Domain`, then **Security**, then **Audit Policy**. The Audit Policy page appears.
4. Use the Audit Component Name drop-down to select the Java component whose audit policy you wish to configure.


Audit Component Name
Oracle Access Manager
Oracle Security Token Service
Oracle Access Manager
Oracle Identity Federation
Reports Server
Directory Integration Platform
Oracle Platform Security Services
Audit Policy Setting
Export...
Import...

   When you select a component, a table of audit categories relevant to the component appears in the area of the page titled Audit Policy Settings. The following example is for Oracle Access Manager:

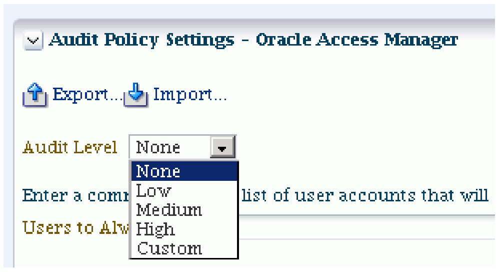


Select Categories to view/customize audit policies for their events

Select All | None | Audit All Events

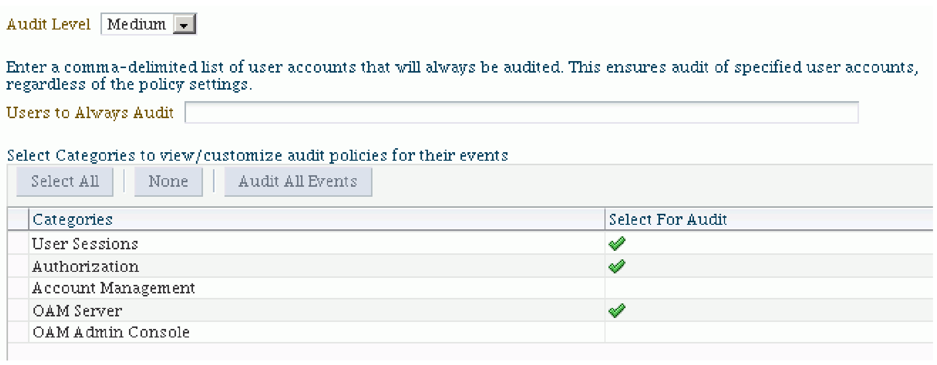
Categories	Select For Audit
User Sessions	
Authorization	
Account Management	
OAM Server	
OAM Admin Console	

5. Use the Audit Level drop-down to select a subset of the audit events for the component. The choices are:



   - None - No event categories selected.
   - Low, Medium, High - Subsets of event categories representing pre-defined levels of auditing.
   - Custom - Enables you to create or modify a custom filter definition.

   Depending on the level you select, check flags appear in the column Select for Audit. Events in the checked categories will be audited. For example, at the Medium level for Oracle Access Manager:



Audit Level: Medium

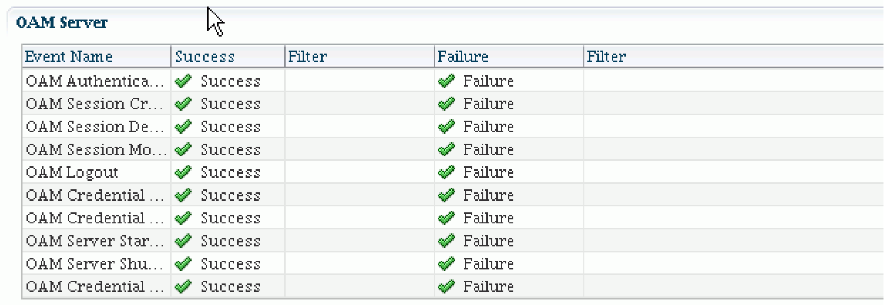
Enter a comma-delimited list of user accounts that will always be audited. This ensures audit of specified user accounts, regardless of the policy settings.

Users to Always Audit

Select Categories to view/customize audit policies for their events

Categories	Select For Audit
User Sessions	✓
Authorization	✓
Account Management	
OAM Server	✓
OAM Admin Console	

The flagged categories are selected for audit. You can view the events within a flagged category by clicking on that row in the categories table. For example, clicking on the "OAM Server" category produces the following event table below it:



OAM Server

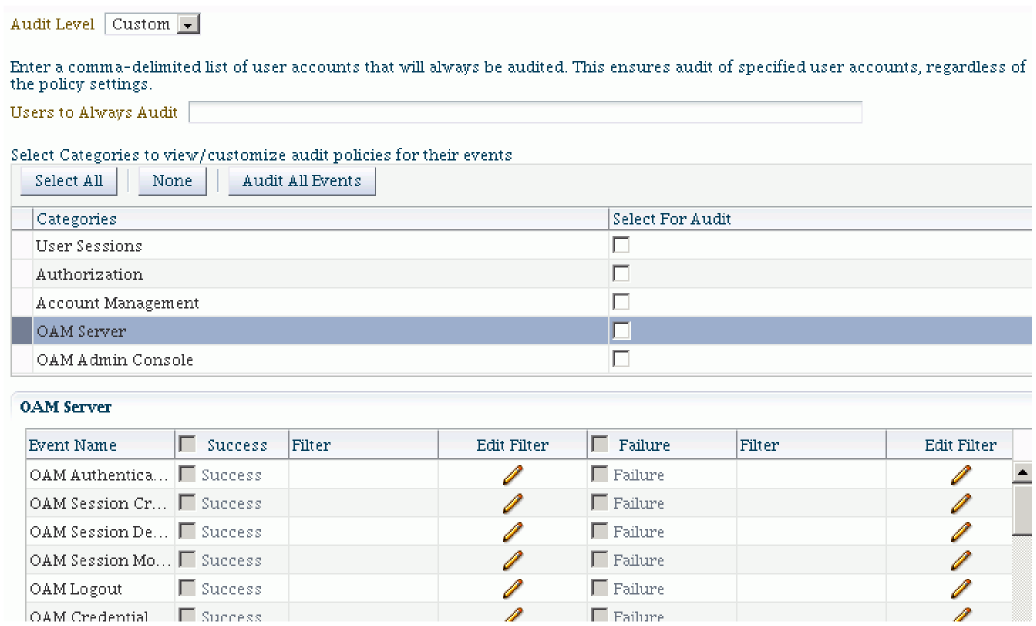
Event Name	Success	Filter	Failure	Filter
OAM Authentica...	✓ Success		✓ Failure	
OAM Session Cr...	✓ Success		✓ Failure	
OAM Session De...	✓ Success		✓ Failure	
OAM Session Mo...	✓ Success		✓ Failure	
OAM Logout	✓ Success		✓ Failure	
OAM Credential ...	✓ Success		✓ Failure	
OAM Credential ...	✓ Success		✓ Failure	
OAM Server Star...	✓ Success		✓ Failure	
OAM Server Shu...	✓ Success		✓ Failure	
OAM Credential ...	✓ Success		✓ Failure	

Note: The table of events can only be edited in custom level. It cannot be edited at the pre-defined levels.

---

6. While the pre-defined levels are sufficient in many cases, you may wish to fine-tune the audit policy for the component.

   To customize the audit policy, use the "Custom" option from the drop-down. This preserves the pre-defined categories but the green check-marks now appear as check-boxes, and all categories are available for selection:



Audit Level Custom

Enter a comma-delimited list of user accounts that will always be audited. This ensures audit of specified user accounts, regardless of the policy settings.

Users to Always Audit

Select Categories to view/customize audit policies for their events

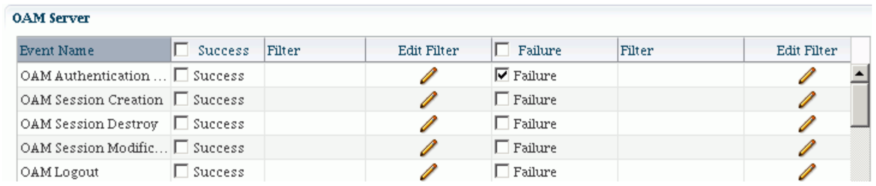
Select All | None | Audit All Events

Categories	Select For Audit
User Sessions	☐
Authorization	☐
Account Management	☐
OAM Server	☐
OAM Admin Console	☐

OAM Server

Event Name	☐ Success	Filter	Edit Filter	☐ Failure	Filter	Edit Filter
OAM Authentica...	☐ Success		✎	☐ Failure		✎
OAM Session Cr...	☐ Success		✎	☐ Failure		✎
OAM Session De...	☐ Success		✎	☐ Failure		✎
OAM Session Mo...	☐ Success		✎	☐ Failure		✎
OAM Logout	☐ Success		✎	☐ Failure		✎
OAM Credential	☐ Success		✎	☐ Failure		✎

You can select desired categories and events to fine-tune the audit policy. For example, you might wish to audit only failed authentication attempts in the Server category:

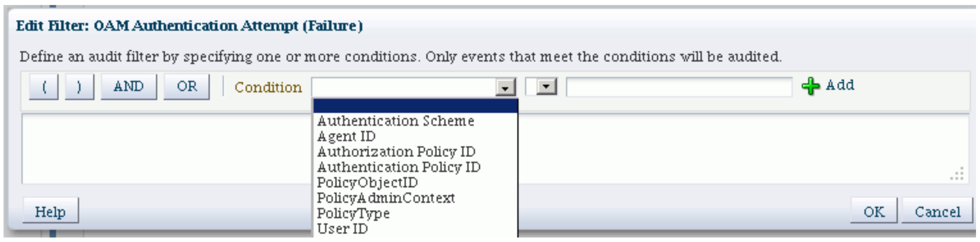


OAM Server

Event Name	☐ Success	Filter	Edit Filter	☐ Failure	Filter	Edit Filter
OAM Authentication ...	☐ Success		✎	☑ Failure		✎
OAM Session Creation	☐ Success		✎	☐ Failure		✎
OAM Session Destroy	☐ Success		✎	☐ Failure		✎
OAM Session Modific...	☐ Success		✎	☐ Failure		✎
OAM Logout	☐ Success		✎	☐ Failure		✎

7. Filters are rule-based expressions that you can define to qualify or filter events for audit. The expressions are based on attributes of the event. For example, you may wish to further fine-tune the policy to narrow down the types of failed authentication attempts to audit.

A pencil icon in the Edit Filter column denotes that a filter is available for the event. Click on the icon to bring up the Edit Filter dialog:

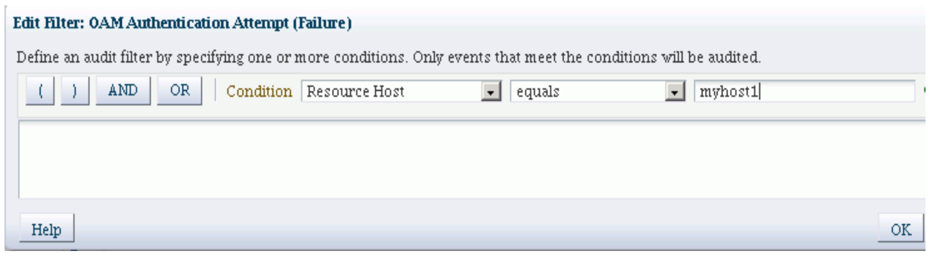


Edit Filter: OAM Authentication Attempt (Failure)

Define an audit filter by specifying one or more conditions. Only events that meet the conditions will be audited.

Specify the filter using the condition boxes, which consist of the filter condition, an operator, and a value. After specifying each condition, click the **add** button. When finished adding conditions, click **OK** to save the changes.

In this example, a failed authentication attempt triggers an audit event only if the server hosting the protected application is named myhost1:



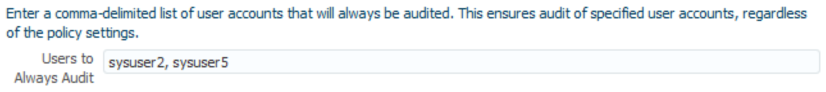
Define an audit filter by specifying one or more conditions. Only events that meet the conditions will be audited.

Condition: Resource Host — equals — myhost1

> **Note:** Each filter attribute has a formal name and a display name. You may see either name in the filter edit dialog. Display names are shown in the drop-down, while names are shown in the edit dialog. For example, if you select 'Client Address IP' in the drop-down box, it is renamed to 'RemoteIP' after you add it to the filter expression.

8. Import/Export - These buttons enable you to save and re-use a policy configuration. At any time while editing the policy, click Export to save the current settings to a file, and Import to load the settings from a saved file.

9. Optionally, under "Users to Always Audit", you can specify a comma-separated list of users to force the audit framework to audit events initiated by these users; auditing occurs regardless of the audit level or filters that have been specified.



Enter a comma-delimited list of user accounts that will always be audited. This ensures audit of specified user accounts, regardless of the policy settings.

Users to Always Audit: sysuser2, sysuser5

> **Notes:**
>
> - Be aware that if you use this feature to audit key users such as system administrators, this will generate audit traffic anytime that user touches any of the auditable events for any component. For example, a component's audit policy may be set to None, but if the user performs some activity in the component instance, it is still audited.
> - User names you enter in this field are not validated.

10. If you made any policy changes, click **Apply** to save the changes. For Java components, you must restart the managed Oracle WebLogic Server (on which the affected Java component is running) for the changes to be effective. Changes are effective after a pre-defined refresh interval, which is 10 minutes by default.

    Click **Revert** to discard any policy changes and revert to the existing policy.

About Component Events

Each component and application in the domain defines its own set of auditable events. Thus, when you expand the Names column of the table, each component displays a list of events that applies to instances of that component.

14.3.2 Managing Audit Policies for System Components with Fusion Middleware Control

This section describes how to view and update audit policies for system components.

Notes:

- Audit policy for Java components is managed in the domain context. See Section 14.3.1, "Managing Audit Policies for Java Components with Fusion Middleware Control".
- See the note at the beginning of Section 14.3, "Managing Audit Policies" titled "Policy Changes Require Server or Instance Restart". Oracle Internet Directory instances do not require a restart.

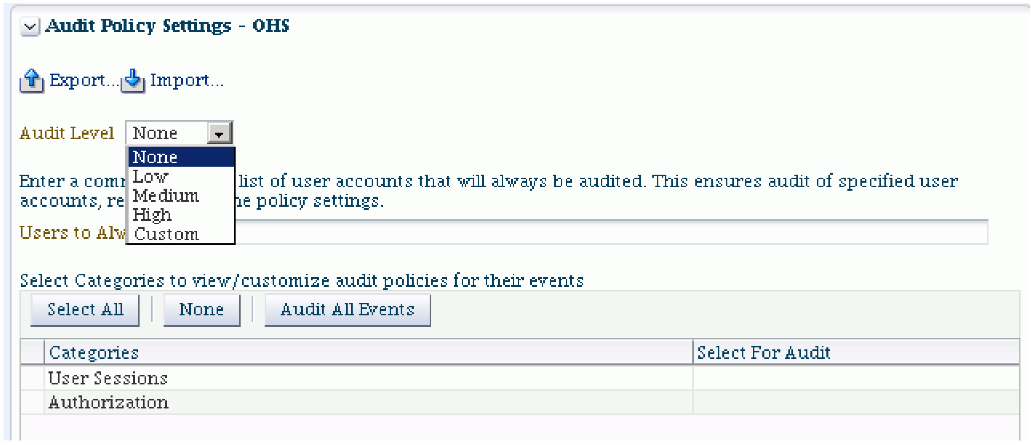
Audit policy for system components is managed in their home pages. The domain Audit Policy Settings page manages audit events for Java components running in the domain.

See Also : Section C.1.1, "What Components Can be Audited?" for the list of auditable components.

The events are organized in a tree structure under the Name column. The tree can be expanded to reveal the details of the events available.

Use these steps to view and update audit policies for system components:

1. Log in to Fusion Middleware Control.
2. Using the topology panel to the left, navigate to the system component of interest such as Oracle HTTP Server.
3. From the component menu, navigate to **Security**, then **Audit Policy**. The Audit Policy Settings page appears.
4. You can select from a drop-down list of pre-configured audit levels. Three pre-defined levels (Low, Medium, High) will automatically pick up a subset of the audit events for the component.


Audit Policy Settings - OHS
Export...
Import...
Audit Level
None
None
Low
Medium
High
Custom
Enter a comr
list of user accounts that will always be audited. This ensures audit of specified user
accounts, re
he policy settings.
Users to Alw
Select Categories to view/customize audit policies for their events
Select All
None
Audit All Events

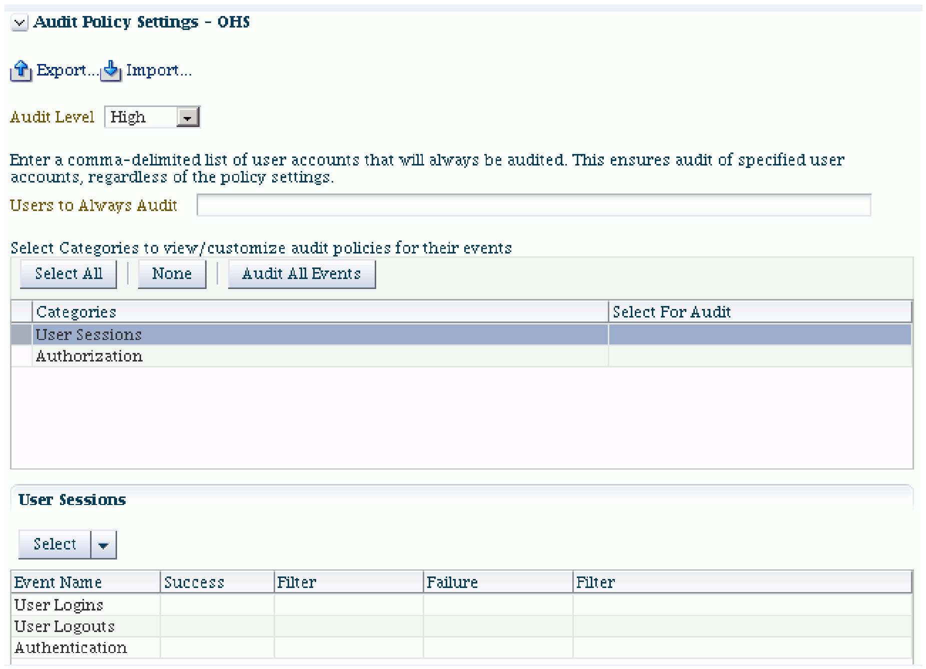
Categories	Select For Audit
User Sessions	
Authorization	

Note: The selection of events under the drop-down box cannot be edited for the pre-defined levels. It can only be edited in custom level.

- None - No events are selected for audit.
- Low - A small set of events is selected, typically those having the smallest impact on component performance.

- Medium or High- This is a larger set of events. These events may have a higher impact on component performance.
- Custom - This level enables you to fine-tune the policy, and is described in Step 5 below.

The table shows the events you can audit for the component instance. This example is for Oracle HTTP Server:



Audit Policy Settings - OHS

Export... Import...

Audit Level: High

Enter a comma-delimited list of user accounts that will always be audited. This ensures audit of specified user accounts, regardless of the policy settings.

Users to Always Audit

Select Categories to view/customize audit policies for their events

Select All | None | Audit All Events

Categories	Select For Audit
User Sessions	
Authorization	

User Sessions

Select

Event Name	Success	Filter	Failure	Filter
User Logins				
User Logouts				
Authentication				

The table consists of these columns:

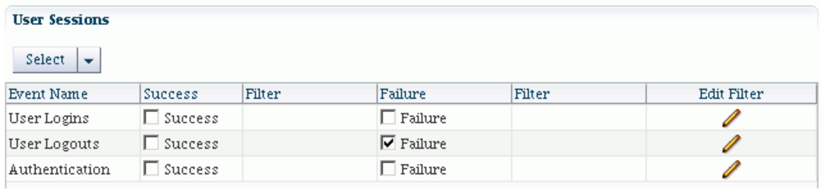
- Name - shows the component events grouped by type, such as Authorization events.
- Enable Audit - shows whether the corresponding event type is being audited. This column is greyed out unless the Custom audit policy is in force.
- Filter - shows any filters in effect for the event type.

5. To customize the audit policy, use the "Custom" option from the drop-down. This allows you to select event categories; to get started, check the "Select For Audit" box for the category you wish to customize.

   A second table lists the events in each category, enabling you to additionally filter for success and failure outcomes of each individual event to further control how they are audited. You can fine-tune filters as explained in Step 6.

6. Filters are rule-based expressions that you can define to qualify or filter events for audit. The expressions are based on attributes of the event. For example, a Login type event could specify an initiator as a user filter in which case the event would generate an audit record whenever the specified user logged in.

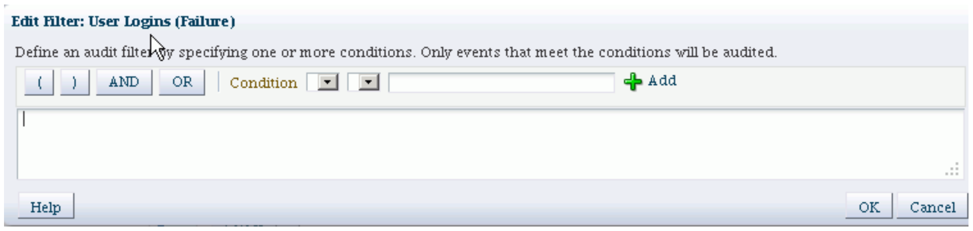
   A pencil icon indicates that a filter is available for the corresponding event.



User Sessions

Event Name	Success	Filter	Failure	Filter	Edit Filter
User Logins	☐ Success		☐ Failure		✎
User Logouts	☐ Success		☑ Failure		✎
Authentication	☐ Success		☐ Failure		✎

Click on a pencil icon to bring up the Edit Filter dialog.



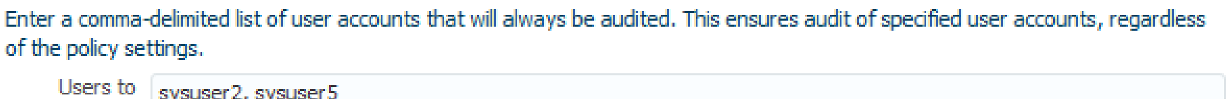
Edit Filter: User Logins (Failure)

Define an audit filter by specifying one or more conditions. Only events that meet the conditions will be audited.

Note: Each filter attribute has a formal name and a display name. You may see either name in the filter edit dialog. Display names are shown in the drop-down, while names are shown in the edit dialog. For example, if you select 'Client Address IP' in the drop-down box, it is renamed to 'RemoteIP' after you add it to the filter expression.

---

7. Click "Select Failures Only" to select only failed events in the policy - for example, a failed authentication. The Enable Audit box is now checked for failed events.
8. Import/Export - These buttons enable you to save and re-use a policy configuration. At any time while editing the policy, click Export to save the current settings to a file, and Import to load the settings from a saved file.
9. Optionally, under “Users to Always Audit”, a comma-separated list of users can be specified to force the audit framework to audit events initiated by these users; auditing occurs regardless of the audit level or filters that have been specified.



Enter a comma-delimited list of user accounts that will always be audited. This ensures audit of specified user accounts, regardless of the policy settings.

Users to svsuser2, svsuser5

Notes:

- Be aware that if you use this feature to audit key users such as system administrators, this will generate audit traffic anytime that user touches any of the auditable events for any component. For example, a component’s audit policy may be set to None, but if the user performs some activity in the component instance, it is still audited.
- No validation is performed for user names you enter in this field.

---

10. If you made any policy changes, click **Apply** to save the changes.

    Click **Revert** to discard any policy changes and revert to the existing policy.

14.3.3 Managing Audit Policies at the Command Line (WLST)

This section explains how to view and update audit policies using the Oracle WebLogic Scripting Tool (WLST) command-line tool:

- Viewing Audit Policies at the Command Line (WLST)
- Updating Audit Policies at the Command Line (WLST)
- Example 1: Configuring an Audit Policy for Users with WLST
- Example 2: Configuring an Audit Policy for Events with WLST
- What Happens to Custom Configuration when the Audit Level Changes?

Note: When running audit `WLST` commands, you must invoke the `WLST` script from the Oracle Common home. See "Using Custom WLST Commands" in Administering Oracle Fusion Middleware for more information.

14.3.3.1 Viewing Audit Policies at the Command Line (WLST)

Take these steps to view audit policies at the command line:

Note: This discussion assumes that you are invoking WLST interactively. For details about WLST and the different options for invoking the tool, see "Getting Started Using the Oracle WebLogic Scripting Tool (WLST)" in *Administering Oracle Fusion Middleware*.

- Connect to the WebLogic Server using the following commands:

  ```
  >java weblogic.WLST
  >connect('userName', 'userPassword', 'url', 'adminServerName')
  ```

- Use the `getAuditPolicy` command to view the audit policy configuration. For example, the following command shows audit policies set at the domain level:

  ```
  wls:/mydomain/serverConfig> getAuditPolicy()
  ```

  The following command shows audit policy for a specific component:

  ```
  wls:/mydomain/serverConfig> getAuditPolicy(componentType="JPS")
  ```

- For system components:
  - obtain the MBean name using the `getNonJava EEAuditMBeanName` command. See Appendix C, "Oracle Fusion Middleware Audit Framework Reference." for details.
  - Use the `getAuditPolicy` command and include the MBean name to view the audit policy configuration. For example:

    ```
    wls:/mydomain/serverConfig> getAuditPolicy
     (on="oracle.security.audit.test:type=CSAuditMBean,name=CSAuditProxyMBean")
    ```

14.3.3.2 Updating Audit Policies at the Command Line (WLST)

Take these steps to update audit policies with the Oracle WebLogic Scripting Tool (WLST) command-line tool:

Note: This discussion assumes that you are invoking WLST interactively. For details about WLST and the different options for invoking the tool, see "Getting Started Using the Oracle WebLogic Scripting Tool (WLST)" in the *Administering Oracle Fusion Middleware*.

- Connect to the WebLogic Server using the following commands:

  ```
  >java weblogic.WLST
  >connect('userName', 'userPassword', 'url', 'adminServerName')
  ```

- Navigate the bean hierarchy to access the domain of interest. For example, if the domain is called `mydomain`:

  ```
  wls:/mydomain/serverConfig>
  ```

- Use the `setAuditPolicy` command to update the audit policy configuration.
- For components that manage their policy locally, use the `setAuditPolicy` command and include an MBean name to update the audit policy configuration. The name for an Audit MBean is of the form:

  ```
  oracle.as.management.mbeans.register:type=component.auditconfig,name=auditconfi
  g1,instance=INSTANCE,component=COMPONENT_NAME
  ```

  For example:

  ```
  oracle.as.management.mbeans.register:type=component.auditconfig,name=auditconfi
  g1,instance=instance1,component=oid1
  ```

- For system components like Oracle HTTP Server or Oracle Internet Directory, explicitly call `save` after issuing a `setAuditPolicy`, or `importAuditConfig`, command.

  If you do not invoke save, the new settings will not take effect.

  In the following example, we navigate to the Root Proxy MBean in the tree and use `invoke` commands to call `setAuditPolicy` and `save`, respectively:

  ```
  ORACLE_COMMON_HOME/common/bin/wlst.sh
  connect('username', 'password', 'protocol://localhost:7001', 'localhost:7001')
  custom()
  cd('oracle.as.management.mbeans.register')
  cd('oracle.as.management.mbeans.register:type=component,name=oid1,instance=inst
  ance1')
  invoke('load',jarray.array([],java.lang.Object),jarray.array([],
  java.lang.String))
  setAuditPolicy(filterPreset='None',
   on='oracle.as.management.mbeans.register:type=component.auditconfig,
   name=auditconfig1,instance=instance1,component=oid1')
  invoke('save',jarray.array([],java.lang.Object),jarray.array([],
  java.lang.String))
  ```

  JavaEE components do not need this step.

14.3.3.3 Example 1: Configuring an Audit Policy for Users with WLST

In this scenario, the domain's current policy audits a user named user1. The administrator would like to add two names, user2 and user3, to the list of users who are always audited, and remove user1 from the list.

The following invocation of `setAuditPolicy` performs this task:

```
setAuditPolicy
   (filterPreset="None",addSpecialUsers="user2,user3",removeSpecialUsers="user1")
```

See Also: Appendix C, "Oracle Fusion Middleware Audit Framework Reference."

14.3.3.4 Example 2: Configuring an Audit Policy for Events with WLST

In this scenario, the domain's current policy audits user logout events. The administrator would like to remove the logout events from the policy and, instead, audit login events.

The following invocation of `setAuditPolicy` for Oracle HTTP Server (OHS) performs this task:

```
setAuditPolicy (on="OHS mbean name")
(filterPreset="Custom",addCustomEvents="OHS:UserLogin",
removeCustomEvents="OHS:UserLogout")
```

Notice that the `Custom` filter preset was invoked to add and remove events.

Note: This example uses the component type OHS for Oracle HTTP Server. Substitute the relevant component type when using the command.

14.3.3.5 What Happens to Custom Configuration when the Audit Level Changes?

When auditing is configured at the custom audit level, and you subsequently use WLST to switch to a different (non-custom) audit level, the custom audit settings are retained unless you explicitly remove those custom settings.

An example illustrates this behavior:

1. Custom audit level is set for a component's policy. An audit filter is specified as part of the configuration.
2. At run-time, audit data is collected according to the specified filter.
3. The component's audit policy is now changed from custom audit level to, say, the low audit level using the WLST `setAuditPolicy` command. However, the filter that was set up as part of the custom audit level persists in the audit configuration.
4. Audit data is now collected based on the low audit level, not the custom level.
5. The component's audit policy is changed back to custom level. An additional filter is added. This filter is appended to the originally configured filter. Unless the original filter is explicitly deleted, it remains part of the configuration.
6. At run-time, audit data is collected based on all prevailing filters at the custom level.

14.3.4 Managing Audit Policies through API Programming

The audit administration service provides a set of APIs that applications can use to retrieve and update audit policies. For details, see Section 23.5.

14.4 Understanding Audit Timestamps

Prior to 11g Release 1 (11.1.1.7), the audit service wrote out database records using the application server's time zone. Starting with 11g Release 1 (11.1.1.7), the audit service can account for different time zones for the application server and the audit data store.

This means:

- New sites see audit events written in Coordinated Universal Time (UTC).
- Sites that upgrade from 11g Release 1 (11.1.1.6), by default, continue to use the application server time zone for audit records unless you explicitly switch to UTC.

Note: Records to bus-stop files always use UTC.

Audit Timestamps at New Sites

At new installations, audit records use UTC timestamps. A service property (`audit.timezone=utc`) in the default audit service configuration implements this convention.

Out-of-the-box, audit service configuration looks like this:

```
<serviceInstance name="audit" provider="audit.provider"
      location="./audit-store.xml">
   <property name="audit.filterPreset" value="None"/>
   <property name="audit.timezone" value="utc" />
   <property name="audit.loader.repositoryType" value="File" />
   <property name="auditstore.type" value="file"/>
</serviceInstance>
```

Audit Timestamps at Upgraded Sites

Audit records that existed prior to 11g Release 1 (11.1.1.7) used the application server timestamp. After upgrading from that release, the audit service configuration remains unchanged and, by default, the records continue to be written using the application server timestamp.

If you wish to use the UTC timestamp after upgrading to 12*c* (12.1.3), you can add the service property `audit.timezone=utc` in the configuration file to get the UTC behavior.

If you switch to UTC after the upgrade, Oracle recommends that you move the existing records first to avoid any potential inconsistency which can affect reporting.

See Also: Section F.2.8

14.5 About Audit Logs and Bus-stop Files

This section contains the following topics:

- Where are Audit Logs Located?
- About Audit Timestamps in Bus-stop Files

14.5.1 Where are Audit Logs Located?

Fusion Middleware Audit Framework provides a set of log files to help with audit administration. You can use these logs to trace errors and for diagnostic purposes when the audit framework is not functioning properly.

For a listing of all audit log locations, how to configure the loggers, and how to use the logs to diagnose issues, see Section J.1.1.7, "About Audit Loggers".

14.5.2 About Audit Timestamps in Bus-stop Files

Time stamps in the audit bus-stop files are recorded in Coordinated Universal Time. This may differ from the machine time depending on the machine's time zone setting.

> **See Also:** Section 14.4

14.6 Advanced Administration of the Database Store

The audit schema is created when the OPSS security schema is created. This section explains the organization of the audit schema and contains the following topics related to maintaining the schema:

- Overview of the Audit Schema
- Table Sizing: Base and Component Table Attributes
- Performance Tuning
- Planning Backup and Recovery
- Importing and Exporting Data
- Partitioning

> **See Also:** For more information on RCU, see *Oracle Fusion Middleware Repository Creation Utility User's Guide*.

14.6.1 Overview of the Audit Schema

The common tables in the Oracle Fusion Middleware Audit Framework schema consist of the following:

- basic data in a base table
- common attributes in the `IAU_COMMON` table
- generic attributes in dedicated tables
- custom attributes in the `IAU_CUSTOM_nnn` tables.

Not all tables are used at the same time. Typical sites using the dynamic metadata model (explained in Section 13.3.1.1) utilize the common and custom tables. The older model uses the base table and component-specific tables.

For details about these tables, see Section 13.4.

About the Bus-stop File

By default, the bus-stop file is maintained in the directory:

```
Weblogic Domain Home/servers/server_name/logs/auditlogs
```

with sub-directories for the components. For example, Oracle Platform Security Services (OPSS) bus-stop files reside in:

```
Weblogic Domain Home/servers/server_name/logs/auditlogs/JPS
```

Here is a sample bus-stop file for OPSS:

```
#Fields:Date Time Initiator EventType EventStatus MessageText HomeInstance ECID
```

```
RID ContextFields SessionId TargetComponentType ApplicationName EventCategory
ThreadId InitiatorDN TargetDN FailureCode RemoteIP Target Resource Roles
CodeSource InitiatorGUID Principals PermissionAction PermissionClass mapName key
#Remark Values:ComponentType="JPS"
2008-12-08 10:46:05.492  - "CheckAuthorization" true "Oracle Platform Security
Authorization Check Permission SUCCEEDED." - - - - - - - "Authorization" "48" - -
"true" - - "(oracle.security.jps.service.policystore.PolicyStoreAccessPermission
context=APPLICATION,name=SimpleServlet getApplicationPolicy)" -
"file:/oracle/work/middleware/oracle_common/modules/oracle.jps_
11.1.1/jps-internal.jar" - "[]" - - - -
```

Common Table and Custom Tables

The common table contains the basic data for an audit record and is related to the custom tables, which contain additional data, through the IAU_ID attribute.

Base Table and Component Tables

Likewise, the base table containing the basic audit record is related to the component-specific tables, which contain additional data, through the IAU_ID attribute.

14.6.2 Table Sizing: Base and Component Table Attributes

When generated, audit records are stored in a file; if an audit database store is configured, the audit loader stores each audit record as follows:

- General information (such as Time, EventType, and EventStatus) is written into one row of the common table (dynamic model) or the base table.
- Component-specific information (such as `CodeSource`) is written into one row of the custom table (dynamic model) or the component table.

The average record size for basic audit records is approximately 0.3 KB. When you plan for tablespace sizing:

- use this number as a guideline for the average record size
- monitor how audit database size is growing based on the audit policy selected and the level of activity
- take into account the period of time for which the audit data is being stored.

The attributes of the base table and the component-specific tables respectively are derived from these files:

```
$ORACLE_HOME/modules/oracle.iau_12.1.3/components/generic/generic_events.xml
```

for the base table, and

```
$ORACLE_HOME/modules/oracle.iau_12.1.3/components/componentName/component_
events.xml
```

for each component table.

Table C–6 lists the attributes defined in the base table `IAU_BASE`. Table C–7 likewise lists the attributes defined in the common table IAU_COMMON.

You can use the `listAuditEvents` WLST command to get a list of all attribute names for individual component tables.

See Also:

- Appendix C, "Oracle Fusion Middleware Audit Framework Reference.".
- Section C.2, "The Audit Schema"

14.6.3 Performance Tuning

For efficient queries, an index is created by default on the Timestamp (IAU_TSTZORIGINATING) in the base table and on each of the component-specific tables.

The default index in `IAU_BASE` is named `EVENT_TIME_INDEX`, and in the component tables it is named *`tableName`*`_INDEX` (such as `OVDCOMPONENT_INDEX`, `OIDCOMPONENT_INDEX`, `JPS_INDEX` and so on).

Additional indexed columns and their indexes are as follows:

- Timestamp (IAU_TSTZORIGINATING) in the common table, with an index called DYN_EVENT_TIME_INDEX.
- IAU_AuditUser, IAU_ComponentType, IAU_EventCategory, IAU_EventType columns, all on the common table. Indexes are called DYN_USER_INDEX, DYN_COMPONENT_TYPE_INDEX, DYN_EVENT_CATEGORY_INDEX and DYN_EVENT_TYPE_INDEX respectively.

14.6.4 Planning Backup and Recovery

Compliance regulations require that audit data be stored for long periods. A backup and recovery plan is needed to protect the data.

A good backup plan takes account of these basic guidelines:

- Growth rate of Audit Events

  The number of audit events generated depends on your audit policy. The number of audit events generated daily determines, in turn, how often you want to perform backups to minimize the loss of your audit data.

- Compliance regulations

  Consult you organization's compliance regulations to determine the frequency of backups and number of years for which audit data storage is mandatory.

- Online or Offline Data Management

  Consult you organization's compliance regulations to determine the frequency of backups and the portion of audit data that needs to be easily accessible.

Oracle Database uses Oracle Recovery Manager (RMAN) for backup and recovery. For details, see:

`http://www.oracle.com/technology/deploy/availability/htdocs/BR_Overview.htm`

`http://www.oracle.com/technology/deploy/availability/htdocs/rman_overview.htm`

Note: The translation table, `IAU_DISP_NAMES_TL`, needs to be backed up only once, since it should not change over time.

14.6.5 Importing and Exporting Data

You can import and export the audit schema to migrate data if you started with multiple audit databases and wish to combine them into a single audit data store, or if you wish to change the database to scale up.

Oracle Database sites can utilize the utilities of Oracle Data Pump to import and export data. For details, refer to:

http://www.oracle.com/technology/products/database/utilities/htdocs/data_pump_overview.html

14.6.6 Partitioning

Not all database systems support partitioning, all the tables in the audit schema are unpartitioned by default.

Since audit data is cumulative and older data is never removed, if you store a high volume of audit data you should consider partitioning the audit schema, as it will allow for easier archiving.

Benefits of partitioning include:

- Improved Performance: If a table is range-partitioned by Timestamps, for example, queries by Timestamps can be processed on the partitions within that time-frame only.
- Better Manageability: Partitions can be created on separate tablespaces (thus different disks). This enables you to move older data to slower and larger disks, while keeping newer data in faster and smaller disks.

  In addition, partitioning makes archival much easier. For example, you can compress a singlve partition rather than having to partition the entire table.
- Increased Availability: If a single partition is unavailable, for example, and you know that your query can eliminate this partition from consideration, the query can be successfully processed without needing to wait for the unavailable partition.

14.6.6.1 Partitioning Tables

Note: This discussion applies to releases prior to 11g Release 1 (11.1.1.7). New 11g Release 1 (11.1.1.7) installations do not use these tables.

In this example, `IAU_BASE` is used as an example to demonstrate how to convert the unpartitioned tables in the audit schema into partitioned tables.

It is recommended that partitioning is done before using this schema for an audit data store to minimize the application down time.

Note: Two sample SQL scripts are shipped with the product:

- `$MW_HOME/oracle_common/common/sql/iau/scripts/convertPartitionedTables.sql` (linux) or `%MW_HOME\oracle_common\common\sql\iau/scripts\convertPartitionedTables.sql` (Windows) converts the base and component tables in the audit schema into partitioned tables.
- `$MW_HOME/oracle_common/common/sql/iau/scripts/createPartitionsByQuarter.sql` (linux) or `%MW_HOME\oracle_common\common\sql\iau/scripts\createPartitionsByQuarter.sql` (Windows) creates partitions by quarter for the base and component tables in the audit schema.

The partitioning steps are as follows:

1. Rename the existing unpartitioned table. For example:

```
RENAME IAU_BASE TO IAU_BASE_NONPART;
```

2. Create a new partitioned table that follows the table structure of the unpartitioned table. This example uses the range-partitioning (by Timestamp) scheme:

```
CREATE TABLE IAU_BASE
PARTITION BY RANGE (IAU_TSTZORIGINATING)
(
    PARTITION IAU_BASE_DEFAULT VALUES LESS THAN (MAXVALUE)
)
AS SELECT * FROM IAU_BASE_NONPART;
```

3. Enable row movement to allow data to automatically move from partition to partition when new partitions are created. For example:

```
ALTER TABLE IAU_BASE
ENABLE ROW MOVEMENT;
```

4. Create a local prefix index for the partitioned table. For example:

```
ALTER INDEX EVENT_TIME_INDEX
RENAME TO EVENT_TIME_INDEX_NONPART;

CREATE INDEX EVENT_TIME_INDEX
ON IAU_BASE(IAU_TSTZORIGINATING) LOCAL;
```

5. Partitions can now be created. In this example partitions are created by calendar quarter:

```
ALTER TABLE IAU_BASE
SPLIT PARTITION IAU_BASE_DEFAULT AT (TO_DATE('01/04/2008', 'DD/MM/YYYY')) INTO
(PARTITION IAU_BASE_Q1_2008, PARTITION IAU_BASE_DEFAULT)
UPDATE INDEXES;

ALTER TABLE IAU_BASE
SPLIT PARTITION IAU_BASE_DEFAULT AT (TO_DATE('01/07/2008', 'DD/MM/YYYY')) INTO
(PARTITION IAU_BASE_Q2_2008, PARTITION IAU_BASE_DEFAULT)
UPDATE INDEXES;

ALTER TABLE IAU_BASE
```

```
SPLIT PARTITION IAU_BASE_DEFAULT AT (TO_DATE('01/10/2008', 'DD/MM/YYYY')) INTO
(PARTITION IAU_BASE_Q3_2008, PARTITION IAU_BASE_DEFAULT)
UPDATE INDEXES;

ALTER TABLE IAU_BASE
SPLIT PARTITION IAU_BASE_DEFAULT AT (TO_DATE('01/01/2009', 'DD/MM/YYYY')) INTO
(PARTITION IAU_BASE_Q4_2008, PARTITION IAU_BASE_DEFAULT)
UPDATE INDEXES;
```

Note: New partitions should be created periodically for new quarters.

14.6.6.2 Backing Up and Recovering Partitioned Tables

Backup and recovery were discussed in Section 14.6.4, "Planning Backup and Recovery". Note that read-only tablespaces can be excluded from whole database backup, so long as a backup copy was created. Thus, you can improve performance by avoiding unnecessary backups for the partitions of archived data residing on those tablespaces.

14.6.6.3 Importing and Exporting Data

Import and export were discussed in Section 14.6.5, "Importing and Exporting Data". Keep in mind that with a range-partitioned table it is much more efficient to drop a partition when you want to remove old data, rather than deleting the rows individually.

```
ALTER TABLE IAU_BASE DROP PARTITION IAU_BASE_Q4_2008;
```

It is also easy to load a partition of new data without having to modify the entire table. However, you have to remove the default partition of "values less than (`MAXVALUE`)" first, and add it back once finished, using a command like the following:

```
ALTER TABLE IAU_BASE ADD PARTITION IAU_BASE_Q4_2008 VALUES LESS THAN
('01-JAN-2009');
```

Once partitions are created, you can purge or back up a particular partition. Refer to Section 14.6.6.4 for details.

In the database mode, the audit loader automatically manages bus-stop files.

14.6.6.4 Purging Data

Oracle Fusion Middleware Audit Framework provides SQL scripts to enable you to purge records from the audit store.

Location of Scripts

The purge scripts are located at:

```
[Linux]
$MW_HOME/oracle_common/common/sql/iau/scripts


[Windows]
%MW_HOME\oracle_common\common\sql\iau\scripts
```

The following scripts reside at this location:

- `auditDataPurge.sql`
- `auditDeleteData.sql`
- `auditReclaimSpace.sql`

Deleting Audit Records

To delete records without taking any other action, use auditDeleteData.sql. The script takes two parameters:

- Audit Schema user
- Number of Days to keep - older records are deleted.

```
auditDeleteData.sql audit_schema_user number_of_days_to_keep
```

For example:

```
sqlplus> @auditDeleteData.sql DEV_IAU 100
```

deletes all records older than 100 days.

Reclaiming Space in Data Store

To reclaim space, use the `auditReclaimSpace.sql` script. The script takes only one parameter, the audit schema user.

```
@auditReclaimSpace.sql audit_schema_user
```

For example:

```
sqlplus> @auditReclaimSpace.sql DEV_IAU
```

Deleting Records and Reclaiming Space

To both delete audit records and reclaim space, use the `auditDataPurge.sql` script. This script takes two parameters:

- Audit Schema user
- Number of Days to keep; older records are deleted.

```
@auditDataPurge.sql audit_schema_user number_of_days_to_keep
```

For example:

```
sqlplus> @auditDataPurge.sql DEV_IAU 100
```

deletes all records older than 100 days, and enables row movements to shrink space.

14.6.6.5 Performing Tiered Archival

Partitioning enables individual partitions (or groups of partitions) to be stored on different storage tiers. You can create tablespaces in high-performance or low-cost disks, and create partitions in different tablespaces based on the value of the data or other criteria. It is also easy to move data in partitions between the tablespaces (storage tiers).

Here is an example:

```
ALTER TABLE IAU_BASE MOVE PARTITION IAU_BASE_Q1_2008
TABLESPACE AUDITARCHIVE UPDATE INDEXES;
```

Note : Partitions can be moved only in Range, List, System, and Hash partitioning schemes.

Oracle Information Lifecycle Management (ILM) features streamlined data management through partitioning and compression. For details, refer to:

`http://www.oracle.com/technetwork/database/enterprise-edition/index-090321.html`

14.7 Best Practices for Managing Audit Event Definitions

This section provides tips and techniques for managing your site's audit configuration and maximizing the value of the collected audit data. It contains these topics:

14.7.1 General Naming Guidelines

Ensure that all audit objects including component type, category, event, and attribute conform to these naming conventions:

- Do not use spaces or special characters in names used for componentType, Category, Event, or Attribute. Only English alphabet characters are allowed.
- Use spaces only in display names.

14.7.2 Naming of Events

Follow these guidelines when naming audit events:

- Do not prefix everything with the component name.
- Try to make names as specific as possible. For example, use "HTTPResponse" instead of "Response" when defining the HTTP response code 400, 302 and so on.
- Do not apply the "Event" suffix to all event names. For example, instead of "AuthenticationEvent", "PolicyEvent" and so on, simply use "Authentication", "Policy" and so on.

14.7.3 Optimizing Event Granularity

Follow these guidelines to optimize event granularity:

- Try to define separate events for each operation. For example, instead of combining them into one "Policy" event and having an attribute "Operation" to distinguish between them, make "PolicyCreate" and "PolicyDelete" as separate events.
- Do not define separate events for Success and Failure. For example, do not have "PolicyCreateSuccess" and "PolicyCreateFailure" as separate events - instead use the EventStatus attribute to record Success or Failure.

14.7.4 Categorizing Events

Follow these guidelines to categorize events:

- Try to reference the System categories when applicable. For example, "UserSession" and "Authorization".

- For configuration operations, make a category around a group of operations. For example, put PolicyCreate, PolicyDelete in a component specific category called "Policy"

14.7.5 Using Generic Attributes

Follow these guidelines to utilize generic attributes:

- Try to include these key event attributes:
  - Initiator - the user who did this action.
  - MessageText - a short description of what happened. Instead of using static text like "Authentication event", put in actual data values to make the description human readable; for example, "jdoe logged in sucessfully" .
  - EventStatus - outcome of the event - true (success) or false (failure)
  - FailureCode - If failure, then the error code applicable to the failure.
- Another commonly used attribute is Resource - the "object" that was affected . For example, a URI that was accessed, or a Permission that was granted. etc. Sometimes you may choose to not use Resource but your own custom attributes.

14.7.6 Using Component Attributes

Ultimately audit framework does a union of all the attributes that you define for each event and defines a database column for each attribute. So try to define as many common attributes as possible.

14.7.7 Cross-Linking

Try to put in enough information so that events generated by your component can be cross linked to other events - for example, by ECID, session id, user name, and so on.

15 Using Audit Analysis and Reporting

This chapter describes how to configure audit reporting and how to view audit reports.

This chapter contains these topics:

> **Note:** To determine whether you can use the reporting features described in this chapter, read Section 15.1 before proceeding.

15.1 About Audit Reporting

Two approaches to audit reporting are possible in Oracle Fusion Middleware Audit Framework. The approach you adopt is dictated by the audit model and component(s) in use at your site:

- Dynamic Metadata Model

  This audit metadata model was introduced in 11*g* Release 1 (11.1.1.6.0). Oracle Fusion Middleware installations starting with 12*c* (12.1.2) automatically use this model.

  Section 15.2 explains how to create audit reports based on this model.

- Report Template Model

  This earlier model is used by system components. In addition, when you upgrade, the audit framework continues to use the model.

15.2 Audit Reporting with the Dynamic Metadata Model

The ultimate goal of audit integration is to create reports of audit events stored in audit database tables. As described in Section 14.6.1, audit events are saved into the common attribute table `iau_common`, and the custom attribute tables *`iau_custom_nnn`*. The OPSS Common Audit Framework generates SQL scripts to create Oracle database views. Component reporting applications can use these views to query audit event data from audit database tables.

The following steps (some of which you have already accomplished) are required:

- Integrate with the audit framework using dynamic or declarative registration (Section 23.2 through Section 23.4).

- Configure audit policies so that the run-time audit service logs events to generate audit data (Section 23.5 and Section 23.6).
- Configure the audit loader to ensure bus-stop files are migrated to the database (Section 14.2).
- Use the `createAuditDBView` command to generate a SQL script of audit definitions ("Viewing Audit Definitions for your Component" in Section 13.5.3.2).
- Log in to the database as the IAU schema user to create a view using the SQL script from the previous step.
- Configure your reporting application to query the view.

Here is a sample output of `createAuditDBView` for component `ApplicationAudit`:

```
-- Audit View for Component

CREATE VIEW ApplicationAudit_AUDITVIEW AS

SELECT IAU_AUDITSERVICE.IAU_TRANSACTIONID AS AUDITSERVICE_TRANSACTIONID,

IAU_COMMON.IAU_COMPONENTTYPE AS ComponentType,

IAU_COMMON.IAU_MAJORVERSION AS MajorVersion,

IAU_COMMON.IAU_MINORVERSION AS MinorVersion,

IAU_COMMON.IAU_INSTANCEID AS InstanceId,

IAU_COMMON.IAU_HOSTID AS HostId,

IAU_COMMON.IAU_HOSTNWADDR AS HostNwaddr,

IAU_COMMON.IAU_MODULEID AS ModuleId,

IAU_COMMON.IAU_PROCESSID AS ProcessId,

IAU_COMMON.IAU_ORACLEHOME AS OracleHome,

IAU_COMMON.IAU_HOMEINSTANCE AS HomeInstance,

IAU_COMMON.IAU_ECID AS ECID,

IAU_COMMON.IAU_RID AS RID,

IAU_COMMON.IAU_CONTEXTFIELDS AS ContextFields,

IAU_COMMON.IAU_SESSIONID AS SessionId,

IAU_COMMON.IAU_TARGETCOMPONENTTYPE AS TargetComponentType,

IAU_COMMON.IAU_APPLICATIONNAME AS ApplicationName,

IAU_COMMON.IAU_EVENTTYPE AS EventType,

IAU_COMMON.IAU_EVENTCATEGORY AS EventCategory,

IAU_COMMON.IAU_EVENTSTATUS AS EventStatus,

IAU_COMMON.IAU_TSTZORIGINATING AS TstzOriginating,
IAU_COMMON.IAU_THREADID AS ThreadId,

IAU_COMMON.IAU_COMPONENTNAME AS ComponentName,

IAU_COMMON.IAU_INITIATOR AS Initiator,

IAU_COMMON.IAU_MESSAGETEXT AS MessageText,

IAU_COMMON.IAU_FAILURECODE AS FailureCode,

IAU_COMMON.IAU_REMOTEIP AS RemoteIP,

IAU_COMMON.IAU_TARGET AS Target,

IAU_COMMON.IAU_RESOURCE AS IAU_RESOURCE,

IAU_COMMON.IAU_ROLES AS Roles,

IAU_COMMON.IAU_DOMAINNAME AS DomainName,

IAU_COMMON.IAU_COMPONENTDATA AS ComponentData,

IAU_COMMON.IAU_AUDITUSER AS AuditUser,

IAU_COMMON.IAU_TENANTID AS TenantId,

IAU_COMMON.IAU_TRANSACTIONID AS TransactionId,

IAU_COMMON.IAU_USERTENANTID AS UserTenantId,

IAU_CUSTOM.IAU_INT_001 AS AccountNumber,

IAU_CUSTOM.IAU_DATETIME_001 AS Date,

IAU_CUSTOM.IAU_FLOAT_001 AS Amount,

IAU_CUSTOM.IAU_STRING_002 AS Status,

IAU_CUSTOM.IAU_FLOAT_002 AS Balance,

IAU_USERSESSION.IAU_AUTHENTICATIONMETHOD AS AuthenticationMethod

FROM IAU_AUDITSERVICE, IAU_COMMON, IAU_CUSTOM, IAU_USERSESSION WHERE IAU_
COMMON.IAU_ID = IAU_AUDITSERVICE.IAU_ID AND IAU_COMMON.IAU_ID = IAU_CUSTOM.IAU_ID
AND IAU_COMMON.IAU_ID = IAU_USERSESSION.IAU_ID AND IAU_COMMON.IAU_ComponentType =
'ApplicationAudit';
```

15.3 Audit Reporting with the Report Template Model

You can leverage Oracle Business Intelligence Publisher to generate reports from your application's audit data, utilizing the same reporting capabilities available to Oracle components.

See Oracle Business Intelligence Publisher documentation for details.

Part IV

Developing with Oracle Platform Security Services APIs

This part explains how to develop custom security solutions in your applications using OPSS APIs, and it contains the following chapters:

16 Integrating Application Security with OPSS

This chapter describes a number of security-related use cases, including the propagation of identities in domains and across domains, and the typical lifecycle of an ADF application security. It also lists code and configuration samples presented elsewhere in this Guide.

This chapter contains the following sections:

16.1 Introduction

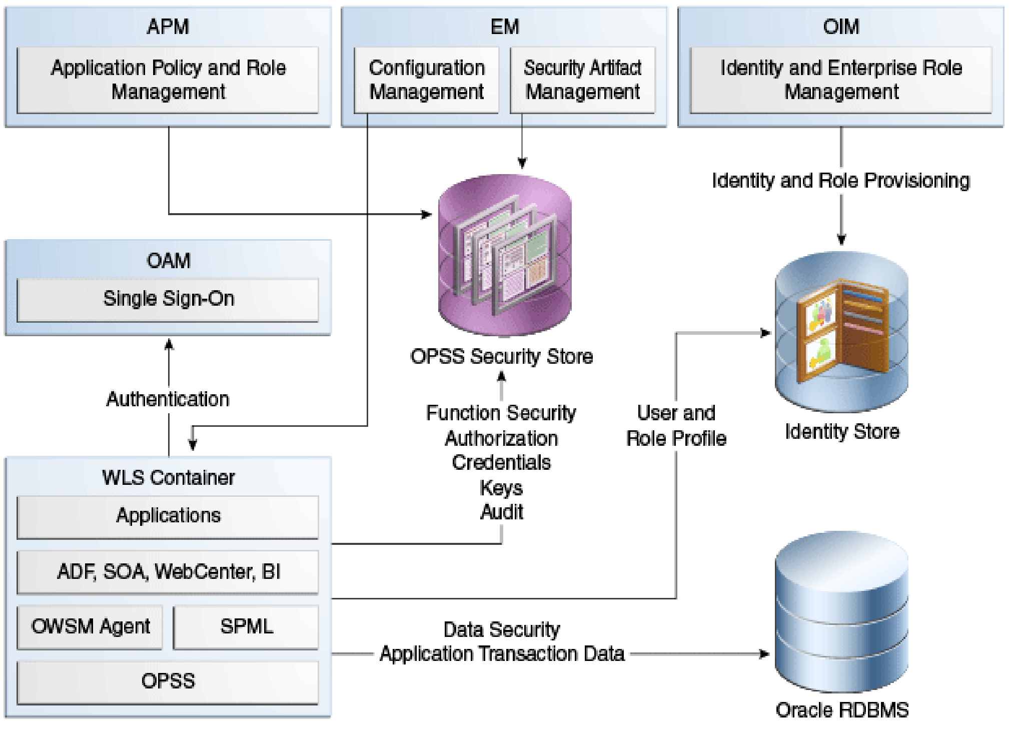
The audience for the material presented in this chapter are developers, security architects, and security administrators. The presentation is not feature-driven, as in most topics in this Guide, but use case-driven: a number of use cases that solve typical application security challenges are introduced as a departing point to solve particular application security requirements. Some of the use cases describe a declarative approach (and do not require changes in application code); others provide a programmatic approach; and others require both approaches.

The top security issues that security architects and developers face include managing users, user passwords, and access to resources. OPSS is a suite of security services that provides solutions to these challenges by supporting:

- Externalizing security artifacts and the security logic from the application
- A declarative approach to security
- A complete user identity lifecycle
- Policy-driven access controls

Figure 16–1 illustrates how applications access the security stores and the tools to manage those stores.

Figure 16–1 Applications, Security Stores, and Management Tools



Links to Related Documentation

Topics explained elsewhere include the following:

- The OPSS Security Architecture - see Section 1.2, "OPSS Architecture Overview."
- Single Sign On - see *Oracle Fusion Middleware Administrator's Guide for Oracle Access Manager with Oracle Security Token Service*.
- ADF applications - see *Developing Fusion Web Applications with Oracle Application Development Framework*.
- Oracle Development Tools - see *Oracle Fusion Middleware Reference for Oracle Security Developer Tools.*

For the list of OPSS APIs, see Appendix G, "OPSS API References."

16.2 Security Integration Use Cases

This section introduces a number of use cases categorized according to a main security feature or security artifact, in the following sections:

- Authentication
- Identities
- Authorization
- Credentials
- Audit
- Identity Propagation
- Administration and Management

- Integration

Each use case contains a brief description of the problem it attempts to solve, the security artifacts required, the features involved, and links to details solving the stated problem.

16.2.1 Authentication

The authentication use cases are the following:

- Java EE Application Requiring Authenticated Users - Users must be authenticated in order to access a Java EE application.
- Java EE Application Requiring Programmatic Authentication - Java EE application requires authenticating a user programmatically.
- Java SE Application Requiring Authentication - Java SE application requires authenticating against a domain identity store.

16.2.1.1 Java EE Application Requiring Authenticated Users

In order to access a Java EE application, users must be authenticated against the identity store in cases where the identity store is any of the following:

- Single LDAP-based store
- Several LDAP-based stores of the same kind (such as all OID, for example)
- Several LDAP-based stores of different kinds; in particular two LDAP-based stores: one AD LDAP and a second one OID LDAP
- Single DB-based store
- Several LDAP- and DB-based stores

This use case requires:

- Allowing access to the application to only authenticated users
- Not modifying the application code, even when customers have user identities in different repositories

This use case features:

- Deploying an application to a WebLogic container
- Configuring the appropriate authenticators according to the particular set of user repositories
- Configuring the OVD authenticator in case of a mixed LDAP types or mixed LDAP and DB types

According to the repository used, the details of this use case are split into the following scenarios:

- Single user repository - Configure the appropriate authenticator with the WebLogic console
- Multiple user repositories (or split profiles across LDAP of the same of different kinds) - Configure the OVD authenticator
- DB-based repositories - Configure the OVD authenticator

For details, see Section 3.2.2, "WebLogic Authenticators."

16.2.1.2 Java EE Application Requiring Programmatic Authentication

A Java EE application, not using deployment descriptors, must authenticate the user programmatically against the configured identity store(s); it applies only to Java EE applications deployed to the Oracle WebLogic Application Server.

This use case requires using the OPSS public API to authenticate a user, and it features:

- Configuring authenticators for a Java EE container
- Using the LoginService API to authenticate the user

For details about this use case, see Section 24.1, "Authenticating Java EE Applications."

16.2.1.3 Java SE Application Requiring Authentication

A Java SE application must authenticate users against the LDAP identity store in use in a domain; the application code requesting authentication must be same regardless of the specifics of the domain's identity store.

This use case requires configuring the identity store(s) against which the authentication should take place and using the LoginService; note that a Java SE application can use only one id login module.

For details about this use case, see Section 25.3.4, "Using the OPSS API LoginService in Java SE Applications."

16.2.2 Identities

The identity use cases are the following:

- Application Running in Two Environments - Application, running in two different environments, needs to access user profile information in an LDAP-based store.
- Application Accessing User Profiles in Multiple Stores - Application needs to access user profile information stored in multiple LDAP-based stores.

16.2.2.1 Application Running in Two Environments

An application, which runs in two different environments, needs to access user profile information, such as a user's email address, stored in an LDAP-based store; the type of the LDAP server can be any of the supported types, and that type may differ with the environment. For details on supported types, see Section 4.1, "Supported File-, LDAP-, and DB-Based Services."

More specifically, this use case assumes that:

- The application uses the method `UserProfile.getEmail()`.
- In one environment, there is an AD LDAP configured as follows:

  ```
  mail.attr = msad_email
  ```

- In the second environment, there is an OID LDAP configured as follows:

  ```
  mail.attr = mail
  ```

In order for the application to retrieve the correct information without modifying the code and regardless of the environment (first or second) in which it runs, the identity store provider must be configured with the correct property in each of those two environments.

In the first environment (AD LDAP), the identity store provider is set to have the following property:

```
<property name="mail.attr" value="msad_mail">
```

In the second one (OID LDAP), the identity store provider is set to have the following property:

```
<property name="mail.attr" value="mail"
```

For details about this use case, see Section 8.2, "Configuring the Identity Store Provider."

16.2.2.2 Application Accessing User Profiles in Multiple Stores

An application needs access to user profile information located in more than one LDAP-based stores.

This use case requires configuring the environment for multiple LDAP-based stores.

For details about:

- Configuring multiple LDAPs, see Section 8.3.2.8, "Viewing the Configuration File"
- Configuring the identity store service, see Section 8.3, "Configuring the Identity Store Service"

16.2.3 Authorization

The authorization use cases are the following:

- Java EE Application Accessible by Specific Roles - Java EE application accessible only by users configured in web descriptors.
- ADF Application Requiring Fine-Grained Authorization - ADF application requires fine-grained authorization.
- Web Application Securing Web Services - Web services application requires securing web services.
- Java EE Application Requiring Codebase Permissions - Java EE application requires codebase permissions.
- Non-ADF Application Requiring Fine-Grained Authorization - Non-ADF application requires fine-grained authorization.

16.2.3.1 Java EE Application Accessible by Specific Roles

A Java EE application needs to be accessible only by users that had been assigned specific roles in web descriptors; the group-to-role assignment must be configurable at deployment based on the customer's environment.

For details about this use case, see sections Using Declarative Security with Web Applications and Using Declarative Security with EJBs in *Developing Applications with the WebLogic Security Service*.

16.2.3.2 ADF Application Requiring Fine-Grained Authorization

An ADF application in container requires fine-grained authorization at the level of individual controls on the pages in the web application; while the application initiates the authorization check, the policies need to be externalized and customized after the application has been deployed.

For details on how to develop and secure Oracle ADF applications, see chapter 30, Enabling ADF Security in a Fusion Web Application, in *Developing Fusion Web Applications with Oracle Application Development Framework*.

For general information about ADF applications, see Section 1.5.2, "Scenario 2: Securing an Oracle ADF Application."

For details about the lifecycle of an ADF application, see Securing an ADF Application.

16.2.3.3 Web Application Securing Web Services

A web application requires securing web services with fine grained policies.

For details about web services security administration, see *Securing Web Services and Managing Policies with OWSM*.

16.2.3.4 Java EE Application Requiring Codebase Permissions

A Java EE application requires codebase permissions to perform specific actions; typical examples are reading a credential from the credential store or looking up policies in the OPSS security store.

For details about creating codebase policies with Fusion Middleware Control, see Section 10.3.3, "Managing System Policies."

16.2.3.5 Non-ADF Application Requiring Fine-Grained Authorization

A non-ADF application needs to be secured with fine-grained authorization checks.

This use case requires:

- Placing checks in the application code at the appropriate places
- Configuring the appropriate policies

For details see Section 17.3, "The JAAS/OPSS Authorization Model."

16.2.4 Credentials

The credential use case is the following:

- Application Requiring Credentials to Access System - Application requires credentials to access a back-end system.

16.2.4.1 Application Requiring Credentials to Access System

An application requires a credential to connect to a back-end system, such as a database or an LDAP server. The application code should reference this credential in such a way that the specifics of the credential can be changed per customer post deployment without modifying the application code. Furthermore, this use case also requires specifying who can access the credential store and what operations an authorized user can perform on credential data.

This use case features:

- Using the credential store to persist credentials
- Fetching credentials at runtime with the CSF API in application code
- Defining and enforcing system policies on codebase

For details about:

- Configuration and code examples, see Section 19.3, "Setting the Java Security Policy Permissions," and Section 19.7, "Examples"
- Credential management, see Chapter 11, "Managing the Credential Store."

- Packaging, see Section 24.5.2, "Packaging Credentials with Application."

16.2.5 Audit

The audit use cases are the following:

- Auditing Security-Related Activity - An application requires recording security-related activity.
- Auditing Business-Related Activity - An application requires recording business activity in the context of a flow.

16.2.5.1 Auditing Security-Related Activity

An application needs to record security-related activity in several security areas; specifically, the application requires logging the following information:

- Changes to a policy: what and when
- The policies that were evaluated in a particular time interval
- Changes to credentials or keys: what and when

The settings explained in this use case apply to all applications and components in a domain.

This use case requires that auditable applications:

- Integrate with the Common Audit Framework (CAF)
- Have built-in capabilities to log security activities
- Set the proper audit filter level to capture activities in specific security areas

This use case features:

- Integrating with the Common Audit Framework
- Allowing applications to define their own audit categories and events in security areas, and making the application audit-aware
- Allowing applications to set the appropriate filter level

For details about:

- Integrating with CAF, see Section 23.2, "Integrating the Application with the Audit Framework."
- Registering applications, see Section 23.4, "Registering the Application with the Audit Service."
- Log audit events, see Section 23.6, "Adding Application Code to Log Audit Events."

16.2.5.2 Auditing Business-Related Activity

An application needs to record business-related activity in the context of a functional flow. Specifically, the application requires logging the users and the business actions performed by them in a particular time interval.

The settings explained in this use case apply to all applications and components in a domain.

This use case requires that applications:

- Create their own audit events based on their business needs

- Be able to log business activities with runtime attributes to audit data repository
- Generate audit reports from audit events
- Manage runtime audit policies
- Modify audit event definitions, if necessary

This use case features:

- Allowing applications to define business functional areas (as audit categories), business activities (as audit events in categories), and attributes in each category.
- Registering applications at deployment; updating audit definitions; deregistering applications after deployment.
- Managing audit artifacts with Fusion Middleware Control or WSLT scripts.

For details about:

- Integrating with CAF, see Section 23.2, "Integrating the Application with the Audit Framework."
- Registering applications, see Section 23.4, "Registering the Application with the Audit Service."
- Log audit events, see Section 23.6, "Adding Application Code to Log Audit Events."
- A sample `component_events.xml` file, see Section 23.3, "Creating Audit Definition Files."
- Managing audit policies, see Section 14.3, "Managing Audit Policies."

16.2.6 Identity Propagation

The identity propagation use cases are the following:

- Propagating the Executing User Identity - Propagating the executing user identity to a web service over SOAP.
- Propagating a User Identity - Propagating a user identity to a web service over SOAP.
- Propagating Identities Across Domains - Propagating a user identity across WebLogic domains.
- Propagating Identities over HTTP - Propagating a user identity over HTTP.

16.2.6.1 Propagating the Executing User Identity

A client application in container needs to propagate the executing user identity to a web service over SOAP; the web service can be running on a different managed server, in the same domain, or in a different domain.

This use case requires that the current executing user identity be propagated to a web service over SOAP.

The features that facilitate this use case are primarily those of Oracle Web Services Manager (OWSM).

For details about OWSM, see *Securing Web Services and Managing Policies with OWSM*.

For details about propagating identities over SOAP, see *Securing Web Services and Managing Policies with OWSM*.

16.2.6.2 Propagating a User Identity

A client application in container needs to propagate a user identity (which is not the executing user identity) to a web service over SOAP; the identity to be propagated is stored in the OPSS security store.

This use case requires that an identity of a user, distinct from the current executing user, be propagated to a web service over SOAP.

This use case features:

- The OPSS security store, where credentials are stored, from where the application gets the specific identity that needs to be propagated as a PasswordCredential.
- Oracle Web Services Manager ability to fetch and propagate the identity to a remote web service.

For details about this use case, see *Securing Web Services and Managing Policies with OWSM*.

16.2.6.3 Propagating Identities Across Domains

A client application in container in a WebLogic domain needs to propagate a user identity (stored in the OPSS security store) to a different WebLogic domain over RMI.

For details about this use case, see section Enabling Trust Between WebLogic Server Domains.

16.2.6.4 Propagating Identities over HTTP

A client application in container (in a WebLogic domain) needs to propagate identities over HTTP.

For requirements and details about this use case, see Propagating Identities with the OPSS Trust Service, the recommended implementation. An alternate implementation (not using the OPSS Keystore Service) is described in Propagating Identities with JKS-Based Key Stores.

16.2.7 Administration and Management

The administration use cases are the following:

- Application Requiring a Central Store - Application requires a central repository of security artifacts where those artifacts are managed.
- Application Requiring Custom Management Tool - Application requires a custom tool to manage a central repository of externalized security artifacts.
- Application Running in a Multiple Server Environment - Application requires modifying security artifacts in a multiple node server environment.

16.2.7.1 Application Requiring a Central Store

An application requires a central repository of policies, credentials, audit configuration, trusts, and keys, and a set of tools to manage that central repository, which is the OPSS security store.

This use case features:

- The OPSS security store
- Managing security artifacts with Fusion Middleware Control
- Managing security artifacts with WLST commands

For details about:

- The OPSS security store, see Section 9.1, "Introduction to the OPSS Security Store."
- Managing security artifacts, see:
  - Section 10.3, "Managing Policies with Fusion Middleware Control"
  - Section 10.4, "Managing Application Policies with WLST commands"
  - Section 11.3, "Managing Credentials with Fusion Middleware Control"
  - Section 11.4, "Managing Credentials with WLST commands"
  - Chapter 12, "Managing Keys and Certificates with the Keystore Service"

16.2.7.2 Application Requiring Custom Management Tool

An application requires a custom tool to manage externalized security artifacts in a context that is meaningful to the application's business.

This use case requires building a custom graphical user interface with calls to OPSS APIs to display and manage security artifacts in the OPSS security store in a context that is meaningful to the application.

This use case features:

- Managing security artifacts with OPSS API

For details about:

- Code sample illustrating the use of the OPSS API to implement some of the operations needed to manage security artifacts, see Implementing a Custom Graphical User Interface.
- The list of OPSS APIs, see Appendix G, "OPSS API References."

16.2.7.3 Application Running in a Multiple Server Environment

Application running in a WebLogic domain where several server instances are distributed across multiple machines requires modifying security artifacts; changes must take effect in all components of the application regardless of where they are running.

This use case features:

- Propagating changes to security artifacts whenever those changes are initiated on the administration server; data on managed server nodes is refreshed based on caching policies.
- Using the MBeans API or Management API to modify security artifacts.

For details about:

- Multiple server nodes, see Section 9.1.1, "Multi-Server Environments"
- OPSS services and MBeans, see Appendix E.2, "Configuring OPSS Services with MBeans"

16.2.8 Integration

The integration use case is the following:

- Application Running in Multiple Domains - Several WebLogic domains sharing a single repository of security artifacts.

16.2.8.1 Application Running in Multiple Domains

A product requires multiple WebLogic domains to run and those domains share a single central OPSS security store.

This use case features:

- OPSS support for several domains to share a security store

For details about:

- Domains sharing a credential store, see Section 11.2, "Encrypting Credentials"
- Using `reassociateSecurityStore` to join to an existing OPSS security store, see Section 10.4.1, "reassociateSecurityStore"

16.3 The OPSS Trust Service

The OPSS trust service allows the propagation of identities across HTTP-enabled applications by providing and validating tokens. The OPSS trust service uses an asserter that is available only on the following platform:

- Oracle WebLogic Application Server - the Identity Asserter

There is one asserter per WebLogic domain.

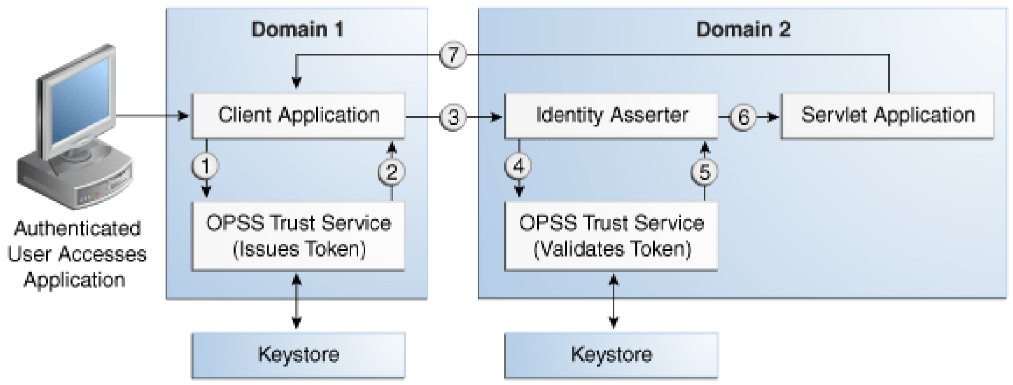
For details about the service configuration properties, see Section F.2.7, "Trust Service Properties." For details about configuring this service with Fusion Middleware Control, see Section 9.7.4, "Configuring the Trust Service Provider," or alternatively, use the WLST command updateTrustServiceConfig. For details about this command, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

16.4 Propagating Identities over the HTTP Protocol

Identity propagation using HTTP calls typically runs as follows (see Figure 16–2):

1. A client application in Domain1 requests a token for an authenticated user from Domain1's OPSS trust service instance.
2. The trust service accesses Domain1’s keystore and issues a token to the client application.
3. The client application encodes the token in an HTML header and dispatches an HTTP request to a servlet application in Domain2. Domain 2's asserter intercepts the request and extracts the token.
4. The asserter requests a validation of that token from Domain2's OPSS trust service instance.
5. The trust service accesses Domain2’s keystore to validate the token and returns a response.
6. Assuming that the validation is successful, the asserter sends the request to the servlet application using the asserted identity.
7. The servlet application sends an HTTP response to the client application request.

Figure 16–2 Identity Propagation with HTTP Calls


Domain 1
Domain 2
Client Application
Identity Asserter
Servlet Application
OPSS Trust Service (Issues Token)
OPSS Trust Service (Validates Token)
Keystore
Keystore
Authenticated User Accesses Application
1
2
3
4
5
6
7

The out-of-the-box configuration sets the key alias based on the WebLogic server name.

Note: The recommended way to implement identity propagation is using the OPSS Trust Service and the OPSS Keystore, as explained in Propagating Identities with the OPSS Trust Service.

16.5 Propagating Identities with the OPSS Trust Service

This section explains how to propagate identities across multiple domains and across containers in a single domain using the OPSS Trust Service. Identity Propagation using the Trust service is supported only on the Oracle Weblogic Server platform.

For details about the OPSS keystore service see Chapter 12, "Managing Keys and Certificates with the Keystore Service."

This section contains code and configuration samples needed to propagate identities on the Oracle WebLogic platform using the OPSS Trust Service, in the following sections:

- Across Multiple WebLogic Domains
- Across Containers in a Single WebLogic Domain
- Embedded Trust Service Provider Properties

16.5.1 Across Multiple WebLogic Domains

In this scenario there are two different domains: Domain1 and Domain2. The client application is running in Domain1; the servlet application is running in Domain2. The client application uses Domain1's OPSS trust service for token generation, and the servlet application uses Domain2's OPSS trust service for token validation.

The samples and configurations required in this scenario are explained in the following sections:

- Token Generation on the Client-Side Domain
- Server Side or Token Validation Domain

16.5.1.1 Token Generation on the Client-Side Domain

On the client side, that is, in the domain where the token is issued, the following are required:

- Developing the Client Application
- Configuring the OPSS Keystore Service
- Adding a TrustServiceAccessPermission Grant
- Configuring the Trust Service Provider

16.5.1.1.1 Developing the Client Application The client application can be a Java SE or Java EE application. The following code sample illustrates the skeleton of a client application:

```
// Authentication type name
public static final String AUTH_TYPE_NAME = "OIT";
// The authenticated username
String user = "weblogic";
// URL of the target application
URL url = "http://<host>:<port>/<destinationApp>";
//-----------------------------------------
JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
JpsContext jpsCtx = ctxFactory.getContext();
final TrustService trustService = jpsCtx.getServiceInstance(TrustService.class);
final TokenManager tokenMgr = trustService.getTokenManager();
final TokenContext ctx = tokenMgr.createTokenContext(
    TokenConfiguration.PROTOCOL_EMBEDDED);
UsernameToken ut = WSSTokenUtils.createUsernameToken("wsuid", user);
GenericToken gtok = new GenericToken(ut);
ctx.setSecurityToken(gtok);
ctx.setTokenType(SAML2URI.ns_saml);
Map<String, Object> ctxProperties = ctx.getOtherProperties();
ctxProperties.put(TokenConstants.CONFIRMATION_METHOD,
    SAML2URI.confirmation_method_bearer);

AccessController.doPrivileged(new PrivilegedAction<String>() {
    public String run() {
        try {
            tokenMgr.issueToken(ctx);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
});

Token token = ctx.getSecurityToken();
String b64Tok = TokenUtil.encodeToken(token);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
connection.setDoOutput(true);
connection.setReadTimeout(10000);
connection.setRequestProperty("Authorization", AUTH_TYPE_NAME + " " + b64Tok);
connection.connect();
BufferedReader rd = new BufferedReader(new InputStreamReader(
    connection.getInputStream()));
StringBuilder sb = new StringBuilder();

String line = null;
while ((line = rd.readLine()) != null) {
    sb.append(line);
}
```

```
connection.disconnect();
System.out.println(sb.toString());
```

16.5.1.1.2 Configuring the OPSS Keystore Service

The trust service keystore should be provisioned with a client certificate and a private key; that certificate is then exported from the keystore and imported into the trust store. Both these stores, the keystore and the trust store, are managed by OPSS KSS in the client domain.

Note: When the trust service property `trust.keystoreType` is set to KSS, it is recommended *not* to use passwords for the keystore or the trust store; in this case, by default, the keystore and trust store used by the trust service are protected by codesource permissions and therefore they do not require password protection.

The following script illustrates these tasks:

```
# Update following values with correct value
user = "<username>"
password = "<password>"
wlsurl = "t3(s)://<host>:<port>"
stripeName = "<stripeNmae>"

#-----------------------------------------------
ksName = "<trustservice_ks>"
tsName = "<trustservice_ts>"
aliasName = "<trustservice>"
issuerDN = "cn=" + aliasName

print "Stripe Name: " + stripeName

print "KeyStore Name: " + ksName
print "TrustStore Name: " + tsName
print "Alias Name: " + aliasName
print "Issuer DN: " + issuerDN

#-----------------------------------------------
connect(user, password, wlsurl)

svc = getOpssService(name='KeyStoreService')
svc.listKeyStores(appStripe=stripeName)

svc.createKeyStore(appStripe=stripeName, name=ksName, password="",
permission=true)
svc.generateKeyPair(appStripe=stripeName, name=ksName, password="", dn=issuerDN,
keysize="1024", alias=aliasName, keypassword="")
svc.exportKeyStoreCertificate(appStripe=stripeName, name=ksName, password="",
alias=aliasName, keypassword="", type="Certificate",
filepath="./trustservice.cer")

svc.createKeyStore(appStripe=stripeName, name=tsName, password="",
permission=true)
svc.importKeyStoreCertificate(appStripe=stripeName, name=tsName, password="",
alias=aliasName, keypassword="", type="TrustedCertificate",
filepath="./trustservice.cer")

svc.listKeyStores(appStripe=stripeName)
svc.listKeyStoreAliases(appStripe=stripeName, name=ksName, password="",
type="Certificate")
exit()
```

16.5.1.1.3 Adding a TrustServiceAccessPermission Grant The following system policy illustrates a codesource grant that allows clients to use the Trust Service APIs, and should be included in the application's `jazn-data.xml` file:

```
<grant>
  <grantee>
    <codesource>

<url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>

<class>oracle.security.jps.service.trust.TrustServiceAccessPermission</class>
      <name>appId=*</name>
      <actions>issue</actions>
    </permission>
  </permissions>
</grant>
```

The grant can be added to the OPSS security store using a WLST command; for details see script grantPermission, in *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

Alternatively, if the application is a Java EE application and the file `jazn-data.xml` is packed with the application, the lifecycle listener can be set so that the grant is migrated to the OPSS security store when the application is deployed; for details, see Section 24.3, "Configuring the Servlet Filter and the EJB Interceptor."

16.5.1.1.4 Configuring the Trust Service Provider The following sample illustrates a configuration of the trust service provider in the `jps-config.xml` file:

```
<propertySet name="trust.provider.embedded">
<property name="trust.keystoreType" value="KSS"/>
  <property name="trust.keyStoreName" value="kss://<stripeName>/<keystoreName>"/>
  <property name="trust.trustStoreName"
value="kss://<stripeName>/<truststoreName>"/>
  <property name="trust.aliasName" value="<aliasName>"/>
  <property name="trust.issuerName" value="<issuerName>"/>
  <property name="trust.provider.className"
value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
  <property name="trust.clockSkew" value="60"/>
  <property name="trust.token.validityPeriod" value="1800"/>
  <property name="trust.token.includeCertificate" value="false"/>
</propertySet>
```

16.5.1.2 Server Side or Token Validation Domain

On the server side, that is, in the domain where the token is received, the following are required:

- Developing the Server Application
- Configuring web.xml
- Configuring the WebLogic Trust Service Asserter

- Provisioning the OPSS Keystore Service
- Adding a TrustServiceAccessPermission
- Configuring the Trust Service Provider

16.5.1.2.1 Developing the Server Application The servlet code can obtain an asserted user as illustrated in the following sample:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   String username = request.getRemoteUser();
   ServletOutputStream out = response.getOutputStream();
   out.print("Asserted username: " + username);
   out.close();
}
```

16.5.1.2.2 Configuring web.xml The file `web.xml` must set the login configuration method to CLIENT-CERT as illustrated in the following sample:

```
<web-app id="WebApp_ID"
…
 <login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>Identity Assertion </realm-name>
 </login-config>
…
</web-app>
```

16.5.1.2.3 Configuring the WebLogic Trust Service Asserter

To configure the WebLogic Trust Service Asserter proceed as follows:

1. Perform one of the following:
   - Login to the WebLogic console as an administrator, and navigate to **Security Realms -> myrealm -> Providers Tab -> Authentication**; then select **TrustServiceIdentityAsserter**.

     This asserter calls the Trust Service APIs to decode and validate the token from the incoming request, and pass the user name to the WebLogic Server to establish the asserted subject.
   - Use the following script:

```
connect("<username>","<password>","t3://<host>:<port>")
edit()
startEdit()
realm = cmo.getSecurityConfiguration().getDefaultRealm()
tsia = realm.lookupAuthenticationProvider("TSIA")
if tsia != None:
    realm.destroyAuthenticationProvider(tsia)
tsia = realm.createAuthenticationProvider("TSIA",
"oracle.security.jps.wls.providers.trust.TrustServiceIdentityAsserter")
save()
activate()
disconnect()
```

16.5.1.2.4 Provisioning the OPSS Keystore Service The client certificate provisioned in the Domain1's key store must be exported from that domain and imported into the Domain2's trust store, as illustrated in the script below.

Note: In case of multi-domain setup, the issuerName set during token generation on the client side is used as the alias name to search for the certificate name on the server side. The certificate in the keystore should be imported with an alias that matches the issuer name on the client side.

```
# Update following values with correct value
user = "<username>"
password = "<password>"
wlsurl = "t3(s)://<host>:<port>"

stripeName = "<stripeName>"

#-----------------------------------------------
ksName = "<trustservice_ks>"
tsName = "<trustservice_ts>"
aliasName = "<trustservice>"

print "Importing certificate for : " + aliasName
print "Stripe Name: " + stripeName

print "TrustStore Name: " + tsName
print "Alias Name: " + aliasName

#-----------------------------------------------
connect(user, password, wlsurl)

svc = getOpssService(name='KeyStoreService')

svc.listKeyStores(appStripe=stripeName)

# switch Trust service to using FKS
svc.createKeyStore(appStripe=stripeName, name=tsName, password="",
permission=true)

svc.importKeyStoreCertificate(appStripe=stripeName, name=tsName, password="",
alias=aliasName, keypassword="", type="TrustedCertificate",
filepath="./trustservice.cer")

svc.listKeyStoreAliases(appStripe=stripeName, name=tsName, password="",
type="TrustedCertificate")

exit()
```

16.5.1.2.5 Adding a TrustServiceAccessPermission The following system policy illustrates a codesource grant that allows clients to validate the Trust Service APIs, and should be included in the application's `jazn-data.xml` file:

```
<grant>
  <grantee>
    <codesource>

<url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
```

```
<class>oracle.security.jps.service.trust.TrustServiceAccessPermission</class>
      <name>appId=*</name>
      <actions>validate</actions>
    </permission>
  </permissions>
</grant>
```

16.5.1.2.6 Configuring the Trust Service Provider The following sample illustrates a configuration of the trust service provider in the `jps-config.xml` file:

```
<propertySet name="trust.provider.embedded">
<property name="trust.keystoreType" value="KSS"/>
  <property name="trust.keyStoreName" value="kss://<stripeName>/<keystoreName>"/>
  <property name="trust.trustStoreName"
value="kss://<stripeName>/<truststoreName>"/>
  <property name="trust.aliasName" value="<aliasName>"/>
  <property name="trust.issuerName" value="<issuerName>"/>
  <property name="trust.provider.className"
value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
  <property name="trust.clockSkew" value="60"/>
  <property name="trust.token.validityPeriod" value="1800"/>
  <property name="trust.token.includeCertificate" value="false"/>
</propertySet>
```

Note: If the server is used only for token validation, the `aliasName` and `issuerName` set in the example above are not used for token validation and are optional attributes. In this case, to validate a token, the trust service looks for the certificate using the issuer name included in the token.

16.5.2 Across Containers in a Single WebLogic Domain

In this scenario the client and server applications run in the same domain; hence both applications can use the same keystore, and therefore *it is not necessary to import the client certificate* (into some other keystore). All other information remains identical to that explained in the multiple-domain scenario.

16.5.3 Embedded Trust Service Provider Properties

The following properties are used to configure the embedded trust service provider in single or multiple domain scenarios:

- `trust.keyStoreType`
- `trust.keyStoreName`
- `trust.trustStoreName`
- `trust.aliasName`
- `trust.issuerName`
- `trust.provider.className`
- `trust.clockSkew`
- `trust.token.validityPeriod`
- `trust.token.includeCertificate`

The following sample illustrates a sample configuration of the embedded trust service provider in the file `jps-config.xml`:

```
<propertySet name="trust.provider.embedded">
  <property name="trust.keystoreType" value="KSS"/>
  <property name="trust.keyStoreName" value="kss://<stripeName>/<keystoreName>"/>
  <property name="trust.trustStoreName"
value="kss://<stripeName>/<truststoreName>"/>
  <property name="trust.aliasName" value="<aliasName>"/>
  <property name="trust.issuerName" value="<aliasName>"/>
  <property name="trust.provider.className"

value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
  <property name="trust.clockSkew" value="60"/>
  <property name="trust.token.validityPeriod" value="1800"/>
  <property name="trust.token.includeCertificate" value="false"/>
</propertySet>
```

To set properties for the embedded trust service provider, use the WLST command `updateTrustServiceConfig`, or alternatively, use Fusion Middleware Control as explained in Section 9.7.4, "Configuring the Trust Service Provider." For details about the command, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

In the configuration explained in Configuring the OPSS Keystore Service, set the values <stripName>, <keystoreName>, <trustStoreName>, <aliasName> accordingly.

16.6 Implementing a Custom Graphical User Interface

This use case illustrates some of the operations needed, for example, when implementing a custom graphic UI to manage policies. The samples presented use the OPSS APIs and demonstrate the following operations:

- Querying users in the identity store.
- Querying application roles in the OPSS security store.
- Querying the mapping of users and groups to application roles; specifically, given a user identify all the application roles mapped to that user (Recall that the mapping of users and groups to application roles is a many-to-many relationship).
- Creating, reading, updating, and deleting the mapping of users and groups to application roles.

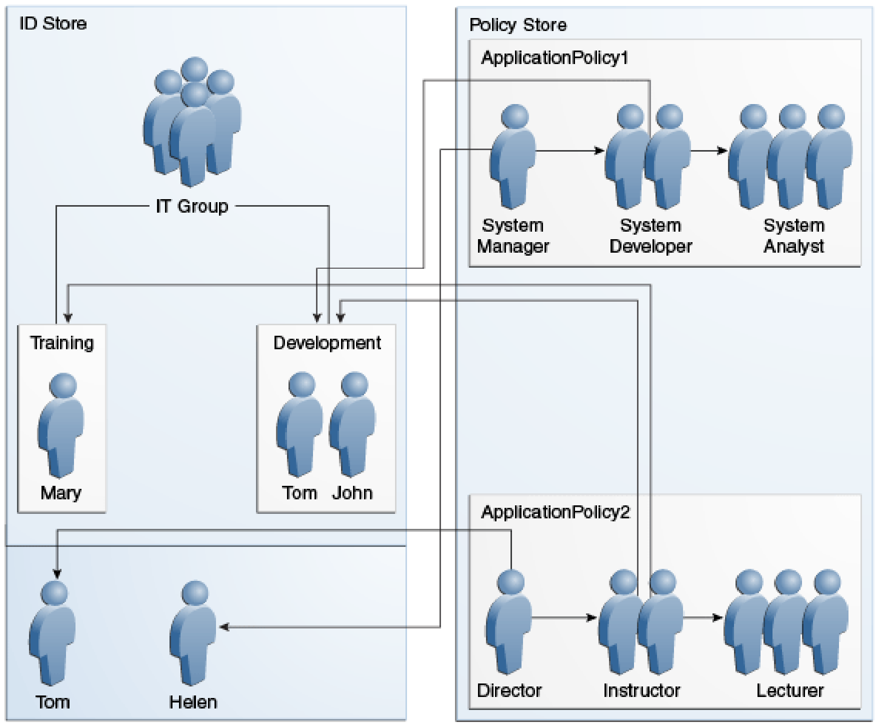
This use case assumes that:

- The identity store is an OID LDAP-based store.
- The OPSS security store is an OID LDAP-based store.
- The identity store contains the following hierarchy of users and groups (enterprise roles):
  - The users Mary, John, Tom, and Helen.
  - The groups IT, Training, and Development.
  - The groups Training and Development are members of the group IT.
  - The user Mary is a member of the group Training.
  - The users Tom and John are members of the group Development.

- The OPSS security store contains the following application policies and hierarchy of application roles:
  - The application policies ApplicationPolicy1 and ApplicationPolicy2.
  - The roles System Manager, System Developer, and System Analyst are application roles referenced in the policy ApplicationPolicy1; the System Manager role is a member of the System Developer role; the System Developer role is a member of the System Analyst role.
  - The roles Director, Instructor, and Lecturer are application roles referenced in the application policy ApplicationPolicy2; the Director role is a member of the Instructor role; the Instructor role is a member of the Lecturer role.
- The mapping of application roles to users and groups is as follows:
  - The role System Manager is mapped to the user Helen.
  - The role System Developer is mapped to the group Development.
  - The role Director is mapped to the user Tom.
  - The role Instructor is mapped to the groups Training and Development.

Figure 16–3 illustrates the hierarchy of application roles, the users and groups, and the mapping of application roles to users and groups, as assumed in this use case.

Figure 16–3 Mapping of Application Roles to Users and Groups



Note that the above role hierarchy implies, for instance, that a user in the System Manager role is also in the System Developer role, and similarly with the other roles. Therefore the role membership for each of the four users is as follows:

- User Tom is a member of the following application roles: System Developer, System Analyst, Director, Instructor, and Lecturer.

- User Helen is a member of the following application roles: System Manager, System Developer, and System Analyst.
- User Mary is a member of the following application roles: Instructor and Lecturer.
- User John is a member of the following application roles: System Developer, System Analyst, Instructor, and Lecturer.

The code samples are detailed in the following sections:

- Imports Assumed - List of imports
- Code Sample 1 - Querying the identity store.
- Code Sample 2 - Creating application roles and assigning members to a role.
- Code Sample 3 - Querying application roles.
- Code Sample 4 - Mapping application roles to users and groups.
- Code Sample 5 - Getting all the roles that have a given user as a member.
- Code Sample 6 - Removing the mapping of an application role to a group.

16.6.1 Imports Assumed

The sample codes in this use case assume the following import statements:

```
import java.security.AccessController;
import java.security.Policy;
import java.security.Principal;
import java.security.PrivilegedExceptionAction;
import java.security.Security;
import java.util.HashSet;
import java.util.List;
import java.util.Set;
import javax.security.auth.Subject;
import oracle.security.idm.Identity;
import oracle.security.idm.IdentityStore;
import oracle.security.idm.ObjectNotFoundException;
import oracle.security.idm.Role;
import oracle.security.idm.RoleManager;
import oracle.security.idm.SearchParameters;
import oracle.security.idm.SearchResponse;
import oracle.security.idm.SimpleSearchFilter;
import oracle.security.idm.User;
import oracle.security.idm.UserProfile;
import oracle.security.jps.ContextFactory;
import oracle.security.jps.JpsContext;
import oracle.security.jps.JpsContextFactory;
import oracle.security.jps.principals.JpsApplicationRole;
import oracle.security.jps.service.idstore.IdentityStoreService;
import oracle.security.jps.service.policystore.ApplicationPolicy;
import oracle.security.jps.service.policystore.PolicyObjectNotFoundException;
import oracle.security.jps.service.policystore.PolicyStore;
import oracle.security.jps.service.policystore.PolicyStoreException;
import oracle.security.jps.service.policystore.entitymanager.AppRoleManager;
import oracle.security.jps.service.policystore.info.AppRoleEntry;
import oracle.security.jps.service.policystore.search.AppRoleSearchQuery;
import oracle.security.jps.service.policystore.search.ComparatorType;
import oracle.security.jps.util.JpsAuth;
import weblogic.security.principal.PrincipalFactory;
```

16.6.2 Code Sample 1

The following sample code illustrates two queries to users in the identity store:

```
private void queryUsers() throws Exception {
        // Using IDM U/R to query ID store
        IdentityStore idmStore = idStore.getIdmStore();

        // Query an individual user by name
        User employee = idmStore.searchUser(USER_TOM);
        log("----------------------------------------------------------");
        log("### Query individual user (Tom) from ID store ###");
        log(USER_TOM + ": " + employee.getName() + " GUID: " +
            employee.getGUID());
        log();

        // Get all users whose name is not "Paul"
        SimpleSearchFilter filter =
            idmStore.getSimpleSearchFilter(UserProfile.NAME,
                                           SimpleSearchFilter.TYPE_NOTEQUAL,
                                           "Paul");
        SearchParameters sps =
            new SearchParameters(filter, SearchParameters.SEARCH_USERS_ONLY);
        SearchResponse result = idmStore.searchUsers(sps);
        log("----------------------------------------------------------");
        log("### Query all users (whose name is not Paul) from ID store ###");
        log("Found the following users:");
        while (result.hasNext()) {
            Identity user = result.next();
            log("\t user: " + user.getName() + ", GUID: " + user.getGUID());
        }
        log();
    }
```

16.6.3 Code Sample 2

The following sample code illustrates how to create an application role and how to make a role a member of another role:

```
private void createAppRoles1() throws Exception {
        AppRoleManager arm1 = ap1.getAppRoleManager();
        log("----------------------------------------------------------");
        log("### Creating app roles in app policy1 with hierachy ###");

        AppRoleEntry sysAnalystRole =
            arm1.createAppRole(APP_ROLE_SYS_ANALYST, APP_ROLE_SYS_ANALYST,
                               APP_ROLE_SYS_ANALYST);
        AppRoleEntry sysDeveloperRole =
            arm1.createAppRole(APP_ROLE_SYS_DEVELOPER, APP_ROLE_SYS_DEVELOPER,
                               APP_ROLE_SYS_DEVELOPER);
        AppRoleEntry sysManagerRole =
            arm1.createAppRole(APP_ROLE_SYS_MANAGER, APP_ROLE_SYS_MANAGER,
                               APP_ROLE_SYS_MANAGER);

        ap1.addPrincipalToAppRole(sysManagerRole, APP_ROLE_SYS_DEVELOPER);
        ap1.addPrincipalToAppRole(sysDeveloperRole, APP_ROLE_SYS_ANALYST);
        log("### App roles in app policy #1 have been created ###");
        log();

    }
```

16.6.4 Code Sample 3

The following code sample illustrates several ways to query application roles:

```
private void queryAppRolesInApplicationPolicy1() throws Exception {
        AppRoleManager arm1 = ap1.getAppRoleManager();

        // Get role that matches a name
        AppRoleEntry are = arm1.getAppRole(APP_ROLE_SYS_MANAGER);
        log("----------------------------------------------------------");
        log("### Query app roles in application policy #1, by name ###");
        log("Found " + are.getName() + " by app role name.");
        log();

        // Get the role that matches a name exactly
        AppRoleSearchQuery q =
            new AppRoleSearchQuery(AppRoleSearchQuery.SEARCH_PROPERTY.NAME,
                                   false, ComparatorType.EQUALITY,
                                   APP_ROLE_SYS_ANALYST,
                                   AppRoleSearchQuery.MATCHER.EXACT);
        List<AppRoleEntry> arel = arm1.getAppRoles(q);
        log("### Query app roles in application policy #1, by exact query ###");
        log("Found " + arel.get(0).getName() + " by exact query.");
        log();

        // Get roles with names that begin with a given string
        q =
   new AppRoleSearchQuery(AppRoleSearchQuery.SEARCH_PROPERTY.NAME, false,
                          ComparatorType.EQUALITY,
                          APP_ROLE_SYS_DEVELOPER.subSequence(0, 7),
                          AppRoleSearchQuery.MATCHER.BEGINS_WITH);
        arel = arm1.getAppRoles(q);
        log("### Query app roles in app policy #1, by begins_with query ###");
        log("Found " + arel.get(0).getName() + " by begins_with query.");
        log();

        // Get roles with names that contain a given substring
        q =
   new AppRoleSearchQuery(AppRoleSearchQuery.SEARCH_PROPERTY.NAME, false,
                          ComparatorType.EQUALITY, "dummy",
                          AppRoleSearchQuery.MATCHER.ANY);
        arel = arm1.getAppRoles(q);
        log("### Query app roles in app policy #1, by matcher any ###");
        log("Found " + arel.size() + " app roles by matcher any.");
        for (AppRoleEntry ar : arel) {
            log("\t" + ar.getName());
        }
        log();
    }
```

16.6.5 Code Sample 4

The following sample illustrates how to map application roles to users and groups:

```
private void assignAppRoleToUsersAndGroups() throws Exception {
        // Obtain the user/group principals
        IdentityStore idmStore = idStore.getIdmStore();
        User tom = idmStore.searchUser(USER_TOM);
        User helen = idmStore.searchUser(USER_HELEN);

        Role trainingRole =
            idmStore.searchRole(IdentityStore.SEARCH_BY_NAME, GROUP_TRAINING);
        Role devRole =
            idmStore.searchRole(IdentityStore.SEARCH_BY_NAME, GROUP_DEV);

        Principal tomPrincipal =
            PrincipalFactory.getInstance().createWLSUser(tom.getName(),
                                                         tom.getGUID(),
                                                         tom.getUniqueName());
        Principal helenPrincipal =
            PrincipalFactory.getInstance().createWLSUser(helen.getName(),
                                                         helen.getGUID(),
                                                         helen.getUniqueName());

        Principal trainingPrincipal =
            PrincipalFactory.getInstance().createWLSGroup(trainingRole.getName(),
                                                          trainingRole.getGUID(),

trainingRole.getUniqueName());
        Principal devPrincipal =
            PrincipalFactory.getInstance().createWLSGroup(devRole.getName(),
                                                          devRole.getGUID(),

devRole.getUniqueName());

        // Application policy #1
        log("----------------------------------------------------------");
        log("### Assigning appl roles to users and groups, app policy #1 ###");
        ap1.addPrincipalToAppRole(helenPrincipal, APP_ROLE_SYS_MANAGER);
        ap1.addPrincipalToAppRole(devPrincipal, APP_ROLE_SYS_DEVELOPER);

        // Application policy #2
        log("### Assigning app roles to users and groups, app policy #2 ###");
        ap2.addPrincipalToAppRole(tomPrincipal, APP_ROLE_DIRECTOR);
        ap2.addPrincipalToAppRole(devPrincipal, APP_ROLE_INSTRUCTOR);
        ap2.addPrincipalToAppRole(trainingPrincipal, APP_ROLE_INSTRUCTOR);

        log("### App roles have been assigned to users and groups ###");
        log();
    }
```

16.6.6 Code Sample 5

The following code sample illustrates how to get all the roles that have a given user as a member:

```
private void showAppRoles() throws Exception {
        Subject tomSubject = getUserSubject(USER_TOM);
        Subject helenSubject = getUserSubject(USER_HELEN);
        Subject johnSubject = getUserSubject(USER_JOHN);
        Subject marySubject = getUserSubject(USER_MARY);

        Set<String> applications = new HashSet<String>();
        applications.add(APPLICATION_NAME1);
        applications.add(APPLICATION_NAME2);

        log("----------------------------------------------------------");
        log("### Query application roles for Tom ###");
        showAppRoles(applications, USER_TOM, tomSubject);
        log();
        log("### Query application roles for Helen ###");
        showAppRoles(applications, USER_HELEN, helenSubject);
        log();

        log("### Query application roles for John ###");
        showAppRoles(applications, USER_JOHN, johnSubject);
        log();

        log("### Query application roles for Mary ###");
        showAppRoles(applications, USER_MARY, marySubject);
        log();
    }

private Subject getUserSubject(String userName) throws Exception {
        Subject subject = new Subject();

        // Query users from ID store using user/role API,for user principal
        IdentityStore idmStore = idStore.getIdmStore();
        User user = idmStore.searchUser(userName);

        Principal userPrincipal =
            PrincipalFactory.getInstance().createWLSUser(user.getName(),
                                                          user.getGUID(),
                                                          user.getUniqueName());

        subject.getPrincipals().add(userPrincipal);

        // Query users from ID store using user/role API, for enterprise roles
        RoleManager rm = idmStore.getRoleManager();
        SearchResponse result = null;
        try {
            result = rm.getGrantedRoles(user.getPrincipal(), false);
        } catch (ObjectNotFoundException onfe) {
            // ignore
        }

        // Add group principals to the subject
        while (result != null && result.hasNext()) {
            Identity role = result.next();
            Principal groupPrincipal =
                PrincipalFactory.getInstance().createWLSGroup(role.getName(),
                                                              role.getGUID(),
                                                         role.getUniqueName());
            subject.getPrincipals().add(groupPrincipal);
        }

        // The subject now contains both user and group principals.
        // In the WebLogic Server, this setting is done by a login module
        return subject;
    }

private void showAppRoles(Set<String> applications, String user, Subject subject)
{
        // Get all granted application roles for this subject
        Set<JpsApplicationRole> result = null;
        try {
            result = JpsAuth.getAllGrantedAppRoles(subject, applications);
        } catch (PolicyStoreException pse) {
            log(pse.toString());
        }
        if (result.size() <= 1) {
            log(user + " has " + result.size() + " application role.");
            if (result.size() == 1) {
                for (JpsApplicationRole ar : result) {
                    log("\tApplication role: " + ar.getName());
                }
            }
        } else {
            System.out.println(user + " has " + result.size() +
                               " application roles.");
            for (JpsApplicationRole ar : result) {
                log("\tApplication role: " + ar.getAppID() + "/" +
                    ar.getName());
            }
        }
    }
```

16.6.7 Code Sample 6

The following sample code illustrates how to remove the mapping of an application role to a group:

```
private void removeAppRoleForUserDirector() throws Exception {
        // Remove instructor role from Dev group
        log("------------------------------------------------------------");
        log("### Removing Instructor application role from Dev group ###");

        IdentityStore idmStore = idStore.getIdmStore();
        Role devRole =
            idmStore.searchRole(IdentityStore.SEARCH_BY_NAME, GROUP_DEV);
        Principal devPrincipal =
            PrincipalFactory.getInstance().createWLSGroup(devRole.getName(),
                                                          devRole.getGUID(),

devRole.getUniqueName());

        ap2.removePrincipalFromAppRole(devPrincipal, APP_ROLE_INSTRUCTOR);
        log("### Instructor app role has been removed from Dev group ###");
        log();

        log("------------------------------------------------------------");
        log("### Now query application roles for user John, again ###");
        Set<String> applications = new HashSet<String>();
        applications.add(APPLICATION_NAME1);
        applications.add(APPLICATION_NAME2);

        Subject johnSubject = getUserSubject(USER_JOHN);
        showAppRoles(applications, USER_JOHN, johnSubject);
        log();
    }
```

16.7 Securing an ADF Application

This section explains the phases that the security of an application goes through. It is assumed that the application uses ADF and that it is developed in the Oracle JDeveloper environment.

The phases of the security lifecycle of an application are the development phase, the deployment phase, and the management phase. The participants are the product manager or application architect, application developers, and application security administrators. For a summary of tasks, see Summary of Tasks per Participant per Phase.

16.7.1 Development Phase

In the development phase developers design the application to work with the full range of security options available in Oracle Fusion Middleware. Developers have access to a rich set of security services exposed by Oracle JDeveloper, the built-in ADF framework, and the Oracle WebLogic Server. All these components are based on OPSS, which ensures a consistent approach to security throughout the application's life span.

Typically, a developer uses the ADF Security Wizard (an authorization editor) and an expression language editor, all within Oracle JDeveloper; additionally and optionally, he may use OPSS APIs to implement more complex security tasks. Thus, some parts of the application use declarative security, others use programmatic security, and they both rely on security features available in the development and run-time environment.

Application developers also define a number of application entitlements and roles (policy seed data) required to secure the application. This policy seed data is kept in a source control system together with the application source code.

16.7.2 Deployment Phase

Once developed, the application is typically tested in a staging environment before being deployed to a production environment. In a production environment, both the application and the run-time services are integrated with other security components, such as user directories, single sign-on systems, user provisioning systems, and auditing. The security services usually change with the phase: for example, during development, a developer relies on a file or Oracle Wallet to store user credentials, but, in a production environment, credentials are stored in an LDAP directory (the OPSS security store).

In the deployment phase, typically, an administrator migrates the policy seed data to the production OPSS security store, and maps application roles to enterprise groups according to application policies.

16.7.3 Management Phase

The management phase starts once an application has been deployed to a production environment. In this phase, application administrators or enterprise security administrators manage day-to-day security tasks, such as granting users access to application resources, reviewing audit logs, responding to security incidents, and applying security patches.

16.7.4 Summary of Tasks per Participant per Phase

The following tables summarize the major responsibilities per participant in each of the security lifecycle phases and Figure 16–4 illustrates the basic flow.

Figure 16–4 Application Lifecycle Phases

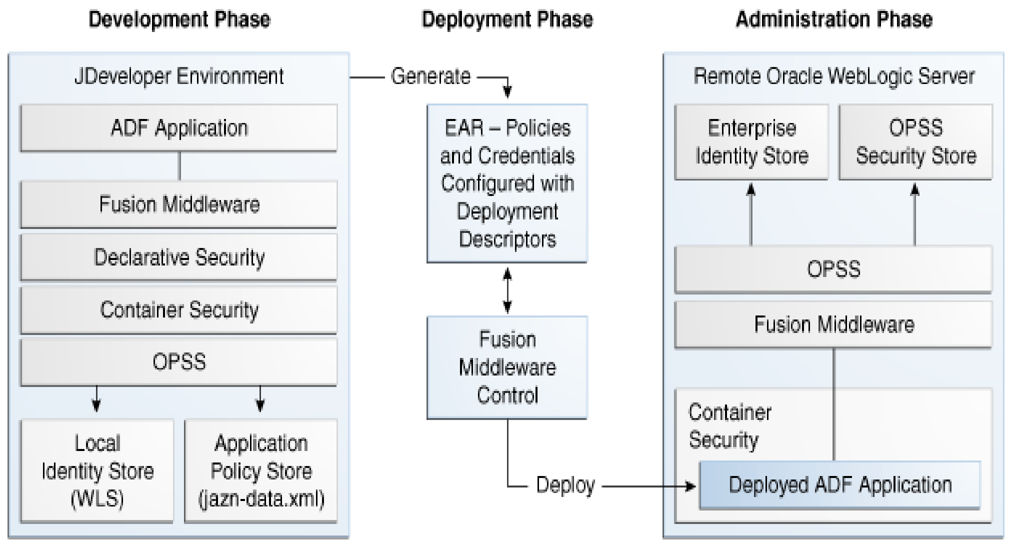


Table 16–1 Security Tasks for the Application Architect

Phase	Task
Development	Defines high-level application roles based on functional security and data security requirements.
	Populates the initial file-based application policy store (`jazn-data.xml`).
Deployment	Defines real-world customer scenarios to be tested by the QA team.
Management	Understands and identifies the requirements to customize application policies.
	Considers defining templates for vertical industries.

Table 16–2 Security Tasks for the Application Developer

Phase	Task
Development	Uses tools and processes, specifically Oracle JDeveloper, to build the application and to create security artifacts, such as application roles and permissions.
	Uses FND Grants to specify data-level security.
	Tests the application using a local policy store with sample users and roles.
Deployment	Assists the QA team to troubleshoot and resolve runtime issues.

Table 16–3 Security Tasks for the Application Security Administrator

Phase	Task
Deployment	Uses deployment services to migrate security seed data in `jazn-data.xml` to the production policy store.
	Maps application roles to enterprise groups so that security policies can be enforced.

Table 16–3 (Cont.) Security Tasks for the Application Security Administrator

Phase	Task
Management	Applies patches and upgrades software, as necessary.
	Manages users and roles, as enterprise users and the application role hierarchy changes overtime.
	Manages policies packed with the application and creates new ones.
	Integrates with and manages the IAM infrastructure.

16.8 Code and Configuration Examples

This section lists most of the code and configuration samples found elsewhere in this Guide, and a fully-written code example.

- Code Examples
- Configuration Examples
- Full Code Example of a Java EE Application with Integrated Security

16.8.1 Code Examples

The following list includes typical security-related programming tasks and links to sample code illustrating implementations:

- Querying an LDAP identity store - See Section 8.4, "Querying the Identity Store Programmatically."
- Querying application roles and the mapping of users and groups to application roles - See Implementing a Custom Graphical User Interface.
- Invoking the method `isUserInRole` - See Section 17.2.2.2, "Programmatic Authorization."
- Managing policies - See Section 17.3.2, "Managing Policies."
- Checking policies - See Section 17.3.3, "Checking Policies."
- Using the class `ResourcePermission` - See Section 17.3.4, "The Class ResourcePermission."
- Using the Identity Store Login Module for authentication in Java SE applications - See Section 25.3.3.1.1, "Using the Identity Store Login Module for Authentication."
- Using the Identity Store Login Module for assertion in Java SE applications - See Section 25.3.3.1.2, "Using the Identity Store Login Module for Assertion."

16.8.2 Configuration Examples

The following list includes typical security-related configuration tasks and links to sample configuration:

- Configuring resource permissions - See Section 17.3.4, "The Class ResourcePermission."
- Configuring the servlet filter and the EJB interceptor - See Section 24.3, "Configuring the Servlet Filter and the EJB Interceptor."
- Configurations involved with `migrateSecurityStore` - See Section 6.6.2.1, "Migrating Policies with migrateSecurityStore," and Section 6.6.2.2, "Migrating Credentials with migrateSecurityStore."

- Configuring an LDAP identity store - See Section 8.3.2.8, "Viewing the Configuration File," and Section 25.3.2, "Configuring an LDAP Identity Store in Java SE Applications."
- Configuring the policy and credential stores in Java SE applications - See Section 18.1, "Configuring the Security Store in Java SE Applications."

16.8.3 Full Code Example of a Java EE Application with Integrated Security

`ezshare` is a full example of a Java EE application whose security has been integrated with OPSS that uses permission-based grants and available at the Oracle Network. To locate the example, search for the keyword ezshare.

16.9 Propagating Identities with JKS-Based Key Stores

The out of box configuration is set so that the token issuer name and the key alias are based on the WebLogic server name. To change these default values, use the procedures explained in Section 16.9.1.8, "Updating the Trust Service Configuration Parameters."

Propagating identities over the HTTP protocol using JKS-based key stores is explained in the following sections:

- Single Domain Scenario
- Multiple Domain Scenario
- Domains Using Both Protocols

16.9.1 Single Domain Scenario

In this scenario, the client and the servlet applications use the same trust service instance to issue and validate tokens. The following sections illustrate code and configuration samples for a client and servlet applications running in the same domain:

- Client Application Code Sample
- Configuring the Keystore Service
- Configuring CSF
- Configuring a Grant
- Servlet Code Sample
- Configuring web.xml
- Configuring the webLogic Asserter and the Trust Service
- Updating the Trust Service Configuration Parameters

16.9.1.1 Client Application Code Sample

The following sample illustrates a client application; note that the file `jps-api.jar` and the OSDT jars: `osdt_ws_sx.jar, osdt_core.jar, osdt_xmlsec.jar, osdt_saml2.jar` must be included the class path for the code sample to compile.

```
// Authentication type name
public static final String AUTH_TYPE_NAME = "OIT";
// The authenticated username
String user = "weblogic";
// URL of the target application
```

```
URL url = "http://<host>:<port>/<destinationApp>";
//------------------------------------------
JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
JpsContext jpsCtx = ctxFactory.getContext();
final TrustService trustService = jpsCtx.getServiceInstance(TrustService.class);
final TokenManager tokenMgr = trustService.getTokenManager();
final TokenContext ctx = tokenMgr.createTokenContext(
    TokenConfiguration.PROTOCOL_EMBEDDED);
UsernameToken ut = WSSTokenUtils.createUsernameToken("wsuid", user);
GenericToken gtok = new GenericToken(ut);
ctx.setSecurityToken(gtok);
ctx.setTokenType(SAML2URI.ns_saml);
Map<String, Object> ctxProperties = ctx.getOtherProperties();
ctxProperties.put(TokenConstants.CONFIRMATION_METHOD,
    SAML2URI.confirmation_method_bearer);

AccessController.doPrivileged(new PrivilegedAction<String>() {
    public String run() {
        try {
            tokenMgr.issueToken(ctx);
        } catch (Exception e) {
            e.printStackTrace();
        }
        return null;
    }
});

Token token = ctx.getSecurityToken();
String b64Tok = TokenUtil.encodeToken(token);

HttpURLConnection connection = (HttpURLConnection) url.openConnection();
connection.setRequestMethod("GET");
connection.setDoOutput(true);
connection.setReadTimeout(10000);
connection.setRequestProperty("Authorization", AUTH_TYPE_NAME + " " + b64Tok);
connection.connect();
BufferedReader rd = new BufferedReader(new InputStreamReader(
    connection.getInputStream()));
StringBuilder sb = new StringBuilder();

String line = null;
while ((line = rd.readLine()) != null) {
    sb.append(line);
}
connection.disconnect();
System.out.println(sb.toString());
```

16.9.1.2 Configuring the Keystore Service

Assuming that the WebLogic server name is `jrfServer_admin`, the following command illustrates the creation of the keystore, represented by the generated file `default-keystore.jks`.

```
JAVA_HOME/bin/keytool -genkeypair
  -alias orakey
  -keypass welcome
  -keyalg RSA
  -dname "CN=jrfServer_admin,O=Oracle,C=US"
  -keystore default-keystore.jks
  -storepass password
```

```
# the generated file must be placed on the domain configuration location
cp default-keystore.jks ${domain.home}/config/fmwconfig
```

Make sure that the keystore service configured in the file `jps-config.xml` points to the generated `default-keystore.jks`; the following sample illustrates a keystore service configuration:

```
<!-- KeyStore Service Instance -->
<serviceInstance name="keystore"
    provider="keystore.provider"  location="./default-keystore.jks">
    <description>Default JPS Keystore Service</description>
    <property name="keystore.provider.type" value="file"/>
    <property name="keystore.file.path" value="./"/>
    <property name="keystore.type" value="JKS"/>
    <property name="keystore.csf.map" value="oracle.wsm.security"/>
    <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
    <property name="keystore.sig.csf.key" value="sign-csf-key"/>
    <property name="keystore.enc.csf.key" value="enc-csf-key"/>
</serviceInstance >
```

16.9.1.3 Configuring CSF

Create a map/key pair used to open the keystore and another map/key pair used to issue tokens. The following commands illustrate these operations using the WLST command `createCred`:

```
// JKS keystore opening password
createCred(map="oracle.wsm.security", key="keystore-csf-key",
           user="keystore", password="password")

// Private key password to issue tokens
createCred(map="oracle.wsm.security", key="sign-csf-key",
           user="orakey", password="password")
```

For details about the WLST command `createCred`, see Section 11.4, "Managing Credentials with WLST commands."

16.9.1.4 Configuring a Grant

Add to the OPSS security store a system policy with a codesource grant like the following, which allows the client application to use the trust service API:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
<class>oracle.security.jps.service.trust.TrustServiceAccessPermission</class>
        <name>appId=*</name>
        <actions>issue</actions>
     </permission>
    </permissions>
</grant>
```

The Oracle WebLogic Server must be stopped and re-started for the above grant to take effect.

16.9.1.5 Servlet Code Sample

The following sample illustrates how a servlet can obtain an asserted user name:

```
public void doGet(HttpServletRequest request, HttpServletResponse response)
   throws ServletException, IOException {
   String username = request.getRemoteUser();
   ServletOutputStream out = response.getOutputStream();
   out.print("Asserted username: " + username);
   out.close();
}
```

16.9.1.6 Configuring web.xml

Set the appropriate login method in the file `web.xml`, as illustrated in the following snippet:

```
<web-app id="WebApp_ID"
…
 <login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>Identity Assertion</realm-name>
 </login-config>
…
</web-app>
```

16.9.1.7 Configuring the webLogic Asserter and the Trust Service

To configure the WebLogic asserter, proceed as follows:

1. Copy the WebLogic identity asserter JAR `jps-wls-trustprovider.jar` to the location `${domain.home}/lib/mbeantypes`, as illustrated by the following command, and then restart the WebLogic Server:

   ```
   cp ${common.components.home}/modules/oracle.jps_
   12.1.3/jps-wls-trustprovider.jar ${domain.home}/lib/mbeantypes
   ```

2. Use WebLogic Console to configure the asserter, as follows:

   1. Login to the console as an administrator.
   2. Navigate to **Security Settings > Security Realms > myrealm > Providers Tab > Authentication**, and click **New** to open the **Create a New Authentication Provider** dialog.
   3. In that dialog, enter `TrustServiceIdentityAsserter` in the name box, and select `TrustServiceIdentityAsserter` from the pull-down in the type box; then click **OK**.

3. Verify that a grant like the following is present in the OPSS security store; this grant is required for the asserter to use the OPSS trust service API; if necessary, use WSLT scripts to specify the following codesource system policy:

   ```
   <grant>
       <grantee>
           <codesource>

   <url>file:${domain.home}/lib/mbeantypes/jps-wls-trustprovider.jar</url>
           </codesource>
       </grantee>
       <permissions>
           <permission>
   ```

```
<class>oracle.security.jps.service.trust.TrustServiceAccessPermission</class>
            <name>appId=*</name>
            <actions>validate</actions>
        </permission>
    </permissions>
</grant>
```

Any changes to the file `jps-config.xml`) requires the server to be re-started before updates take effect.

16.9.1.8 Updating the Trust Service Configuration Parameters

This section explains how to modify the trust service parameters in the file `jps-config.xml` with a script.

Out-of-the-box the values of the parameters `trust.aliasName` and `trust.issuerName` are set to the WebLogic server name. To modify their values to deployment-specific values, use a script like the following:

```
import sys

wlsAdmin = 'weblogic'
wlsPwd ='password_value'
wlUrl='t3://localhost:7001'
issuer= 'issuer'
alias = 'alias'

print "OPSS Trust Service provider configuration management script.\n"

instance = 'trust.provider'
name = 'trust.provider.embedded'
cfgProps = HashMap()
cfgProps.put("trust.issuerName", issuer)
cfgProps.put("trust.aliasName", alias)
pm = PortableMap(cfgProps);

connect(wlsAdmin, wlsPwd, wlUrl)
domainRuntime()

params = [instance, name, pm.toCompositeData(None)]
sign = ["java.lang.String", "java.lang.String",
"javax.management.openmbean.CompositeData"]
on = ObjectName("com.oracle.jps:type=JpsConfig")
mbs.invoke(on, "updateTrustServiceConfig", params, sign)
mbs.invoke(on, "persist", None, None)

print "Done.\n"
```

16.9.2 Multiple Domain Scenario

In this scenario there are two different domains: Domain1 and Domain2. The client application is running in Domain1; the servlet application is running in Domain2. It is assumed that each of these two domains have each a trust store service and keystore properly configured as explained under the heading WebLogic Asserter and Trust Store Configuration in the Single Domain Scenario. In this scenario, the client application uses Domain1's trust service for token generation, and the servlet application uses Domain2's trust service for token validation.

In Domain1, the client sample code and the following configurations are identical to those described in the Single Domain Scenario:

- the client application as illustrated in Client Application Code Sample.
- the configuration of the keystore as illustrated in Configuring the Keystore Service.
- the CSF configuration as illustrated in Configuring CSF.
- the grant configuration as in Configuring a Grant.

In Domain 2, the servlet sample code and `web.xml` configuration are identical to those described in the Single Domain Scenario, but there is some extra setup required:

- The servlet application code as illustrated in Servlet Code Sample.
- The configuration of the file `web.xml` as illustrated in Configuring web.xml.
- The client certificate that is used to sign the token in Domain1 must be present in Domain2's keystore; therefore, the administrator proceeds as follows:

  1. Exports the certificate from Domain 1's keystore, as illustrated by the following command:

     ```
     JAVA_HOME/bin/keytool -export
     -orakey orakey.cer
     -keystore default-keystore.jks
     -storepass password
     ```

  2. Imports the certificate into Domain 2's keystore as illustrated by the command below; note that the alias used to import the certificate must match the name of the issuer on the client side.

     ```
     JAVA_HOME/bin/keytool -importcert
       -alias orakey
       -keypass welcome
       -keyalg RSA
       -keystore default-keystore.jks
       -storepass password
     ```

  3. Sets the Domain2's keystore password in the (Domain2's) credential store using the WLST command `createCred` as follows:

     ```
     createCred(map="oracle.wsm.security", key="keystore-csf-key",
     user="keystore", password="password")
     ```

  For details about this command, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

16.9.3 Domains Using Both Protocols

In this scenario, applications use either the HTTP protocol or the SOAP protocol, and not all applications in the domain use the same protocol. In such scenario, the keystore can be shared by the trust service used by the HTTP protocol and the SOAP service used by Oracle Web Services Manager. But in order for the trust service to work in this case, some special configurations in the file `jps-config.xml` are required as explained in the following sections:

- Single Domain Scenario
- Multiple Domain Scenario

16.9.3.1 Single Domain Scenario

In this scenario, there is one keystore. The following snippet illustrates the configuration required for a certificate with alias `orakey`:

```
<propertySet name="trust.provider.embedded">
    <property name="trust.provider.className"

value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
    <property name="trust.clockSkew" value="60"/>
    <property name="trust.token.validityPeriod" value="1800"/>
    <property name="trust.token.includeCertificate" value="false"/>

    <!-- The alias used to get the signing certificate from JKS -->
    <property name="trust.aliasName" value="orakey"/>

    <!-- The issuer name to be added in the token used by the destination
    trust service instance as an alias to pick up the corresponding certificate
    to validate the token signature -->
    <property name="trust.issuerName" value="orakey"/>
</propertySet>
```

16.9.3.2 Multiple Domain Scenario

In this scenario, there are two domains and two keystores. The following snippet illustrates the configuration required in the domain that is issuing tokens for a certificate with alias `orakey`:

```
<!-- issuer domain trust store must have a signing certif. w. alias orakey -->
<propertySet name="trust.provider.embedded">
    <property name="trust.provider.className"

value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
    <property name="trust.clockSkew" value="60"/>
    <property name="trust.token.validityPeriod" value="1800"/>
    <property name="trust.token.includeCertificate" value="false"/>

    <!-- the signing certificate alias in local JKS -->
    <property name="trust.aliasName" value="orakey"/>

    <!-- the token issuer’s name -->
    <property name="trust.issuerName" value="domain1"/>
</propertySet>
```

Note 1: On the client side, the value of `trust.issuerName` can be same as `trust.aliasName`. However the issuer name value can be overridden by setting a different value for `trust.issuerName` (as shown in the sample above). This issuer name will be set in the token generated on the client side.

Note 2: On the server side, if the server is used only for token validation, it is not mandatory to set `trust.aliasName` and `trust.issuerName`. The issuer name set during the token generation is used while looking for a certificate on the server side. Hence the certificate imported from the client should be exported on the server side with the client side issuer name as the alias ('domain1' in the example above).

The following snippet illustrates the configuration required in the domain that is receiving tokens for a certificate with alias `orakey`:

```
<!- important: recipient domain must have a token validation certificate for
domain1,
which is the one was used to sign the token with alias "domain1" -->
<propertySet name="trust.provider.embedded">
    <property name="trust.provider.className"

value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
    <property name="trust.clockSkew" value="60"/>
    <property name="trust.token.validityPeriod" value="1800"/>
    <property name="trust.token.includeCertificate" value="false"/>

    <!-- the signing certificate alias in local JKS -->
    <property name="trust.aliasName" value="orakey"/>

    <!-- the token issuer's name -->
    <property name="trust.issuerName" value="domain2"/>
</propertySet>
```

17 The OPSS Policy Model

This chapter explains the OPSS policy and authorization models.

The topics are presented in the following sections:

- The Security Policy Model
- Authorization Overview
- The JAAS/OPSS Authorization Model

17.1 The Security Policy Model

For details about the OPSS policy model and the security artifacts used in it, see *Oracle Fusion Middleware Administrator's Guide for Oracle Entitlements Server*.

17.2 Authorization Overview

This section compares and contrasts the authorization available in the Java EE and the JAAS models, in the following sections:

- Introduction to Authorization
- The Java EE Authorization Model
- The JAAS Authorization Model

17.2.1 Introduction to Authorization

A Java 2 policy specifies the permissions granted to signed code loaded from a given location. A JAAS policy extends Java 2 grants by allowing an optional list of principals, permissions are granted only to code from a given location, possibly signed, and run by a user represented by those principals.

The OPSS security store is a repository of system and application-specific policies and roles. Application roles can be granted (mapped) to enterprise users and groups specific to the application (such as administrative roles). A policy can grant permissions to any of these roles, groups, or users as principals.

For more details about policy-related security artifacts, see Chapter 3.3, "Policy Store Basics."

An application can delegate the enforcement of authorization to the container, or it can implement its own enforcement of policy checking with calls to methods such as `checkPermission`, `checkBulkAuthorization`, or `getGrantedResources`.

For details about policy checking with API calls, see Checking Policies.

17.2.2 The Java EE Authorization Model

The Java EE authorization model uses role membership to control access to EJB methods and web resources that are referenced by URLs; policies assign permissions to users and roles, and they are enforced by the container to protect resources.

In the Java EE model, authorization is implemented in either of the following ways:

- Declaratively, where authorization policies are specified in deployment descriptors; the container reads those policies from deployment descriptors and enforces them. No special application code is required to enforce authorization.
- Programmatically, where authorization policies are checked in application code; the code checks whether a subject has the appropriate permission to execute specific sections of code. If the subject fails to have the proper permission, the code throws an exception.

Table 17–1 shows the advantages and disadvantages of each approach.

Table 17–1 Comparing Authorization in the Java EE Model

Authorization Type	Advantages	Disadvantages
Declarative	No coding needed; easy to update by modifying just deployment descriptors.	Authorization is coarse-grained and specified at the URL level or at the method level (for EJBs).
Programmatic	Specified in application code; can protect code at a finer levels of granularity.	Not so easy to update, since it involves code changes and recompilation.

A container can provide authorization to applications running in it in two ways: declaratively and programmatically; these topics and an example are explained in the following sections:

- Declarative Authorization
- Programmatic Authorization
- Java EE Code Example

17.2.2.1 Declarative Authorization

Declarative authorization allows to control access to URL-based resources (such as servlets and pages) and methods in EJBs.

The basic steps to configure declarative authorization are the following:

1. In standard deployment descriptors, specify the resource to protect, such as a web URL or an EJB method, and a logical role that has access to the resource.

   Alternatively, since Java EE 1.5 supports annotations, use code annotations instead of deployment descriptors.

2. In proprietary deployment descriptors (such as `web.xml`), map the logical role defined in step 1 to an enterprise group.

17.2.2.2 Programmatic Authorization

Programmatic authorization provides a finer grained authorization than the declarative approach, and it requires that the application code invoke the method `isUserInRole` (for servlets and JSPs) or the method `isCallerInRole` (for EJBs), both available from standard Java APIs.

Although these methods still depend on role membership to determine authorization, they give finer control over authorization decisions since the controlling access is not limited at the resource level (EJB method or URL).

17.2.2.3 Java EE Code Example

The following example illustrates a servlet calling the method `isUserInRole`. It is assumed that the EAR file packing the servlet includes the configuration files `web.xml` and `weblogic-application.xml`, and that these files include the following configuration fragments:

web.xml

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

weblogic-application.xml

The following snippet shows the mapping between the user `weblogic` and the security role `sr_developer`:

```
<wls:security-role-assignment>
  <wls:role-name>sr_developer</wls:role-name>
  <wls:principal-name>weblogic</wls:principal-name>
</wls:security-role-assignment>
```

Code Example Invoking isUserInRole

```
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.Date;

public class PolicyServlet extends HttpServlet {

 public PolicyServlet() {
        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( "<br>request.getRemoteUser = " + request.getRemoteUser() +
"<br>");
        out.println("request.isUserInRole('sr_developer') = " +
request.isUserInRole("sr_developer") + "<br>");
```

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

17.2.3 The JAAS Authorization Model

The JAAS authorization introduces permissions but can still use the notion of roles. An authorization policy binds permissions with a Subject (role, group, or user) and, optionally, with source code. Granting to a role is achieved through calls to `addPrincipalsToAppRole`.

Permissions are evaluated by calls to the static method `AccessController.checkPermission`, and the model allows fine-grained control to resources.

In this model, an authorization policy specifies the following information:

- Application roles and enterprise groups.
- Permissions granted to users, groups (in application policies), and code sources (in system policies). For users and groups, they determine what a user or the member of a group is allowed to access. For code sources, they determine what actions the code is allowed to perform.

When programming with this model, sensitive lines of code are preceded with calls to check whether the current user or role is granted the appropriate permissions to access the code. If the user has the appropriate permissions, the code is run. Otherwise, the code throws and exception.

For an example of a code source grant with multiple permissions, see Section 24.6.6, "Using Supported Permission Classes."

17.3 The JAAS/OPSS Authorization Model

JAAS/OPSS authorization is based on controlling the operations that a class can perform when it is loaded and run in the environment.

This section is divided into the following sections:

- The Resource Catalog
- Managing Policies
- Checking Policies
- The Class ResourcePermission

17.3.1 The Resource Catalog

OPSS supports the specification and runtime support of the resource catalog in file-, LDAP-, and DB-based OPSS security stores.

Using the resource catalog provides the following benefits:

- Describes policies and secured artifacts in human-readable terms.
- Allows defining and modifying policies independently of and without knowledge of the application source code.
- Allows browsing and searching secured artifacts.

- Allows grouping of secured artifacts in building blocks (entitlements or permission sets) which can be later used in authorization policies.

17.3.2 Managing Policies

Resource catalog artifacts can be managed with the policy management API. Specifically, the following interfaces, all subinterfaces of the interface `oracle.security.jps.service.policystore.EntityManager`, are directly relevant to the artifacts in the resource catalog:

- `GrantManager` - This interface includes methods to query grants using search criteria, to obtain list of grants that satisfy various combinations of resource catalog artifacts, and to grant or revoke permissions to principals.
- `PermissionSetManager` - This interface includes methods to create, modify, and query permission sets (entitlements).
- `ResourceManager` - This interface includes methods to create, delete, and modify resource (instances).
- `ResourceTypeManager` - This interface includes methods to create, delete, modify, and query resource types.

For details about these interfaces, see the Javadoc document *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services*.

The following code snippet illustrates the creation of a resource type, a resource instance, actions, and a permission set:

```
import oracle.security.jps.service.policystore.entitymanager.*;
import oracle.security.jps.service.policystore.search.*;
import oracle.security.jps.service.policystore.info.resource.*;
import oracle.security.jps.service.policystore.info.*;
import oracle.security.jps.service.policystore.*;
import java.util.*;

public class example {
  public static void main(String[] args) throws Exception {
     ApplicationPolicy ap;

     ResourceTypeManager rtm = ap.getEntityManager(ResourceTypeManager.class);
     ResourceTypeSearchQuery query = new ResourceTypeSearchQuery();
     query.setANDMatch();
     query.addQuery(ResourceTypeSearchQuery.SEARCH_PROPERTY.NAME, false,
     ComparatorType.EQUALITY, "resourceType", BaseSearchQuery.MATCHER.EXACT);
     List<ResourceTypeEntry> allResourceTypes = rtm.getResourceTypes(query);

     ResourceManager rm = ap.getEntityManager(ResourceManager.class);
     ResourceSearchQuery ResourceQuery = new ResourceSearchQuery();
     ResourceQuery.setANDMatch();
     ResourceQuery.addQuery(ResourceSearchQuery.SEARCH_PROPERTY.NAME, false,
     ComparatorType.EQUALITY, "R2", BaseSearchQuery.MATCHER.EXACT);
     List<ResourceEntry> allResources = rm.getResources("RT2", ResourceQuery);

     PermissionSetManager psm = ap.getEntityManager(PermissionSetManager.class);
     PermissionSetSearchQuery pssq = new PermissionSetSearchQuery();
     pssq.setANDMatch();
     pssq.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.NAME, false,
     ComparatorType.EQUALITY, "PS1", BaseSearchQuery.MATCHER.EXACT);
     List<PermissionSetEntry> allPermSets = psm.getPermissionSets(pssq);
```

```
        RoleCategoryManager rcm = ap.getEntityManager(RoleCategoryManager.class);
        RoleCategorySearchQuery rcsq = new RoleCategorySearchQuery();
        rcsq.setANDMatch();
        rcsq.addQuery(RoleCategorySearchQuery.SEARCH_PROPERTY.NAME, false,
        ComparatorType.EQUALITY, "roleCategoryCartoon",
        BaseSearchQuery.MATCHER.EXACT);

        List<RoleCategoryEntry> allRoleCategories = rcm.getRoleCategories(rcsq);
    }
}
```

The following code snippet illustrates a complex query involving resource catalog elements:

```
//ApplicationPolicy ap as in the preceeding example
ResourceTypeManager rtm = ap.getEntityManager(ResourceTypeManager.class);
ResourceTypeSearchQuery query = new ResourceTypeSearchQuery();
query.setANDMatch();
query.addQuery(ResourceTypeSearchQuery.SEARCH_PROPERTY.NAME, false,
ComparatorType.EQUALITY, "resourceType", BaseSearchQuery.MATCHER.EXACT);
List<ResourceTypeEntry> enties = rtm.getResourceTypes(query);

ResourceManager rm = ap.getEntityManager(ResourceManager.class);
ResourceSearchQuery ResourceQuery = new ResourceSearchQuery();
ResourceQuery.setANDMatch();
ResourceQuery.addQuery(ResourceSearchQuery.SEARCH_PROPERTY.NAME, false,
ComparatorType.EQUALITY, "R2", BaseSearchQuery.MATCHER.EXACT);
ArrayList<BaseSearchQuery> querries = ResourceQuery.getQueries();
List<ResourceEntry> resources = rm.getResources("RT2", ResourceQuery);

PermissionSetManager psm = ap.getEntityManager(PermissionSetManager.class);
PermissionSetSearchQuery pssq = new PermissionSetSearchQuery();
pssq.setANDMatch();
pssq.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.NAME, false,
ComparatorType.EQUALITY, "PS1", BaseSearchQuery.MATCHER.EXACT);
List<PermissionSetEntry> psets = psm.getPermissionSets(pssq);

RoleCategoryManager rcm = ap.getEntityManager(RoleCategoryManager.class);
RoleCategorySearchQuery rcsq = new RoleCategorySearchQuery();
rcsq.setANDMatch();
rcsq.addQuery(RoleCategorySearchQuery.SEARCH_PROPERTY.NAME, false,
ComparatorType.EQUALITY, "roleCategoryCartoon", BaseSearchQuery.MATCHER.EXACT);
ArrayList<BaseSearchQuery> queries = rcsq.getQueries();
List<RoleCategoryEntry> rcs = rcm.getRoleCategories(rcsq);
```

The following code sample illustrates how to create a grant:

```
GrantManager gm = ap.getEntityManager(GrantManager.class);
Set<PrincipalEntry> pe = new HashSet<PrincipalEntry>();
List<AppRoleEntry> are = ap.searchAppRoles(appRoleName);
pe.addAll(are);
gm.grant(pe, null, permissionSetName);
```

17.3.3 Checking Policies

This section illustrates several ways to check policies programmatically, in the following sections:

- Using the Method checkPermission
- Using the Methods doAs and doAsPrivileged

- Using the Method checkBulkAuthorization
- Using the Method getGrantedResources

Important Notes:

1. By default, authorization failures are not visible in the console. To have authorization failures sent to the console you must set the system variable `jps.auth.debug` as follows: `-Djps.auth.debug=true`

In particular, to have `JpsAuth.checkPermission` failures sent to the console, you must set the variable as stated above.

2. The OPSS policy provider must be explicitly set in Java SE applications, as illustrated in the following snippet:

```
java.security.Policy.setPolicy(new
oracle.security.jps.internal.policystore.JavaPolicyProvider())
```

Not setting the policy provider explicitly in a Java SE application may cause runtime methods (such as `JpsAuth.checkPermission`) to return incorrect values.

17.3.3.1 Using the Method checkPermission

Oracle Fusion Middleware supports the use of the method `checkPermission` in the classes `java.security.AccessController` and `oracle.security.jps.util.JpsAuth`.

Oracle recommends the use of `checkPermission` in the class `JpsAuth` because it provides better debugging support, better performance, and audit support.

The static method `AccessController.checkPermission` uses the default access control context (the context inherited when the thread was created). To check permissions on some other context, call the instance method `checkPermission` on a particular `AccessControlContext` instance.

The method `checkPermission` behaves according to the value of the JAAS mode (see JAAS mode in Chapter 24.3, "Configuring the Servlet Filter and the EJB Interceptor"), as listed in the following table:

Table 17–2 Behavior of checkPermission According to JAAS Mode

JAAS Mode Setting	checkPermission
off or undefined	Enforces codebase security based on the security policy in effect, and there is no provision for subject-based security.
doAs	Enforces a combination of codebase and subject-based security using the access control context created through the `doAs` block.
doAsPrivileged	Enforces subject-based security using a null access control context.
subjectOnly	Takes into consideration grants involving principals *only* (and it disregards those involving codebase) when evaluating a permission.

Note: If `checkPermission` is called inside a `doAs` block and the check permission call fails, to display the failed protection domain you must set the system property `java.security.debug=access,failure`.

The following example illustrates a servlet checking a permission. It is assumed that the EAR file packing the servlet includes the configuration files `jazn-data.xml` and `web.xml`.

jazn-data.xml

The application file-based policy store is as follows:

```
<?xml version="1.0" ?>
<jazn-data>
  <policy-store>
    <applications>
      <application>
        <name>MyApp</name>

        <app-roles>
        <app-role>
          <name>AppRole</name>
          <display-name>AppRole display name</display-name>
          <description>AppRole description</description>
          <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
          <class>oracle.security.jps.service.policystore.ApplicationRole</class>
        </app-role>
      </app-roles>

      <resource-types>
        <resource-type>
          <name>MyResourceType</name>
          <display-name>MyResourceType display name</display-name>
          <description>MyResourceType description</description>
          <provider-name>MyResourceType provider</provider-name>
          <matcher-class>oracle.security.jps.ResourcePermission</matcher-class>
          <actions-delimiter>,</actions-delimiter>
          <actions>write,read</actions>
        </resource-type>
      </resource-types>

      <resources>
        <resource>
          <name>MyResource</name>
          <display-name>MyResource display name</display-name>
          <description>MyResource description</description>
          <type-name-ref>MyResourceType</type-name-ref>
        </resource>
      </resources>

      <permission-sets>
        <permission-set>
          <name>MyEntitlement</name>
          <display-name>MyEntitlement display name</display-name>
          <description>MyEntitlement description</description>
          <member-resources>
            <member-resource>
              <type-name-ref>MyResourceType</type-name-ref>
              <resource-name>MyResource</resource-name>
              <actions>write</actions>
            </member-resource>
          </member-resources>
        </permission-set>
      </permission-sets>

      <jazn-policy>
        <grant>
          <grantee>
            <principals>
              <principal>
                <class>
              oracle.security.jps.service.policystore.ApplicationRole</class>
                <name>AppRole</name>
                <guid>F5494E409CFB11DEBFEBC11296284F58</guid>
              </principal>
            </principals>
          </grantee>

          <!-- entitlement-based permissions -->
          <permission-set-refs>
            <permission-set-ref>
              <name>MyEntitlement</name>
            </permission-set-ref>
          </permission-set-refs>
        </grant>
      </jazn-policy>
    </application>
  </applications>
 </policy-store>
 <jazn-policy></jazn-policy>
</jazn-data>
```

web.xml

The filter `JpsFilter` is configured as follows:

```
<web-app>
 <display-name>PolicyTest: PolicyServlet</display-name>
 <filter>
  <filter-name>JpsFilter</filter-name>
  <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
   <init-param>
    <param-name>application.name</param-name>
    <param-value>PolicyServlet</param-value>
   </init-param>
  </filter>
  <filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>PolicyServlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
  </filter-mapping>...
```

Code Example

In the following example, `Subject.doAsPrivileged` may be replaced by `JpsSubject.doAsPrivileged`:

```
import javax.security.auth.Subject;
import javax.servlet.ServletConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.StringWriter;
import java.security.*;
import java.util.Date;
import java.util.PropertyPermission;
import java.io.FilePermission;

public class PolicyServlet extends HttpServlet {

 public PolicyServlet() {
        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( "<br>request.getRemoteUser = " + request.getRemoteUser() +
"<br>");
        out.println("request.isUserInRole('sr_developer') = " +
request.isUserInRole("sr_developer") + "<br>");
        out.println("request.getUserPrincipal = " + request.getUserPrincipal() +
"<br>");

 Subject s = null;
        s = Subject.getSubject(AccessController.getContext());

        out.println("Subject in servlet " + s);
        out.println("<br>");
        final RuntimePermission rtPerm = new RuntimePermission("getClassLoader");
 try {
 Subject.doAsPrivileged(s, new PrivilegedAction() {
                public Object run() {
                    try {
                        AccessController.checkPermission(rtPerm);
                        out.println("<br>");
                        out.println("CheckPermission passed for permission: " +
rtPerm+ " seeded in application policy");
                        out.println("<br>");
                    } catch (IOException e) {
                        e.printStackTrace();
                        printException ("IOException", e, out);
                    } catch (AccessControlException ace) {
                        ace.printStackTrace();
                        printException ("Accesscontrol Exception", ace, out);
                    }
                    return null;
                }
            }, null);

} catch (Throwable e) {
            e.printStackTrace();
            printException("application policy check failed", e, out);
        }
        out.println("</BODY>");
        out.println("</HTML>");
    }

 void printException(String msg, Throwable e, ServletOutputStream out) {
        Throwable t;
        try {
            StringWriter sw = new StringWriter();
            PrintWriter pw = new PrintWriter(sw, true);
            e.printStackTrace(pw);

            out.println("<p>" + msg + "<p>");
            out.println("<code>");
            out.println(sw.getBuffer().toString());
            t = e;
            /* Print the root cause */
            while ((t = t.getCause()) != null) {
                sw = new StringWriter();
                pw = new PrintWriter(sw, true);
                t.printStackTrace(pw);

                out.println("<hr>");
                out.println("<p>  Caused By ... </p>");
                out.println(sw.getBuffer().toString());
            }
            out.println("</code><p>");
        } catch (IOException ioe) {
            ioe.printStackTrace();
        }
    }
}
```

17.3.3.2 Using the Methods doAs and doAsPrivileged

Oracle Fusion Middleware supports the methods `doAs` and `doAsPrivileged` in the standard class `javax.security.auth.Subject`.

Oracle recommends, however, the use of these methods in the class `oracle.security.jps.util.JpsSubject` because they render better performance and provide auditing.

> **Note:** If `checkPermission` is called inside a `doAs` block and the check permission call fails, to display the failed protection domain you must set the system property `java.security.debug=access,failure`.

17.3.3.3 Using the Method checkBulkAuthorization

The method `checkBulkAuthorization` determines whether a Subject has access to one or more resource actions. Specifically, the method returns the set of resource actions the passed Subject is authorized to access in the passed resources.

When invoking this method (in a Java SE application), make sure that:

1. The system property `java.security.policy` has been set to the location of the OPSS/Oracle WebLogic Server policy file.
2. Your application *must* call first the method `setPolicy` to explicitly set the policy provider, as illustrated in the following lines:

   ```
   java.security.Policy.setPolicy(new
   oracle.security.jps.internal.policystore.JavaPolicyProvider())
   ```

3. Your application calls `checkBulkAuthorization`() after the call to `setPolicy`.

In any application, checkBulkAuthorization assumes that the caller can provide:

- A Subject with User and Enterprise Role Principals.
- A list of resources including the stripe each resource belongs to.

Grants using resource permissions must include the required resource type.

checkBulkAuthorization also assumes that the application has visibility into the OPSS security store stripes configured in the domain where the application is running.

`checkBulkAuthorization` does not require resources to be present in the OPSS security store.

17.3.3.4 Using the Method getGrantedResources

The method `getGrantedResources` provides a runtime authorization query to fetch all granted resources on a given Subject by returning the resource actions that have been granted to the Subject; only permissions associated with resource types (directly or indirectly through permission sets) are returned by this method, and it is available only when the OPSS security store is LDAP-based.

17.3.4 The Class ResourcePermission

A permission class provides the means to control the actions that a grantee is allowed on a resource. Even though a custom permission class provides the application designer complete control over the actions, target matching, and the "implies" logic, to work as expected at runtime, a custom permission class must be specified in the system classpath of the server so that it is available and can be loaded when required. But modifying the system class path in environments is difficult and, in some environments, such modification might not be even possible.

OPSS includes the class `oracle.security.jps.ResourcePermission` that can be used as the permission class within any application grant to protect application or system resources. Therefore, the application developer no longer needs to write custom permission classes, since the class `ResourcePermission` is available out-of-the-box and can be readily used in permissions within application grants stored in any supported policy provider. This class is not designed to be used in system policies, but only in application policies.

Configuring Resource Permissions

A permission that uses the class `ResourcePermission` is called a *resource permission*, and it specifies the resource type, the resource name, and an optional list of actions according to the format illustrated in the following XML sample:

```
<permission>
  <class>oracle.security.jps.ResourcePermission</class>
  <name>resourceType=type,resourceName=name</name>
```

```
  <actions>character-separated-list-of-actions</actions>
</permission>
```

The above specification *requires* that the resource type encoded in the type name be defined. Even though the resource type information is not used at runtime, its definition must be present for a resource permission to be migrated successfully; moreover, resource types help administrators model resources and manage their use.

The following fragments illustrate the specifications of resource permissions and the corresponding required resource types:

```
<permission>
  <class>oracle.security.jps.ResourcePermission</class>
  <name>resourceType=epm.calcmgr.permission,resourceName=EPM_Calc_Manager</name>
</permission>

<resource-types>
  <resource-type>
    <name>epm.calcmgr.permission</name>
    <display-name>CalcManager ResourceType</display-name>
    <description>Resourcetype for managing CalcManager grants</description>
    <provider-name></provider-name>
    <matcher-class>oracle.security.jps.ResourcePermission</matcher-class>
    <actions-delimiter>,</actions-delimiter>
    <actions></actions>
  </resource-type>
</resource-types>

<permission>
  <class>oracle.security.jps.ResourcePermission</class>
  <name>resourceType=oracle.bi.publisher.Reports,resourceName=GLReports</name>
  <actions>develop;schedule</actions>
</permission>

<resource-types>
  <resource-type>
    <name>oracle.bi.publisher.Reports</name>
    <display-name>BI Publisher Reports</display-name>
    <provider-name></provider-name>
    <matcher-class>oracle.security.jps.ResourcePermission</matcher-class>
    <actions-delimiter>;</actions-delimiter>
    <actions>view;develop;schedule</actions>
  </resource-type>
</resource-types>
```

Note that a resource type associated with a resource permission can have an empty list of actions. The following important points apply to a resource permission:

- The name must conform to the following format:

  `resourceType=aType,resourceName=aName`

  The resource type of a resource permission must be defined and it is returned by the method `ResourcePermission.getType()`.

- The character-separated list of actions is optional; if specified, it must be a subset of the actions specified in the associated resource type. This list is returned by the method `ResourcePermission.getActions()`.

  The character used to separate the items of the list must equal to the character specified in the <actions-delimiter> of the associated resource type.

- The display name of a resource used in a permission is returned by the method `ResourcePermission.getResourceName()`.
- No wildcard use is supported in a resource permission.

Managing and Checking Resource Permissions

The code snippet below illustrates the instantiation of a resource permission and how to check it programmatically; the following code snippet is based on one of the configuration examples described in Configuring Resource Permissions:

```
ResourcePermission rp =
   new ResourcePermission("oracle.bi.publisher.Reports","GLReports","develop");
JpsAuth.checkPermission(rp);
```

At runtime the permission check will succeed if the resource permission satisfies all the following four conditions:

- The permission is an instance of the class `ResourcePermision`.
- The resource type name (first argument) matches (ignoring case) the name of a resource type.
- The resource (second argument) name matches exactly the name of a resource instance.
- The list of actions (third argument) is a comma-separated subset of the set of actions specified in the resource type.

About the Matcher Class for a Resource Type

When creating a resource type, a matcher class can be optionally supplied. If unspecified, it defaults to `oracle.security.jps.ResourcePermission`.

If, however, two or more resource types are to share the same resource matcher class, then that class must be one of the following:

- The class `oracle.security.jps.ResourcePermission`.
- A concrete class extending the abstract class `oracle.security.jps.AbstractTypedPermission`, as illustrated by the class MyAbstractTypedPermission in the following sample:

  ```
  public class MyAbstractTypedPermission extends AbstractTypedPermission {
     private static final long serialVersionUID = 8665318227676708586L;
     public MyAbstractTypedPermission(String resourceType,
                                      String resourceName,
                                      String actions) {super(resourceType,
  resourceName, actions);
      }
  }
  ```
- A class implementing the class `oracle.security.jps.TypePermission` and extending the class `java.security.Permission`.

18 Developing with the Authorization Service

This chapter explains how to use OPSS authorization in Java SE applications and lists some unsupported methods.

These topics are presented in the following sections:

- Configuring the Security Store in Java SE Applications
- Unsupported Methods for File-Based Policy Stores

For other OPSS services for Java SE applications, see Section 25.2, "Implementing Security Services in Java SE Applications."

18.1 Configuring the Security Store in Java SE Applications

The configuration of policy and credential stores in Java SE applications is explained in the following sections:

- Configuring File-Based Policy and Credential Stores
- Configuring LDAP-Based Policy and Credential Stores
- Configuring DB-Based OPSS Security Stores

System properties should be set, as appropriate, for authorization to work in Java SE applications. For a complete list of properties, see Section F.1, "OPSS System Properties."

A Java SE application can use file-, LDAP-, or DB-based store providers; these services are configured in the application file `jps-config-jse.xml`.

18.1.1 Configuring File-Based Policy and Credential Stores

A file-based policy store is specified in the file `system-jazn-data.xml`; a file-based credential store is specified in the file `cwallet.sso` (this wallet file should not be confused with the bootstrap file, also named `cwallet.sso`, which contains the credentials to access LDAP stores, when the application security is LDAP-based).

For details about wallets, see Section 24.6.3, "Using a Wallet-Based Credential Store." For details about modifying or adding bootstrap credentials, see modifyBootStrapCredential and addBootStrapCredential.

The following fragments illustrate the configuration of file-based policy and credential stores, and the jpsContext that reference them:

```
<serviceProviders>
  <serviceProvider type="CREDENTIAL_STORE" name="credstoressp"
  class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider">
    <description>SecretStore-based CSF Provider</description>
  </serviceProvider>
  <serviceProvider type="POLICY_STORE" name="policystore.xml.provider"
class="oracle.security.jps.internal.policystore.xml.XmlPolicyStoreProvider">
  <description>XML-based PolicyStore Provider</description>
  </serviceProvider>
</serviceProviders>

<serviceInstances>
  <serviceInstance name="credstore" provider="credstoressp" location="./">
    <description>File-based Credential Store Service Instance</description>
  </serviceInstance>

  <serviceInstance name="policystore.xml" provider="policystore.xml.provider"
location="./system-jazn-data.xml">
   <description>File-based Policy Store Service Instance</description>
   <property name="oracle.security.jps.policy.principal.cache.key" value="false"/>
  </serviceInstance>
</serviceInstances>

<jpsContexts default="TestJSE">
  <jpsContext name="TestJSE">
    <serviceInstanceRef ref="credstore"/>
    <serviceInstanceRef ref="policystore.xml"/>
    ...
  </jpsContext>
  ...
</jpsContexts>
```

Note the required setting of the property oracle.security.jps.policy.principal.cache.key to false in the policy store instance.

18.1.2 Configuring LDAP-Based Policy and Credential Stores

This section assumes that an LDAP-based store has been set to be used as the policy and credential stores; for details about setting up nodes in an Oracle Internet Directory, see section Section 9.2.1, "Prerequisites to Using an LDAP-Based Security Store."

The following fragments illustrate the configurations of providers and instances for LDAP-based policy and credential stores for a Java SE application:

```
<serviceProviders
  <serviceProvider type="POLICY_STORE" mame="ldap.policystore.provider"
 class=oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"/>

  <serviceProvider type="CREDENTIAL_STORE" mame="ldap.credential.provider"
 class=oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider"/>
</serviceProviders>

<serviceInstances>
 <serviceInstance provider="ldap.policystore.provider" name="policystore.ldap">
  <property value="OID" name="policystore.type"/>
  <property value="bootstrap" name="bootstrap.security.principal.key"/>
  <property value="cn=PS1domainRC3" name="oracle.security.jps.farm.name"/>
  <property value="cn=myTestNode" name="oracle.security.jps.ldap.root.name"/>
  <property value="ldap://myComp.com:1234" name="ldap.url"/>
 </serviceInstance>

 <serviceInstance provider="ldap.credential.provider" name="credstore.ldap">
  <property value="bootstrap" name="bootstrap.security.principal.key"/>
  <property value="cn=PS1domainRC3" name="oracle.security.jps.farm.name"/>
  <property value="cn=myTestNode" name="oracle.security.jps.ldap.root.name"/>
  <property value="ldap://myComp.com:1234" name="ldap.url"/>
 </serviceInstance>
</serviceInstances>
```

The following fragment illustrates the configuration of the bootstrap credentials file (`cwallet.sso`), which allows the program access to the LDAP server:

```
<serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.cred">
  <property value="./bootstrap" name="location"/>
</serviceInstance>
```

The following fragment illustrates the configuration of the necessary jpsContexts that reference the instances above:

```
<jpsContexts default="TestJSE">
  <jpsContext name="TestJSE">
    <serviceInstanceRef ref="policystore.ldap"/>
    <serviceInstanceRef ref="credstore.ldap"/>
  </jpsContext>
  <jpsContext name="bootstrap_credstore_context">
    <serviceInstanceRef ref="bootstrap.cred"/>
  </jpsContext>
</jpsContexts>
```

The following code fragment illustrates how to obtain programmatically a reference to the LDAP-based policy store configured above, and it assumes that the system property `oracle.security.jps.config` has been set to the location of the file `jps-config-jse.xml`:

```
String contextName="TestJSE"; ...
public static PolicyStore getPolicyStore(String contextName) {
      try-block
            JpsContextFactory ctxFact;
            ctxFact = JpsContextFactory.getContextFactory();
            JpsContext ctx = ctxFact.getContext(contextName);
            return ctx.getServiceInstance(PolicyStore.class);
      catch-block
...
```

18.1.3 Configuring DB-Based OPSS Security Stores

This section assumes that a DB-based store has been set to be used as the OPSS security store. For details about setting up nodes in a DB, see section Section 9.3.1, "Prerequisites to Using a DB-Based Security Store."

Note the following important points regarding the sample configuration below:

- The value of the configuration property `jdbc.url` should be identical to the name of the JDBC data source entered when the data source was created.
- The values of the bootstrap credentials (map and key) must match those passed to the WLST command `addBootStrapCredential` when the bootstrap credential was created.

The following fragment illustrates configuration of DB-based policy, credential, and key stores in the file `jps-config-jse.xml`:

```
<jpsConfig  …>
  <propertySets>
   <propertySet name="props.db.1">
```

```
    <property value="cn=myDomain" name="oracle.security.jps.farm.name"/>
    <property value="DB_ORACLE" name="server.type"/>
    <property value="cn=myRoot" name="oracle.security.jps.ldap.root.name"/>
    <property name="jdbc.url" value="jdbc:oracle:thin:@myhost.com:1521/srv_name"/>
    <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver"/>
    <property name="bootstrap.security.principal.key" value="myKeyName" />
    <property name="bootstrap.security.principal.map" value="myMapName" />
   </propertySet>
  </propertySets>
  <serviceProviders>
    <serviceProvider
class="oracle.security.jps.internal.policystore.OPSSPolicyStoreProvider"
           type="POLICY_STORE" name="policy.rdbms">
      <description>DBMS based PolicyStore</description>
    </serviceProvider>

    <serviceProvider
class="oracle.security.jps.internal.credstore.rdbms.DbmsCredentialStoreProvider"
               type="CREDENTIAL_STORE" name="db.credentialstore.provider" >

  <serviceProvider class="oracle.security.jps.internal.keystore.KeyStoreProvider"
               type="KEY_STORE" name="keystore.provider" >
    <property name="provider.property.name" value="owsm"/>
  </serviceProvider>
 </serviceProviders>

 <serviceInstances>
   <serviceInstance name="policystore.rdbms" provider="db.policystore.provider">
    <propertySetRef ref = "props.db.1"/>
    <property name="policystore.type"  value="DB_ORACLE"/>
   </serviceInstance>

   <serviceInstance name="credstore.rdbms" provider="db.credstore.provider">
    <propertySetRef ref = "props.db.1"/>
   </serviceInstance>

   <serviceInstance name="keystore.rdbms" provider="db.keystore.provider">
    <propertySetRef ref = "props.db.1"/>
    <property name="keystore.provider.type"  value="db"/>
   </serviceInstance>
  </serviceInstances>

  <jpsContexts default="default">
    <jpsContext name="default">
      <serviceInstanceRef ref="policystore.rdbms"/>
      <serviceInstanceRef ref="credstore.rdbms"/>
      <serviceInstanceRef ref="keystore.rdbms"/>
    </jpsContext>
  </jpsContexts>
</jpsConfig>
```

18.2 Unsupported Methods for File-Based Policy Stores

This release does not support, for file-based policy stores, methods involving the following features:

- Complex queries
- Cascading deletions

Complex queries relates to any method that takes a query. When the policy store is file-based, the query must be simple; if such a method is passed a complex query and the policy store is file-based, the method will throw an exception.

A simple query is a query with just one search criterion; a complex query is a query with two or more search criteria; each call to `addQuery` adds a criterion to the query.

The following code fragment that illustrates the building of a simple query that returns of all permissions with a display name matching the string `MyDisplayName`:

```
PermissionSetSearchQuery query = new PermissionSetSearchQuery();
query.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.DISPLAY_NAME,
               false,
               ComparatorType.EQUALITY,
               "MyDisplayName",
               BaseSearchQuery.MATCHER.EXACT);
getPermissionSets(query);
```

The following example illustrates the building of a complex query that returns all permission sets with a given resource type and a given resource instance name:

```
PermissionSetSearchQuery query = new PermissionSetSearchQuery();
query.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.RESOURCE_TYPE,
               false,
               ComparatorType.EQUALITY,
               "MyResourceType",
               BaseSearchQuery.MATCHER.EXACT);

query.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.RESOURCE_NAME,
               false,
               ComparatorType.EQUALITY,
               "MyResourceInstanceName",
               BaseSearchQuery.MATCHER.EXACT);

query.setANDMatch();
getPermissionSets(query);
```

Cascading deletions relates to any method that includes the Boolean argument `cascadeDelete`. The only value allowed for this argument in case the policy store is file-based is FALSE. Here is an example of such a method in the interface `ResourceTypeManager`:

```
void deleteResourceType(EntryReference rtRef, boolean cascadeDelete)
                 throws PolicyObjectNotFoundException,
                        PolicyStoreOperationNotAllowedException,
                        PolicyStoreException
```

19 Developing with the Credential Store Framework

This chapter describes how to work with the Credential Store Framework (CSF) APIs in the following sections:

19.1 About the Credential Store Framework API

A credential store is used for secure storage of credentials. The credential store framework (CSF) API is used to access and perform operations on the credential store.

The Credential Store Framework:

- enables you to manage credentials securely
- provides an API for storage, retrieval, and maintenance of credentials in different back-end repositories
- supports file-based (Oracle wallet) and LDAP-based credential management

Critical (create, update, delete) functions provided by the CSF API include:

- verifying if a credential map, or a credential with a given key, exists in the store
- returning credentials associated with `<mapname, key>`
- assigning credentials to `<mapname, key>`
- deleting credentials associated with a given map name, or a given map name and key
- resetting credentials for a specified `<mapname, key>`

Operations on `CredentialStore` are secured by `CredentialAccessPermission`, which implements the fine-grained access control model utilized by CSF.

See Also:

- Chapter 11, "Managing the Credential Store"

19.2 Overview of Application Development with CSF

Knowledge of the following areas is helpful in getting your applications to work with the credential store framework:

- Determining appropriate map names and key names to use. This is critical in an environment with multiple applications storing credentials in the common credential store.
- Provisioning Java security policies.

  Policy permissions are set in the policy store, which can be file-based (`system-jazn-data.xml`) or LDAP-based. Setting appropriate permissions to enable application usage without compromising the security of your data requires careful consideration of permission settings.

  See Also: Section 10.2, "Managing the Policy Store".

- How to define the credential store instance in `jps-config.xml`.

  You will need to define the service instance in `jps-config.xml` only if manually crafting the configuration file.

  Note: The file-based provider is already configured by default, and can be changed to an LDAP-based provider. See Section 9.6, "Migrating the OPSS Security Store".

- Steps to set up an environment.

  These steps differ according to the type of application: Java SE applications or Java EE applications.

Subsequent sections in this chapter provide detailed settings for each type of application.

19.3 Setting the Java Security Policy Permissions

The Oracle Platform Security Services policy provider is set when the server is started. When the provider is file-based, the policy data is stored in `system-jazn-data.xml`.

CSF supports securing credentials:

- at the map level, or
- with finer granularity for specific `<mapname, key>`

Notes:

- To properly access the CSF APIs, you need to grant Java permissions in the policy store.
- The code invoking CSF APIs needs code source permission. The permissions are typically for specific code jars and not for the complete application.

19.3.1 Guidelines for Granting Permissions

The Credential Store Framework relies on Java permissions to grant permissions to credential store objects.

It is highly recommended that only the requisite permissions be granted, and no more.

WARNING: It is risky and inadvisable to grant unnecessary permissions, particularly permissions to all maps and/or keys.

19.3.2 Permissions Grant Example 1

Note: In the examples, the application jar file name is `AppName.jar`.

The `CredentialStore` maintains mappings between map names and credential maps. Each map name is mapped to a `CredentialMap`, which is a secure map of keys to `Credential` objects.

This example grants permissions for a specific map name and a specific key name of that map.

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <!-- This is the location of the jar -->
            <!-- as loaded with the run-time -->
            <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
            <permission>
               <class>oracle.security.jps.service.credstore.
                      CredentialAccessPermission</class>
               <name>context=SYSTEM,mapName=myMap,keyName=myKey</name>
              <!-- All actions are granted -->
              <actions>*</actions>
            </permission>
        </permissions>
    </grant>
</jazn-policy>
```

where:

- `MapName` is the name of the map (typically the name of the application) for which you want to grant these permissions (read, write, update, and delete permissions denoted by the wildcarded actions).
- `KeyName` is the key name in use.

19.3.3 Permissions Grant Example 2

In this example permissions are granted for a specific map name and all its key names.

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
           <permission>
              <class>oracle.security.jps.service.credstore.
                     CredentialAccessPermission</class>
              <name>context=SYSTEM,mapName=myMap,keyName=*</name>
              <!-- Certain actions are explicitly specified -->
              <!-- Compare to wild-card grant in previous example -->
              <actions>read,write,update,delete</actions>
        </permission>
        </permissions>
    </grant>
</jazn-policy>
```

19.4 Guidelines for the Map Name

When the domain-level credential store is used, name conflicts can arise with the various map names in the store for different applications. To avoid this, each application must have a unique map name in the store.

To achieve this, it is recommended that the map name you use uniquely identify the application.

Within a given map name, an application can store multiple credentials each of which is identifiable by a key. The map name and the key together constitute a primary key within a given credential store.

If there is a requirement that an application use more than one map name, then uniqueness continues to be maintained.

For example, consider three applications:

- a Repository Creation Utility (RCU) based application,
- a Oracle WebCenter application, and
- a Fusion Middleware Control application

For RCU, a map name of RCU is chosen and the keys for three credentials are (say) Key1, Key2, and Key3:

Note: The map names and key names used here are arbitrary and chosen for illustration only. Your application can use altogether different map names and/or keynames.

```
MapName -> RCU, Key -> Key1 and Credential -> PasswordCredential1
MapName -> RCU, Key -> Key2 and Credential -> PasswordCredential2
MapName -> RCU, Key -> Key3 and Credential -> GenericCredential1
```

For Oracle WebCenter, the map name is Web and the key for a single credential is Key1:

```
MapName -> Web, Key -> Key1 and Credential -> PasswordCredential3
```

For Fusion Middleware Control, the map name is denoted by EM and the keys for two credentials are Key1 and Key2 respectively:

```
MapName -> EM, Key -> Key1 and Credential -> PasswordCredential4
MapName -> EM, Key -> Key2 and Credential -> GenericCredential2
```

Note that the map name and key name are just two arbitrary strings and can have any valid string values in practice. However, implementing this way makes map names easier to manage.

19.5 Configuring the Credential Store

The administrator needs to define the credential store instance in a configuration file which contains information about the location of the credential store and the provider classes. Configuration files are located in:

```
$DOMAIN_HOME/config/fmwconfig
```

and are named as follows:

- `jps-config.xml` for Oracle WebLogic Server
- `jps-config-jse.xml` for Java SE

For details, see Chapter 11, "Managing the Credential Store".

See Also: Section 25.1, "Using OPSS in Java SE Applications".

19.6 Using the CSF API

You can use the credential store framework within Oracle WebLogic Server or in a standalone environment.

- Using the CSF API in Java SE Applications
- Using the CSF API in Java EE Applications

19.6.1 Using the CSF API in Java SE Applications

See Also: Section 25.1, "Using OPSS in Java SE Applications".

To use the CSF API in Java SE applications, proceed as follows:

1. Set up the classpath. Ensure that the `jps-manifest.jar` file is in your classpath. For details, see Required JAR in Classpath in Section 1.5.3, "Scenario 3: Securing a Java SE Application".
2. Set up the policy; to provide access to the CSF APIs, you need to configure the access permissions in the reference policy store. For examples, see Section 19.3, "Setting the Java Security Policy Permissions".
3. Set JVM command-line options. See details below.
4. Run the application.

Command-line options include:

`-Doracle.security.jps.config`
specifies the full path to the configuration file `jps-config.xml`, typically located in the directory *domain_home*`/config/fmwconfig`.

`-Djava.security.policy`
specifies the location of the OPSS/Oracle WebLogic Server policy file `weblogic.policy`, typically located in the directory *wl_home*`/server/lib`.

`-Dcommon.components.home`
specifies the location of the oracle_common directory under middleware home.

`-Dopss.version`
specifies the version used in the environment; its value is 12.1.3.

`-Djava.security.debug=all`
is helpful for debugging purposes

19.6.2 Using the CSF API in Java EE Applications

To use the CSF API in Java EE applications, proceed as follows:

1. The credential store service provider section of the `jps-config.xml` file is configured out-of-the-box in the following directory:

   ```
   $DOMAIN_HOME/config/fmwconfig
   ```

   If needed, reassociate to an LDAP credential store.
2. Set up the policy; to provide access to the CSF APIs, you need to configure the access permissions in the reference policy store. For examples, see Section 19.3, "Setting the Java Security Policy Permissions".
3. Start Oracle WebLogic Server.
4. Deploy and test the application.

19.7 Examples

This section presents several code samples illustrating the use of the credential store framework APIs. Each code sample illustrates how credential store operations are affected by permissions, and they include sample configuration files.

See Also: Section 25.1, "Using OPSS in Java SE Applications".

The section includes the following samples:

- Common Code for CSF Operations
- Example 1: Java SE Application with Wallet Store

- Example 2: Java EE Application with Wallet Store
- Example 3: Java EE Application with OID LDAP Store
- Example 4: Java EE Application with Oracle DB Store

19.7.1 Common Code for CSF Operations

The following "utility" program performs the CSF API operations, and it is called by code in the examples that follow this section.

```
package demo.util;

import java.security.AccessController;
import java.security.PrivilegedAction;

import oracle.security.jps.JpsException;
import oracle.security.jps.service.credstore.Credential;
import oracle.security.jps.service.credstore.CredentialAlreadyExistsException;
import oracle.security.jps.service.credstore.CredentialFactory;
import oracle.security.jps.service.credstore.CredentialStore;
import oracle.security.jps.service.credstore.PasswordCredential;

public class CsfUtil {
    final CredentialStore store;
    public CsfUtil(CredentialStore store) {
        super();
        this.store = store;
    }

    private void doOperation() {
        try {
            PasswordCredential pc = null;
            try {
                // this call requires read privilege
                pc = (PasswordCredential)store.getCredential("pc_map", "pc_key");
                if (pc == null) {
                    // key not found, create one
                    pc = CredentialFactory.newPasswordCredential("jdoe",
                          "password".toCharArray());
                    // this call requires write privilege
                    store.setCredential("pc_map", "pc_key", pc);
                    System.out.print("Created ");
                }
                else {
                    if (pc instanceof PasswordCredential){
                      System.out.print("Found ");
                    } else {
                      System.out.println("Unexpected credential type found");
                }

                System.out.println("password credential: Name=" + pc.getName() +
                                   ",Password=" +
                                   new String(pc.getPassword()));

            } catch (CredentialAlreadyExistsException e) {
                // ignore since credential already exists.
                System.out.println("Credential already exists for
                <pc_map, pc_key>: " + pc.getName() + ":" +
                new String(pc.getPassword()));
            }
            try {
                // permission corresponding to
                // "context=SYSTEM,mapName=gc_map,keyName=gc_key"
                byte[] secret =
                    new byte[] { 0x7e, 0x7f, 0x3d, 0x4f, 0x10,
                                 0x20, 0x30 };
                Credential gc =
                    CredentialFactory.newGenericCredential(secret);
                store.setCredential("gc_map", "gc_key", gc);
                System.out.println("Created generic credential");
            } catch (CredentialAlreadyExistsException e) {
                // ignore since credential already exists.
                System.out.println("Generic credential already exists
                  for <gc_map,gc_key>");
            }

            try {
                //no permission for pc_map2 & pc_key2 to perform
                //operation on store
                Credential pc2 =
                    CredentialFactory.newPasswordCredential("pc_jode2",
                    "pc_password".toCharArray());
                store.setCredential("pc_map2", "pc_key2", pc2);

            } catch (Exception expected) {
                //CredentialAccess Exception expected here. Not enough permission
                System.out.println("This is expected :" +
                                   expected.getLocalizedMessage());
            }

        } catch (JpsException e) {
            e.printStackTrace();
        }

    }

    /*
     * This method performs a non-privileged operation. Either all code
     * in the call stack must have CredentialAccessPermission
     * OR
     * the caller must have the CredentialAccessPermission only and
     * invoke this operation in doPrivileged block
     */
    public void doCredOperation() {
        doOperation();
    }

    /*
     * Since this method performs a privileged operation, only current class or
     * jar containing this class needs CredentialAccessPermission
     */
    public void doPrivilegedCredOperation() {
        AccessController.doPrivileged(new PrivilegedAction<String>() {
                public String run() {
                    doOperation();
                    return "done";
                }
            });
    }
```

```
}
```

19.7.2 Example 1: Java SE Application with Wallet Store

This example shows a sample Java SE application using wallet credentials, that is, a file-based provider.

> **See Also:** Section 25.1, "Using OPSS in Java SE Applications," for details about securing a Java SE applications with OPSS.

The example illustrates:

- how the permissions are set in an xml-based policy store (jazn-data-xml)
- how the configuration file is set up
- the Java SE code

jazn-data.xml File

For illustration, the example uses an xml-based policy store file which has the appropriate permissions needed to access the given credential from the store. The file defines the permissions for different combinations of map name (alias) and key. Other combinations, or attempts to access the store beyond the permissions defined here, will be disallowed.

> **Note:** The default policy store to which this grant is added is `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml`.

Here the system property `projectsrc.home` is set to point to the directory containing the Java SE application, and `clientApp.jar` is the application jar file which is present in sub-directory `dist`.

The corresponding policy grant looks like this:

```
<grant>
   <grantee>
      <codesource>
         <url>file:${projectsrc.home}/dist/clientApp.jar</url>
      </codesource>
   </grantee>
   <permissions>
      <permission>
         <class>oracle.security.jps.service.credstore.CredentialAccessPermission
         </class>
         <name>context=SYSTEM,mapName=pc_map,keyName=*</name>
         <actions>read,write</actions>
      </permission>
      <permission>
         <class>oracle.security.jps.service.credstore.CredentialAccessPermission
         </class>
         <name>context=SYSTEM,mapName=gc_map,keyName=gc_key</name>
         <actions>write</actions>
      </permission>
   </permissions>
</grant>
```

Note that no permission has been granted to `mapName=pc_map2,keyName=pc_key2`, hence the `setCredential` call for this map and key combination in Section 19.7.1, "Common Code for CSF Operations" is expected to fail.

jps-config-jse.xml File

Note: For the complete configuration file see the default file shipped with the distribution at `$DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml`.

The location property of the credential store service shows the directory containing the wallet file:

```
<jpsConfig>
 ...
    <serviceInstances>
        <serviceInstance name="credstore_file_instance"
                         provider="credstore_file_provider">
            <property name="location" value="store" />
        </serviceInstance>
    </serviceInstances>
 ...
</jpsConfig>
```

Note: The default value of location is "./", that is, the current directory relative to the location of `jps-config-jse.xml`. To use a different path, be sure to specify the full path.

The wallet name is always `cwallet.sso` which is the default file-based Oracle wallet.

Java Code

Here is the Java SE code that calls the utility program.

```
package demo;

import java.io.ByteArrayInputStream;

import java.security.AccessController;
import java.security.PrivilegedAction;

import oracle.security.jps.JpsContext;
import oracle.security.jps.JpsStartup;
import oracle.security.jps.JpsContextFactory;
import oracle.security.jps.JpsException;
import oracle.security.jps.jaas.JavaPolicy;
import oracle.security.jps.service.credstore.Credential;
import oracle.security.jps.service.credstore.CredentialAlreadyExistsException;
import oracle.security.jps.service.credstore.CredentialFactory;
import oracle.security.jps.service.credstore.CredentialStore;
import oracle.security.jps.service.credstore.PasswordCredential;
import oracle.security.jps.service.policystore.PolicyStore;
import oracle.security.jps.service.policystore.PolicyStoreException;

import demo.util.CsfUtil;
```

```
public class CsfApp {

    public CsfApp() {
        super();
    }

    public static void main(String[] a) {
        // perform operation as privileged code
        JpsContextFactory ctxFactory;
        try {
              new JpsStartup().start();
            ctxFactory = JpsContextFactory.getContextFactory();
            JpsContext ctx = ctxFactory.getContext();

            CredentialStore store =
                ctx.getServiceInstance(CredentialStore.class);
            CsfUtil csf = new CsfUtil(store);
            // #1 - this call is in a doPrivileged block
            // #1 - this should succeed.
            csf.doPrivilegedCredOperation();

            // #2 - this will also pass since granted all application
            // code necessary permission
            // NOTE: Since this call is not in a doPrivileged block,
            // this call would have failed if CredentialAccessPermission
            // wasn't granted to this class.
            /*
            csf.doCredOperation();
            */
        } catch (JpsException e) {
            e.printStackTrace();
        }

    }
}
```

Notes:

- It is not necessary to replace the JDK-wide policy object. Since the example grant shown conforms to the OPSS XML policy store, it is reasonable to set the policy provider to the OPSS provider.
- In a Java EE environment for a supported application server, the OPSS policy provider will have been initialized.

19.7.3 Example 2: Java EE Application with Wallet Store

This example shows a sample Java EE application using wallet credentials. A simple servlet calls the CSF API.

The jazn-data.xml File

The `jazn-data.xml` file for this example defines the appropriate permissions needed to access the given credential from the store. The file defines both the codesource permissions and the permissions for different combinations of map name (alias) and key. Other combinations, or attempts to access the store beyond the permissions defined here, will be disallowed.

A fragment of the policy file showing the corresponding policy grant looks like this:

```
<grant>
   <grantee>
      <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
      </codesource>
   </grantee>
   <permissions>
      <permission>
         <class>oracle.security.jps.service.credstore.CredentialAccessPermission
         </class>
         <name>context=SYSTEM,mapName=pc_map,keyName=*</name>
         <actions>read,write</actions>
      </permission>
      <permission>
         <class>oracle.security.jps.service.credstore.CredentialAccessPermission
         </class>
         <name>context=SYSTEM,mapName=gc_map,keyName=gc_key</name>
         <actions>write</actions>
      </permission>
   </permissions>
</grant>
```

Note that the first map and key permissions enable both read and write operations; the second enable write operations but not reads.

jps-config.xml File

A portion of the default configuration file `jps-config.xml` showing the credential store configuration is as follows:

```
<jpsConfig>
    <serviceProviders>
        <serviceProvider type="CREDENTIAL_STORE" name="credstoressp"
    class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider">
            <description>SecretStore-based CSF provider</description>
        </serviceProvider>
    </serviceProviders>

    <serviceInstances>
        <serviceInstance name="credstore" provider="credstoressp">
            <property name="location" value="./" />
        </serviceInstance>
    </serviceInstances>

    <jpsContexts default="default">
        <jpsContext name="default">
        ...
            <serviceInstanceRef ref="credstore"/>
        ...
        </jpsContext>
    </jpsContexts>
</jpsConfig>
```

The `location` property specifies the wallet location; this specification is essentially the same as in Example 1, except that in this example the wallet is located inside the configuration directory. The wallet name is always `cwallet.sso`.

Java Code

```
package demo;

import demo.util.CsfUtil;

import java.io.IOException;
import java.io.PrintWriter;

import java.net.URL;

import java.util.Date;

import javax.servlet.*;
import javax.servlet.http.*;

import oracle.security.jps.JpsException;
import oracle.security.jps.service.JpsServiceLocator;
import oracle.security.jps.service.credstore.CredentialStore;

public class CsfDemoServlet extends HttpServlet {
    private static final String CONTENT_TYPE = "text/html; charset=windows-1252";

    public void init(ServletConfig config) throws ServletException {
        super.init(config);
    }

    public void doGet(HttpServletRequest request,
                      HttpServletResponse response) throws ServletException,
                                                           IOException {
        response.setContentType(CONTENT_TYPE);
        PrintWriter out = response.getWriter();
        //ServletOutputStream out = response.getOutputStream();
        try {
            response.setContentType("text/html");
            out.println("<html><body bgcolor=\"#FFFFFF\">");
            out.println("<b>Current Time: </b>" + new Date().toString() +
                        "<br><br>");

            //This is to get hold of app-level CSF service store
            //Outside app context, this call returns domain-level CSF store
            //This call also works in Java SE env
            final CredentialStore store =
              JpsServiceLocator.getServiceLocator().lookup(CredentialStore.class);
            CsfUtil csf = new CsfUtil(store);

            csf.doPrivilegedCredOperation();
            out.println("Credential operations completed using privileged code.");
        } catch (JpsException e) {
            e.printStackTrace(out);
        }
    }
}
```

The credential create operation is conducted using privileged code.

Note About Java SE Environment

In the Java SE environment, the following calls are equivalent:

```
CredentialStore store =
JpsServiceLocator.getServiceLocator().lookup(CredentialStore.class);
```

and:

```
CredentialStore store =
JpsContextFactory.getContextFactory().getContext().getServiceInstance(CredentialSt
ore.class);
```

The latter call is shown in Section 19.7.2, "Example 1: Java SE Application with Wallet Store".

19.7.4 Example 3: Java EE Application with OID LDAP Store

This example uses the same Java EE application used in Example 2: Java EE Application with Wallet Store; the only difference is that the credential store in the following example is LDAP-based and instead of file-based (wallet).

The domain's `jps-config.xml` must configure the following properties:

- The root name:

  ```
  <property name="oracle.security.jps.ldap.root.name"
  value="cn=OracleJpsContainer"/>
  ```

- The farm name:

  ```
  <property name="oracle.security.jps.farm.name" value="cn=OracleFarmContainer"
  />
  ```

- The LDAP URL:

  ```
  <property name="ldap.url" value="ldap://mynode.us.mycorp.com:1234"/>
  ```

- The principal key, which (in conjunction with the above required properties) is used to locate the credentials to access the LDAP store:

  ```
  <property name="bootstrap.security.principal.key" value="bootstrap"/>
  ```

Here is an example of a `jps-config.xml` file specifying the required properties for a OID LDAP-based store:

```
<jpsConfig>
    <serviceProviders>
        <serviceProvider name="ldap.credentialstore.provider"
 class="oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider">
            <description>Prototype LDAP-based CSF provider</description>
        </serviceProvider>
    </serviceProviders>

    <serviceInstances>
       <serviceInstance provider="ldap.credentialstore.provider"
          name="credstore.ldap">
          <property value="bootstrap"
             name="bootstrap.security.principal.key"/>
          <property value="cn=wls-jrfServer"
             name="oracle.security.jps.farm.name"/>
          <property value="cn=jpsTestNode"
             name="oracle.security.jps.ldap.root.name"/>
          <property value="ldap://mynode.us.mycorp.com:1234"
             name="ldap.url"/>
       </serviceInstance>
    </serviceInstances>
```

```
    <jpsContexts default="appdefault">
        <jpsContext name="appdefault">
            <serviceInstanceRef ref="credstore.ldap"/>
        </jpsContext>
    </jpsContexts>
</jpsConfig>
```

19.7.5 Example 4: Java EE Application with Oracle DB Store

This example uses the same Java EE application as in Example 2: Java EE Application with Wallet Store; the only difference is that the credential store in the following example is DB-based instead of file-based (wallet).

The following properties must be configured in the domain's `jps-config.xml` file:

- The root name:

  ```
  <property name="oracle.security.jps.ldap.root.name"
   value="cn=OracleJpsContainer"/>
  ```

- The farm name:

  ```
  <property name="oracle.security.jps.farm.name"
   value="cn=OracleFarmContainer"/>
  ```

- The JDBC URL:

  ```
  <property name="jdbc.url" value="jdbc:oracle:thin:@<host>:<port>:<service>"/>
  ```

- The JDBC driver:

  ```
  <property name="jdbc.driver" value="oracle.jdbc.OracleDriver"/>
  ```

- The datasource JNDI name:

  ```
  <property name="datasource.jndi.name" value="jdbc/OpssDS"/>
  ```

- The principal key, which (in conjuntion with the above required properties) is used to locate the credentials to access the DB store:

  ```
  <property value="bootstrap" name="bootstrap.security.principal.key"/>
  ```

Here is an example of a `jps-config.xml` file specifying the required properties for a DB-based store:

```
<jpsConfig>
 <serviceProviders>
  <serviceProvider type="CREDENTIAL_STORE" name="db.credentialstore.provider"
class="oracle.security.jps.internal.credstore.rdbms.DbmsCredentialStoreProvider"/>
   <description>DB-based CSF provider</description>
   </serviceProvider>
 </serviceProviders>

 <serviceInstances>
  <serviceInstance provider="db.credentialstore.provider"
   name="credstore.db">
   <property value="bootstrap"
    name="bootstrap.security.principal.key"/>
   <property value="cn=wls-jrfServer"
    name="oracle.security.jps.farm.name"/>
   <property value="cn=jpsTestNode"
    name="oracle.security.jps.ldap.root.name"/>
```

```
    <property name="jdbc.url" value="jdbc:oracle:thin:@localhost:5521:ldapoid"/>
    <property name="jdbc.driver" value="oracle.jdbc.OracleDriver"/>
    <property name="datasource.jndi.name" value="jdbc/OpssDS"/>
  </serviceInstance>
 </serviceInstances>

 <jpsContexts default="appdefault">
   <jpsContext name="appdefault">
     <serviceInstanceRef ref="credstore.db"/>
   </jpsContext>
 </jpsContexts>
</jpsConfig>
```

19.8 Best Practices

In a clustered environment and when the store is file-based, use the Credential Store Mbean API over the Credential Store Framework API to create, retrieve, update, and delete credentials for an application.

If you are simply reading credentials, however, either API can be used.

20 Developing with the User and Role API

This chapter explains how to access identity information with the User and Role API.

This chapter includes the following sections:

Note: The User and Role API is deprecated and may be withdrawn in a future release. Your new applications should be developed on the Identity Governance Framework. Plan to migrate existing applications to the Identity Governance Framework in a future release.

For details, see the *Developer's Guide for Identity Governance Framework*.

See Also: Section 25.1, "Using OPSS in Java SE Applications".

20.1 Introduction to the User and Role 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 repository could be an LDAP directory server such as Oracle Internet Directory, Active Directory (from Microsoft), or Oracle Directory Server Enterprise Edition, 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.

Note: These APIs are not meant for authentication or authorization functions, but for maintaining identity information.

You can use a basic usage model (without container integration) or a usage model with container integration that allows your code to be portable.

When the application is intended to run in the context of an Oracle WebLogic Server container, the principal class should be cast to `weblogic.security.principal.WLSUserImpl`.

Note: The following are required to invoke the User and Role API in context of an Oracle WebLogic Server container:

- The identity store is LDAP-based
- The domain administration server is up and running

A Note about Using the User and Role API

As a general rule of thumb, authentication should only be performed by authentication providers, not through the User and Role API.

Additionally, it is recommended that authentication providers be configured with the connect DN of a user that does not have write privileges.

20.1.1 Understanding User and Role API and the Oracle WebLogic Server Authenticators

The User and Role API is automatically configured to use the first Oracle WebLogic Server authenticator and does not require any special configuration.

Note, however, that configuration is required if the User and Role API is going against other authenticators.

The API can access data only from the first LDAP authenticator listed in an Oracle WebLogic Server domain. When more than one authenticator is present, the precedence is determined by their control flag priority. If both have the same priority, the first one is picked. Any LDAP authenticators below the first one on the list are not accessed.

About Concurrent Use of WebLogic APIs

Your application should not try to use both the User and Role API and the WebLogic LDAPAuthenticator API (such as EmbeddedLDAPAuthenticator, OracleInternetDirectoryAuthenticator, OracleVirturalDirectoryAuthenticator) to work on entries in the same LDAP server concurrently. To understand why, consider two LDAP clients, both with caching enabled, that access the same LDAP server; one is deleting entries, and the other tries to use the deleted entries.

The conflict caused by the two clients cannot be resolved unless caching capability is disabled, and the LDAP operations are coordinated among the clients.

20.2 Listing Roles and Classes

Table 20–1 lists the classes and interfaces of the User and Role API.

Table 20–1 Classes and Interfaces in the User and Role API

Name	Type	Description
AuthenticationException	Class	This exception is thrown when an authentication error occurs while accessing the identity store. An authentication error can happen, for example, when the credentials supplied by the user program is invalid or otherwise fails to authenticate the user to the identity store.
AuthenticationWarningException	Class	This class extends IMException (see below).
ComplexSearchFilter	Interface	A complex search filter represents a complex logical expression that can be used to filter results from underlying identity repository. Complex search filter combines multiple SearchFilter instances together with a single logical operator (AND/OR). Each of these component SearchFilter can itself be a complex filter, enabling you to form a complex nested search filter. See the Javadoc (Section 20.9, "User and Role API Reference") for an example of creating a complex search filter.
ConfigurationException	Class	This exception is thrown when there is a configuration problem. This can arise when configuration information required to access the service provider is malformed or missing.
Identity	Interface	This interface represents a basic identity in the identity repository.
IdentityStore	Interface	IdentityStore represents a handle to actual identity repository. This handle can be used to search, create, drop, and modify identities in the repository.
IdentityStoreFactory	Interface	IdentityStoreFactory is a programmatic representation of underlying identity repository. Actual handle to the identity repository can be obtained by calling `getIdentityStoreInstance(Hashtable)` on this object.
IdentityStoreFactoryBuilder	Class	This class builds the identity store factory.
IMException	Class	This exception is the superclass of all the exceptions thrown by ADF identity management APIs. The nature of failure is described by the name of the subclass. See the Javadoc (Section 20.9, "User and Role API Reference") for a list of the direct known subclasses.
ModProperty	Class	This class represents the modification of a property object. ModProperty is called with property name, modified value(s) and type of modification. Modification type can be one of `ADD`, `REMOVE`, or `REPLACE`.
NoPermissionException	Class	This exception is thrown when attempting to perform an operation for which the API caller has no permission. The access control/permission model is dictated by the underlying identity store.
ObjectExistsException	Class	This exception is thrown when an identity with given name is already present in the underlying identity store. For example this exception is thrown when create user API call tries to create a user with the name of an existing user.
ObjectNotFoundException	Class	This exception is thrown when a specified identity does not exist in the identity store.

Table 20–1 (Cont.) Classes and Interfaces in the User and Role API

Name	Type	Description
OperationFailureException	Class	This exception is thrown when an operation fails during execution in the underlying identity store.
OperationNotSupportedException	Class	This exception is thrown by an service provider if it does not support an operation. For example this can be thrown by the service provider, in IdentityStore.getUserManager() call, if it does not provide support for UserManager.
PasswordPolicyException	Class	This class extends IMException (see above).
Property	Class	Property contains name-value information.
PropertySet	Class	A collection of property name and value pairs. Property class is used to represent the property name and value(s) pair. PropertySet guarantees that no two properties have same name.
Role	Interface	This interface represents a role in the identity store.
RoleManager	Interface	This interface represents a role manager that manages execution of various operations, involving roles, in the identity repository.
RoleProfile	Interface	This interface represents the detailed profile of a role.
SearchFilter	Interface	This interface represents a search filter to be used in searching the identity repository.
SearchParameters	Class	This class represents search parameters that need to be specified while performing searches on the identity store. These search parameters are: ■ Search filter, ■ Search identity type, ■ page size, ■ time limit, and ■ count limit.
SearchResponse	Interface	This interface represents search results obtained after searching the identity store. Its implementation is service provider-specific.
SimpleSearchFilter	Interface	This interface represents a simple search filter to be used while searching the identity repository. Each simple search filter is a logical expression consisting of a search attribute/property, evaluation operator and value. This logical expression will be applied to the underlying identity repository while searching and matching results will be filtered out. See the Javadoc (Section 20.9, "User and Role API Reference") for an example of a simple search filter.
StoreConfiguration	Interface	StoreConfiguration holds the configuration properties for a given IdentityStore instance. The behavior of this IdentityStore instance can be controlled by changing the properties in this configuration object. The actual configuration properties and their values are specific to the service provider. Some service providers may not support any configuration property at all.
SubjectParser	Interface	This interface provides utility methods for extracting out the user and role principals from the given Subject. Service provider needs to provide the implementation for this interface.
User	Interface	This interface represents a user in the identity store.
UserManager	Interface	This interface represents a user manager that manages execution of various operations, involving users, in the identity repository.

Table 20–1 (Cont.) Classes and Interfaces in the User and Role API

Name	Type	Description
UserProfile	Interface	This interface represents the detailed profile of a user. It allows for user properties to be accessed in a generic manner. You can read or modify any property of `user` with these APIs: ■ getProperty(java.lang.String) ■ getProperties(java.lang.String[]) ■ setProperty(oracle.security.idm.ModProperty) ■ setProperties(oracle.security.idm.ModProperty[])

20.3 Working with Service Providers

In this section we describe basic provider concepts and lifecycle, and explain how to set up, configure, and use the provider to work with user repositories in an Oracle Platform Security Services environment.

After ensuring the environment is properly set up, implementing the provider involves:

- identifying the underlying repository and selecting the provider factory class appropriate to that repository
- creating instances of the provider factory and the identity store
- configuring the provider

This section contains these topics:

- Understanding Service Providers
- Setting Up the Environment
- Selecting the Provider
- Properties for Provider Configuration
- Programming Considerations
- Provider Lifecycle

20.3.1 Understanding Service Providers

Although the User and Role API is called for user and role management, the API does not directly interact with the underlying identity repository. Instead, security applications make use of *providers* which carry out the actual communication with the underlying repository. This offers flexibility since the same code can be used with various underlying repositories simply by modifying the provider/connection information.

20.3.2 Setting Up the Environment

This section describes how to configure the environment for the User and Role API.

20.3.2.1 Jar Configuration

Several jars must be present in your environment:

- the provider jar file, which implements the desired underlying identity repository
- the User and Role API jars

- other component jars which the provider may need, including Toplink, jdbc, xdb, and so on

Ensure that your application classpath includes the relevant jars.

20.3.2.2 User Classes in jps-config.xml (Oracle Virtual Directory only)

Note: Make this change only for the Oracle Virtual Directory authenticator.

For efficiency when fetching user attributes, add the following entry in `jps-config.xml` to specify the user object classes for the search:

```
.
.
     <serviceInstance name="idstore.ldap" provider="idstore.ldap.provider">
            <property name="idstore.config.provider"
value="oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfigProvider"/
>
            <property name="CONNECTION_POOL_CLASS"
value="oracle.security.idm.providers.stdldap.JNDIPool"/>
          <extendedProperty>
            <name>user.object.classes</name>
            <values>
               <value>top</value>
               <value>person</value>
               <value>inetorgperson</value>
               <value>organizationalperson</value>
               <value>otherActiveDirectorySpecificClasses</value>
               ...
            </values>
          </extendedProperty>
.
.
```

20.3.2.3 Reading Privileges for Provider User (Oracle Internet Directory Only)

For the Oracle Internet Directory provider, the configured user should have privileges to read the "`cn=common,cn=products,cn=oraclecontext`" container at the root level.

20.3.3 Selecting the Provider

Oracle Platform Security Services support a range of user repositories, including the following LDAP directories:

- Microsoft Active Directory
- Novell eDirectory
- Oracle Directory Server Enterprise Edition
- Oracle Internet Directory
- Oracle Virtual Directory
- OpenLDAP
- Oracle WebLogic Server Embedded LDAP Directory

- Microsoft ADAM
- IBM Tivoli

The choice of identity repository dictates the provider class to use with the provider. The provider class must implement the interface specified by the User and Role API framework. Table 20–2 shows the available provider classes:

Table 20–2 LDAP Identity Provider Classes

Provider	Factory Name
Microsoft Active Directory	oracle.security.idm.providers.ad.ADIdentityStoreFactory
Novell eDirectory	oracle.security.idm.providers.edir.EDIdentityStoreFactory
Oracle Directory Server Enterprise Edition	oracle.security.idm.providers.iplanet.IPIdentityStoreFactory
Oracle Internet Directory	oracle.security.idm.providers.oid.OIDIdentityStoreFactory
OpenLDAP	oracle.security.idm.providers.openldap.OLdapIdentityStoreFactory
Oracle WebLogic Server Embedded LDAP Directory	oracle.security.idm.providers.wlsldap.WLSLDAPIdentityStoreFactory
Oracle Virtual Directory	oracle.security.idm.providers.ovd.OVDIdentityStoreFactory
Microsoft ADAM	oracle.security.idm.providers.ad.ADIdentityStoreFactory
IBM Tivoli	oracle.security.idm.providers.openldap.OLdapIdentityStoreFactory

20.3.4 Creating the Provider Instance

Once the provider's class name is identified, take these steps to create the provider:

1. Use the `getIdentityStoreFactory` method of the IdentityStoreFactoryBuilder class to build a factory instance. The builder class API accepts:
   - the provider class name
   - the necessary environment properties from a hash table
2. Use the `getIdentityStoreInstance` method of the IdentityStoreFactory class to create a store instance

The following example creates a factory instance for the Oracle Internet Directory store:

```
IdentityStoreFactoryBuilder builder = new
   IdentityStoreFactoryBuilder ();

IdentityStoreFactory oidFactory = builder.getIdentityStoreFactory(
   "oracle.security.idm.providers.oid.OIDIdentityStoreFactory", factEnv);
```

Now obtain the store reference, which is the actual handle to the identity store:

```
oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
```

Note that two hash-table objects are supplied in these examples:

- the factEnv hash table provides the factory instance environment

- the storeEnv hash table provides the store instance environment

To learn more about these hash tables see the example in Section 20.7.1.

20.3.5 Properties for Provider Configuration

Configuration is dependent on the identity store provider being used.

You can fine-tune the behavior of all types of LDAP-based identity store providers by configuring a number of properties for the factory instance and the store instance. The following properties are relevant for LDAP-based providers only:

- URL
- the port at which the repository runs
- the user and password to use in accessing the repository

For a list of supported LDAP-based providers, see Section 20.3.3, "Selecting the Provider".

This section explains the following provider configuration topics:

- Start-time and Run-time Configuration
- ECID Propagation
- When to Pass Configuration Values

20.3.5.1 Start-time and Run-time Configuration

The properties that can be configured fall into two categories:

- Start-time configuration - the naming convention uses property names starting with ST_.
- Run-time configuration - the naming convention uses property names starting with RT_.

Start-time Configuration Properties

Start-time configuration is performed only once, and once set, the configuration settings persist for the duration of the provider's lifetime.

With the exception of ST_SUBSCRIBER_NAME, the start-time properties are specified when creating the provider factory instance; ST_SUBSCRIBER_NAME is set when creating the store instance.

Table 20–3 lists the start-time configuration properties:

Table 20–3 Start-time Identity Provider Configuration Properties

Property Name	Description
ST_BINARY_ATTRIBUTES	An array of Array of String objects containing the names of binary attributes stored in the underlying LDAP server. The provider will treat these attributes as binary while sending data to and receiving it from the LDAP server.
ST_CONNECTION_POOL	External connection pool, an instance of class oracle.idm.connection.ConnectionPool. If set, the provider uses this pool to acquire connections to the LDAP server, and the properties ST_SECURITY_PRINCIPAL, ST_SECURITY_CREDENTIALS, and ST_LDAP_URL are ignored.
ST_USER_NAME_ATTR	The attribute used to determine the username of the user in the identity repository.

Table 20–3 (Cont.) Start-time Identity Provider Configuration Properties

Property Name	Description
ST_GROUP_NAME_ATTR	The attribute used to determine the role name in the identity repository.
ST_USER_LOGIN_ATTR	The attribute used to determine the login ID of the user in the identity repository.
ST_SECURITY_PRINCIPAL	The user (principal).
ST_SECURITY_CREDENTIALS	The credentials necessary to log in to the identity repository.
ST_LDAP_URL	The URL of the identity repository.
ST_MAX_SEARCHFILTER_LENGTH	The maximum length of the search filter allowed by the LDAP server.
ST_LOGGER	The logger object that is to be used by the API.
ST_SUBSCRIBER_NAME	The base DN of operations in the LDAP server. This property is specified while creating the IdentityStore instance and is used to determine default values for remaining properties. This property must be specified while creating the IdentityStore instance; however, subsequent changes to its value have no effect on IdentityStore behavior.
ST_CONNECTION_POOL_CLASS	The fully-qualified Connection Pool implementation class name.
ST_INITIAL_CONTEXT_FACTORY	The fully-qualified class name of the initial context factory that will create the initial context.

Run-time Configuration Properties

Properties set at runtime affect all subsequent operations executed by the provider and control the behavior of the IdentityStore instance of the provider.

Runtime properties are configured by specifying the appropriate parameters and values for the StoreConfiguration object obtained from the IdentityStore instance. All runtime properties have default values when the IdentityStore instance is created, and can be subsequently changed.

Table 20–4 lists the run-time configuration properties:

Table 20–4 Runtime Identity Provider Configuration Properties

Property Name	Description
RT_USER_OBJECT_CLASSES	array of object classes required to create a user in the LDAP server
RT_USER_MANDATORY_ATTRS	attribute names that must be specified while creating a user
RT_USER_CREATE_BASES	Base DNs in the LDAP server where a new user can be created
RT_USER_SEARCH_BASES	
RT_USER_SEARCH_BASES	the base DNs in the LDAP server that can be searched for users
RT_USER_FILTER_OBJECT_CLASSES	array of object classes to use when searching for a user in the LDAP server
RT_GROUP_OBJECT_CLASSES	array of object classes required to create a role in the LDAP server
RT_GROUP_MANDATORY_ATTRS	attribute names that must be specified when creating a role

Table 20–4 (Cont.) Runtime Identity Provider Configuration Properties

Property Name	Description
RT_GROUP_CREATE_BASES	the base DNs in the LDAP server where a new role can be created
RT_GROUP_SEARCH_BASES	the base DNs in the LDAP server that can be searched for a role
RT_GROUP_MEMBER_ATTRS	An array of member attribute(s) in a role. All members of a role have value(s) for the attribute(s).
RT_GROUP_FILTER_OBJECT_CLASSES	an array of object classes to use when searching for a role in the LDAP server
RT_USER_SELECTED_CREATE_BASE	The currently selected user create base. The user will be created in this base DN upon execution of the createUser() call. If the selected create base is null and the ST_SUBSCRIBER_NAME is not specified, the first supplied value of the RT_USER_CREATE_BASE is used. If the ST_SUBSCRIBER_NAME is specified, the default value is relative to the subscriber name based on the identity store type.
RT_GROUP_SELECTED_CREATE_BASE	The currently selected role create base. This role will be created in this base DN upon execution of the createRole() call. If the selected create base is null and the ST_SUBSCRIBER_NAME is not specified, the first supplied value of the RT_GROUP_CREATE_BASE is used. If the ST_SUBSCRIBER_NAME is specified, the default value is relative to the subscriber name based on the identity store type.
RT_GROUP_GENERIC_SEARCH_BASE	A generic role search base to use in searching the roles related to a given identity. For example while searching all granted roles for a user, or all managed roles for a user, we need a search base under which all the required groups would reside; this helps in optimizing the searches. This search base is usually a common parent. By default, in all LDAP providers this value is set to the subscriber name if provider, else it uses the first group search base.
RT_SEARCH_TYPE	determines whether a search on the LDAP server should be of type SIMPLE, PAGED, or VIRTUAL_LIST_VIEW

20.3.5.2 ECID Propagation

By default, ECID support is disabled in the User and Role API.

When initializing the API, set the `ST_ECID_ENABLED` property to true for ECID support, as illustrated in the following example:

```
factEnv.put(OVDIdentityStoreFactory.ST_ECID_ENABLED, "true");
```

Note: This action is necessary only if either Oracle Internet Directory or Oracle Virtual Directory is used as the back-end identity store. It is not necessary if using other repositories such as Microsoft Active Directory or Novell eDirectory.

20.3.5.3 When to Pass Configuration Values

You can specify configuration data:

- when creating a factory instance

> **See Also:** Section 20.3.6, "Configuring the Provider When Creating a Factory Instance"

- when creating a store instance

> **See Also:** Section 20.3.7, "Configuring the Provider When Creating a Store Instance"

- at runtime, through a store configuration object

> **See Also:** Section 20.3.8, "Runtime Configuration"

20.3.6 Configuring the Provider When Creating a Factory Instance

This section contains topics related to configuring the provider during factory instance creation.

Configuration at this stage affects the entire factory object as well as objects created using this specific factory instance. Many start-time properties are set at this time, including these common properties:

- ST_LDAP_URL - the URL of the LDAP repository
- ST_SECURITY_PRINCIPAL - the user name
- ST_SECURITY_CREDENTIAL - the user credentials required to connect to the repository

20.3.6.1 Oracle Internet Directory Provider

In this example, the provider is configured when setting up an Oracle Internet Directory (OID) factory:

```
IdentityStoreFactoryBuilder builder = new
   IdentityStoreFactoryBuilder();
IdentityStoreFactory oidFactory = null;
Hashtable factEnv = new Hashtable();

// Creating the factory instance
factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_PRINCIPAL,
   "<User DN>");
factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
   "<User password>");
factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
   "ldap://ldaphost:port/");
oidFactory = builder.getIdentityStoreFactory(
   "oracle.security.idm.providers.oid.
   OIDIdentityStoreFactory", factEnv);
```

> **Note:** The values in italics must be replaced with appropriate values prior to execution.

20.3.6.2 Using Existing Logger Objects

You can supply named logger objects to the User and Role API. The API uses the specified logger to log messages. You must supply the external logger's name as an environment variable during the factory creation.

Here is an example:

```
Logger mylogr = Logger.getLogger("mylogger.abc.com");
FileHandler fh = new FileHandler("userroleapi.log");
mylogr.addHandler(fh);

…

factEnv.put(OIDIdentityStoreFactory.ST_LOGGER_NAME,
"mylogger.abc.com");
oidFactory = builder.getIdentityStoreFactory(
   "oracle.security.idm.providers.oid.
   OIDIdentityStoreFactory", factEnv);
```

This code directs that all the log messages should be redirected to the log file named `userroleapi.log`.

20.3.6.3 Supplying Constant Values

You can overwrite constants or pre-supply values for missing constants by supplying the map in the ST_PROPERTY_ATTRIBUTE_MAPPING property during factory creation.

This example code sets the mapping of RoleProfile.OWNER to the "myowner" attribute. In this way, all operations related to the owner, such as getOwners(), getOwnedRoles(), and so on, are performed using this attribute.

```
factEnv.put
   (IPIdentityStoreFactory.ST_SECURITY_PRINCIPAL, "<User DN>");
factEnv.put
   (IPIdentityStoreFactory.ST_SECURITY_CREDENTIALS, "<User password>");
factEnv.put(IPIdentityStoreFactory.ST_LDAP_URL,
   "ldap://ldaphost:port/");

Map m = new Hashtable();
m.put(RoleProfile.OWNER, "myowner");

factEnv.put
   (IPIdentityStoreFactory.ST_PROPERTY_ATTRIBUTE_MAPPING, m);

ipFactory = builder.getIdentityStoreFactory(
   "oracle.security.idm.providers.iplanet.IPIdentityStoreFactory",
   factEnv);
```

20.3.6.4 Configuring Connection Parameters

You can configure the connection pool parameters for minimum/maximum connections using ST_CONNECTION_POOL_MIN_CONNECTIONS and ST_CONNECTION_POOL_MAX_CONNECTIONS respectively. By default, the values for these parameters are "0" and "10" respectively. There is an additional restriction that:

```
(ST_CONNECTION_POOL_MAX_CONNECTIONS  - ST_CONNECTION_POOL_MIN_CONNECTIONS) >= 10
```

Here is an example:

```
factEnv.put
   (LDIdentityStoreFactory.ST_CONNECTION_POOL_MIN_CONNECTIONS,  "3");

factEnv.put
   (LDIdentityStoreFactory.ST_CONNECTION_POOL_MAX_CONNECTIONS, "16");
```

20.3.6.5 Configuring a Custom Connection Pool Class

To use a custom connection pool, you must provide the fully qualified class name of the custom connection pool class, as follows:

```
factEnv.put(OIDIdentityStoreFactory.ST_CONNECTION_POOL_CLASS,
"oracle.security.idm.providers.stdldap.JNDIPool");
```

For related information, see Section J.6.1, "Failure to Connect to the Embedded LDAP Authenticator."

20.3.7 Configuring the Provider When Creating a Store Instance

The IdentityStore configuration affects the store object and all objects that are created using this store instance. A configuration parameter commonly used with the store is ST_SUBSCRIBER_NAME, which is the only start-time property accepted here. (All the runtime properties can be supplied during identity store creation.)

Continuing with the earlier example in Section 20.3.6, "Configuring the Provider When Creating a Factory Instance" which created a factory instance, this code creates a handle instance to the store.

```
IdentityStore oidStore = null;
Hashtable storeEnv = new Hashtable();

// Creating the store instance
storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,
   "dc=us,dc=oracle,dc=com");
oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
```

Note: Directories require that you supply a valid subscriber name. For Oracle Internet Directory, you can supply the STsubscriber name as either a proper DN or as the nickname of the realm.

20.3.8 Runtime Configuration

Earlier, in Section 20.3.6, "Configuring the Provider When Creating a Factory Instance" and Section 20.3.7, "Configuring the Provider When Creating a Store Instance", we demonstrated how to perform configuration when creating an instance. To facilitate adding and modifying properties at runtime, the User and Role APIs also provide a Configuration class.

The Configuration instance can be obtained from the store instance using the `IdentityStore.getStoreConfiguration()` API call. Properties can be modified using the configuration object.

Only *runtime* properties can be modified using this approach, and the effect is visible only at runtime.

This example sets the RT_USER_SEARCH_BASES property:

```
StoreConfiguration conf = oidStore.getStoreConfiguration();
conf.setProperty("RT_USER_SEARCH_BASES", "dc=us,dc=oracle,dc=com");
```

20.3.9 Programming Considerations

This section contains tips for working with providers and provider artifacts.

20.3.9.1 Provider Portability Considerations

To ensure that your application is portable when switching providers (say, from OpenLDAP provider to Oracle Internet Directory provider or the converse), follow these guidelines when working with the User and Role API:

1. Use only the corresponding `oracle.security.idm.UserProfile` constants to refer to user properties. Avoid using native names which are not portable across identity repositories. For example, if the application needs to obtain a user's login name, fetch it using the `UserProfile.USER_NAME` constant:

   ```
   Property prop = usrprofile.getProperty(UserProfile.USER_NAME);
   ```

2. For obvious reasons, `UserProfile` constants are provided for most standard user properties but not for all possible properties. If the application needs to obtain all the properties of a user generically, use the following code:

   ```
   UserProfile upf = null;

   // Obtain the user profile from user object. User object can be obtained using
   search

   // get the properties supported for given user in currently configured
   repository
   List proplst = store.getUserPropertyNames();

   String[] proparr = (String[]) proplst.toArray(new String[proplst.size()]);

   // get all properties of the user
   PropertySet pset = upf.getProperties(proparr);
   ```

3. When creating search filters, do not use native wild card characters directly in your search filter string. Instead, use the `SimpleSearchFilter.getWildCardChar()` method. This will fetch the correct wild character based upon the underlying provider. For example, the API will return `%` for a database provider and return `*` for the Oracle Internet Directory provider.

   ```
   SmpleSearchFilter sf = m_identityStore.getSimpleSearchFilter(
      attrName, SimpleSearchFilter.TYPE_EQUAL, null);

   sf.setValue( filterStringWithoutWildcard+sf.getWildCardChar());
   ```

4. If your application accepts user-supplied filter strings with a predefined wild card character, apply the following conversion on the filter while generating the User and Role API filter:

   ```
   //User supplied filter which assumes "%" as the wildcard character

   String userDefinedFilter = ................

   SmpleSearchFilter sf = m_identityStore.getSimpleSearchFilter(
      attrName, SimpleSearchFilter.TYPE_EQUAL, null);

   userDefinedFilter =
      userDefinedFilter.replaceall("%", sf.getWildCardChar());

   sf.setValue(userDefinedFilter);
   ```

   The line in bold converts the user-supplied filter to the generic User and Role API filter format.

20.3.9.2 Considerations when Using IdentityStore Objects

Keep the following considerations in mind when coding your applications.

Thread Safety

The current IdentityStore implementations are not thread-safe. The User and Role API assumes that the store instances are not generally shared among threads. If the store instance is shared among threads, the application code must take care to handle any required thread safety issues.

There are trade-offs between thread safety and performance. Use cases that need to implement thread safety must be willing to consider the performance implications of doing so.

One Store Instance per Session

In applications such as Delegated Administration Server, each session (corresponding to one logged-in user) can change its own create/search bases and various other runtime settings; these are defined as runtime properties in the User and Role API. The IdentityStore object encapsulates all these settings and changes its runtime behavior accordingly. For this reason, the rule of one IdentityStore instance per session is enforced.

20.3.10 Provider Lifecycle

A given provider exists for the lifetime of the factory instance created for that provider. The life of a factory instance ends whenever the close() API is called on that instance. When the provider instance ends, all the objects that were created using that instance become invalid, and subsequent API calls on those objects return unanticipated output.

Similar considerations apply to IdentityStore instances.

Note:

- Factory instances are thread-safe while this is not the case with IdentityStore instances.
- It is best practice to cascade close server connections and explicitly delete objects and instances no longer in use.

20.4 Searching the Repository

The User and Role API provides two types of query functions:

- functions that return a single identity
- functions that return a list of identities

This section describes searches and related tasks you can accomplish with the API, and provides details on specifying search parameters:

- Searching for a Specific Identity
- Searching for Multiple Identities
- Specifying Search Parameters
- Using Search Filters
- Searching by GUID

20.4.1 Searching for a Specific Identity

You can query the identity store directly for a specific user or role using the `searchUser` and `searchRole` APIs:

```
IdentityStore.searchUser(String name);

IdentityStore.searchUser(Principal principal);

IdentityStore.searchUser(int searchType, String name);
```

where `searchType` can be:

- SEARCH_BY_NAME
- SEARCH_BY_UNIQUE_NAME

```
IdentityStore.searchRole(int searchType, String value);
```

These functions facilitate simple queries where a particular user/role identity is known to exist in the store, and you simply need the object reference to that identity. The functions are minimal in that:

- they accept only string values
- they do not support regular expressions

The functions raise an exception if multiple entities with the same value exist in the store.

20.4.2 Searching for Multiple Identities

The User and Role APIs contain several functions that can perform searches to return multiple identities:

```
IdentityStore.search(SearchParams params);
IdentityStore.searchUsers(SearchParams params);
IdentityStore.searchRoles(int searchType, SearchParams params);
IdentityStore.searchProfiles(SearchParams params);
```

Each function accepts a search object and returns a search response object.

20.4.3 Specifying Search Parameters

The SearchParams Object

The SearchParams object contains the following information:

- Search Filter - this is discussed in Section 20.4.4, "Using Search Filters"
- Search Identity of type - you can search identities of type `Roles` or `Users`
- Page Size - By default the paging option is turned off. If the query needs paging, set the page size to a relevant positive value.
- Timeout limit – timeout is specified in seconds
- Count Limit – limits the number of results returned by the query

The SearchResponse Object

SearchResponse is a data structure used when retrieving multiple identities. Your code can iterate through the identities contained in this structure using these functions:

- `hasNext()` - returns `true` if more elements are present, `false` otherwise
- `next()` – returns the next element if it is available, an exception otherwise

20.4.4 Using Search Filters

The User and Role API includes different types of search filters to facilitate a variety of search operations. This section explains key facts about the use of search filters:

- Operators in Search Filters
- Handling Special Characters when Using Search Filters
- Search Filter for Logged-In User
- Examples of Using Search Filters

20.4.4.1 Operators in Search Filters

Observe these rules when using search filter operators.

Supported Operators

The standard LDAP store accepts only "=" (equals operator), "<" (less-than operator), ">" (greater-than operator), "&" (AND operator), "|" (OR operator) and "!" (NOT operator). IdentityStore provides two more operators to simplify usage, namely "<=" (less than or equal to) and ">=" (greater than or equal to).

The operators "=", "<",">", "<=" and ">=" are used to create simple search filters while the "&" and "|" operators are used to combine two or more search strings to make a complex search filter.

NOT Operator

You can use the `NOT` operator in both the simple search filter and complex search filters. This operator is used to negate the state of the filter, that is, the state of the filter is changed to accept the entities which were earlier rejected by the filter, or to reject entities that were earlier accepted.

The NOT operator is accessible using the following `SearchFilter` API:

- void negate();
- boolean isNegated();

20.4.4.2 Handling Special Characters when Using Search Filters

According to RFC-2254 (String Representation of LDAP Search Filters), "*", "(", ")","\" and NULL characters are to be handled separately. The User and Role API handles "(", ")" and "\" operators but does not handle the "*" operator, which is also a wild-card character for LDAP stores. The API user is not required to separately handle these characters as the User and Role API framework handles these characters.

20.4.4.3 Search Filter for Logged-In User

Applications commonly need to retrieve the identity of the logged-in user and the user's group name.

The Oracle WebLogic Server authenticator uses two attributes related to users: `user.login.attr` and `groupname.attr`. Upon login, the authenticator uses `user.login.attr` to store the user and `groupname.attr` for the group.

Your application should use `UserProfile.getUserName()` (which maps to `user.login.attr`) to obtain the identity of the logged-in user. To obtain the role (group) name, it should use `RoleProfile.getProperty(RoleProfile.NAME)` (which maps to `groupname.attr`).

Sample calls showing how to obtain the logged-in user and role are shown in Example 20–6 and Example 20–7, respectively.

20.4.4.4 Examples of Using Search Filters

Several usage examples are presented in this section.

Example 20–1 Simple Filter to Retrieve Users by Name

The implementation of the simple search filter depends on the underlying store; you can obtain an instance of the search filter through the store instance.

In this example, the filter allows all entries with a non-null value for the "name" field:

```
SimpleSearchFilter sf =
   oidStore.getSimpleSearchFilter(UserProfile.NAME,
   SimpleSearchFilter.TYPE_EQUAL, null);
sf.setValue(sf.getWildCardChar());
```

Example 20–2 Simple Filter to Find Users by Language Preference

This example retrieves users whose preferred language is not English:

```
SimpleSearchFilter sf =
   oidStore.getSimpleSearchFilter(
      UserProfile.PREFERRED_LANGUAGE,
      SimpleSearchFilter.TYPE_EQUAL,
      "english");
sf.negate();
```

Example 20–3 Complex Filter for Names by Starting Letter

This complex filter combines multiple search filters with operators "&" or "|". It searches for users whose name starts with a letter between "a" and "j":

```
SimpleSearchFilter sf1 =
   oidStore.getSimpleSearchFilter(
      UserProfile.NAME,
      SimpleSearchFilter.TYPE_GREATER,
      null);

sf1.setValue("a"+sf1.getWildCardChar());
SimpleSearchFilter sf2 =
   oidStore.getSimpleSearchFilter(UserProfile.NAME,
      SimpleSearchFilter.TYPE_LESS, null);
sf2.setValue("j"+sf2.getWildCardChar());
SimpleSearchFilter sfArray[] = new SimpleSearchFilter[] {sf1, sf2};
ComplexSearchFilter cf1 =
store.getComplexSearchFilter(sfArray, ComplexSearchFilter.TYPE_AND);
```

Example 20–4 Complex Filter with Restrictions on Starting Letter

In this example, complex filters are nested to enable a search for users whose name starts with a letter between "a" and "j" but not with the letter "i":

```
[continue from Example 3]

SimpleSearchFilter sf3 =
   oidStore.getSimpleSearchFilter(
      UserProfile.NAME,
      SimpleSearchFilter.TYPE_EQUAL,
      null);

sf3.setValue("i"+sf3.getWildCardChar());
sf3.negate();

SearchFilter sfArray2[] = new SearchFilter[] {cf1, sf3};
ComplexSearchFilter cf2 =
   store.getComplexSearchFilter(sfArray2, ComplexSearchFilter.TYPE_AND);
```

Example 20–5 Complete Search with Output

This example filters names starting with the letter "a" and outputs the return values:

```
// search filter (cn=a*)
SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
      UserProfile.NAME,
      SimpleSearchFilter.TYPE_EQUAL,
      null);
sf.setValue("a"+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()) {
   Identity idy = resp.next();
   System.out.println("Unique name: "+idy.getUniqueName());
}
```

Example 20–6 Obtaining the Identity of the Logged-in User

This example shows how to retrieve the logged-in user:

```
SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
      UserProfile.USER_NAME,  SimpleSearchFilter.TYPE_EQUAL, "sampleUserName");
   SearchParameters ssp = new SearchParameters(sf, SearchParameters.SEARCH_USERS_
ONLY);

   // Searching for users
   SearchResponse resp = oidStore.searchUsers(ssp);
   System.out.println("Searched users are:");
   while (resp.hasNext()) {
       Identity idy = resp.next();
       String foundUserName = ((User)idy).getUserProfile().getUserName();
       System.out.println("Found user name: "+ foundUserName );
   }
```

Note: The name returned by `((User)idy).getName` is derived from the RDN, which might be different from the login name.

Example 20–7 Obtaining the Role/Group Name

This example shows how to retrieve the role (group) name:

```
Role aRole = idStore.searchRole(IdentityStore.SEARCH_BY_NAME, "sampleRoleName");
    Property prop = aRole.getRoleProfile().getProperty(RoleProfile.NAME);

    List roleList = prop.getValues();
    Iterator itr = roleList.iterator();
   System.out.println("Searched roles are:");
    while (itr.hasNext()) {
        String foundRoleName = (String)itr.next();
        System.out.println("Found role name: "+ foundRoleName );
    }
```

20.4.5 Searching by GUID

In this example, GUID values obtained from the User and Role API can be directly used in the search:

```
// up = user.getUserProfile();
String guid = up.getGUID();
SimpleSearchFilter sf1 = oidStore.getSimpleSearchFilter(
   UserProfile.GUID,
   SimpleSearchFilter.TYPE_EQUAL, guid);
SearchParameters params = new SearchParameters();
params.setFilter(sf1);
SearchResponse resp = oidStore.search(params);
while (resp.hasNext())
   System.out.println("user for guid : " + guid + ","+ resp.next());
```

20.5 Authenticating Users

For verification purposes, you can use the User and Role API for password-based authentication of users. (As mentioned earlier, the API is not meant for authentication and authorization.)

The `authenticateUser` API accepts a user login name and attempts to authenticate the user with the specified password. If authentication is successful, it returns the user object.

Here is an example of password-based authentication:

```
store.getUserManager().authenticateUser("testuser","password");
```

20.6 Creating and Modifying Entries in the Identity Store

The User and Role API facilitates adding new identities to the identity store and modifying identities in the store. The UserManager and RoleManager classes address the user- and role-specific data creation, modification and deletion operations.

UserManager and RoleManager instances can be obtained from the store instance as follows:

```
UserManager um = oidStore.getUserManager();
RoleManager rm = oidStore.getRoleManager();
```

Topics in this section include:

- Handling Special Characters when Creating Identities
- Creating an Identity
- Modifying an Identity
- Deleting an Identity

20.6.1 Handling Special Characters when Creating Identities

RFC-2253 defines the string representation of Distinguished Names for LDAP v3. This means that all the characters specified in the RFC are handled. The User and Role API user does not need to escape/de-escape those special characters; attempting to do so will cause erroneous results.

There could be a problem when creating identities with empty properties. In this case, the "RDN name" is used to fill in the values of various mandatory attributes. Some of these attributes could have stricter validation rules. In this case, the creation of the identity fails and an exception is raised.

20.6.2 Creating an Identity

Two functions in the UserManager class facilitate creating a user:

```
createUser(java.lang.String name, char[] password)
```

creates a user with the specified name and password in the underlying repository.

When the identity store designates that some attributes are mandatory, all such fields will be populated with the "name" value.

```
createUser(java.lang.String name, char[] password, PropertySet suppliedProps)
```

Properties are set using the supplied property values. If any mandatory attribute values are not supplied, the missing attributes will use the "name" value as the default.

Likewise, RoleManager APIs are used to create roles.

Roles are organized into two categories:

- application scope
- enterprise scope

When you invoke `RoleManager` to create a role, by default the role is created in the enterprise scope unless you specify otherwise.

RoleManager APIs supporting role creation are:

```
createRole(String roleName);
createRole(String roleName, int roleScope);
```

The procedure for creating a role is similar to that for creating a user, and all mandatory attributes must be supplied with `roleName`.

20.6.3 Modifying an Identity

To modify an identity, you need a reference to the identity. The User, UserProfile, Role, and RoleProfile classes provide the following APIs to facilitate modifying identities:

```
user.setProperty(ModProperty prop);

user.setProperties(ModProperty [] props);
```

ModProperty structure consists of:

- the field name
- its new value(s)
- the modifying operator

Valid operators are:

```
ModProperty.ADD
ModProperty.REMOVE
ModProperty.REPLACE
```

In this example, a display name is replaced:

```
UserProfile usrprofile = usr.getUserProfile();

ModProperty mprop = new ModProperty(UserProfile.DISPLAY_NAME,
   "modified display name",
   ModProperty.REPLACE);

usrprofile.setProperty(mprop);
```

Modifying a particular value in a multi-valued attribute is a two-step process; first remove the value, then add the new value.

20.6.4 Deleting an Identity

You drop an identity with the `dropUser` and `dropRole` APIs.

You need both user and role references in your code when dropping an identity. Here is an example:

```
User usr;
Role role;
…
…
usrmanager.dropUser(usr);
rolemanager.dropRole(role);
```

20.7 Examples of User and Role API Usage

See Also: Section 25.1, "Using OPSS in Java SE Applications".

This section contains some examples illustrating practical applications of the User and Role API:

- Example 1: Searching for Users
- Example 2: Understanding User Management in an Oracle Internet Directory Store
- Example 3: Understanding User Management in a Microsoft Active Directory Store

20.7.1 Example 1: Searching for Users

In this example the identity store is Oracle Internet Directory, and a simple search filter is set up to search for users:

```
import oracle.security.idm.*;
import oracle.security.idm.providers.oid.*;
import java.util.*;
import java.io.*;

public class SearchUsersOID
{
  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,
                   "<User DN>");
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
                   "<User password>");
      factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
                  "ldap://ldaphost:port/");
oidFactory =  builder.getIdentityStoreFactory(
              "oracle.security.idm.providers.oid.OIDIdentityStoreFactory",
              factEnv);

      // creating the store instance
         storeEnv.put(OIDIdentityStoreFactory.RT_SUBSCRIBER_NAME,
         "<Subscriber name>");
      oidStore = oidFactory.getIdentityStoreInstance(storeEnv);

      // search filter (cn=a*)
      SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
                    UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
      sf.setValue("a"+sf.getWildCardChar());
// sf2 search filter (!(cn=*a))
SimpleSearchFilter sf2 = oidStore.getSimpleSearchFilter(
                    UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, null);
sf2.setValue(sf2.getWildCardChar()+"a");
sf2.negate();

SimpleSearchFilter sfArray[] = new SimpleSearchFilter[] {sf,sf2};
ComplexSearchFilter cf1 = oidStore.getComplexSearchFilter(sfArray,
ComplexSearchFilter.TYPE_AND);

SearchParameters params = new SearchParameters();
params.setFilter(cf1);

 // Searching for users
SearchResponse resp = oidStore.searchUsers(params);
System.out.println("Searched users are:");
while (resp.hasNext()) {
    Identity idy = resp.next();
    System.out.println("Unique name: "+idy.getUniqueName());
}
```

```
    }catch (IMException e)
    {
      e.printStackTrace();
    }
  }
}
```

Searching for Users and Searching for Groups

When searching for users, you invoke `UserProfile`, as in the above example with `SimpleSearchFilter`. When searching for groups, however, you use `RoleProfile` instead.

20.7.2 Example 2: Understanding User Management in an Oracle Internet Directory Store

In this example several user management tasks such as creating, modifying, and dropping an identity are performed in an Oracle Internet Directory store:

- creating a user
- modifying the user's display name
- dropping the user

```
public class CreateModifyDeleteUser
{
  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,
                   "<User DN>");
      factEnv.put(OIDIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
                   "<User password>");
      factEnv.put(OIDIdentityStoreFactory.ST_LDAP_URL,
                  "ldap://ldaphost:port/");
      oidFactory =  builder.getIdentityStoreFactory(
                  "oracle.security.idm.providers.oid.
OIDIdentityStoreFactory",
               factEnv);

      // creating the store instance
      storeEnv.put(OIDIdentityStoreFactory.RT_SUBSCRIBER_NAME,
                   "dc=us,dc=oracle,dc=com");
      oidStore = oidFactory.getIdentityStoreInstance(storeEnv);

      //get UserManager
      UserManager usrmanager = oidStore.getUserManager();
      // create user
      String usrname = "testuser";
      // delete user if already exists
      try
      {
        User usr = oidStore.searchUser(usrname);
        usrmanager.dropUser(usr);
      }catch(IMException ime){}

      System.out.println("creating user "+usrname);
      User usr =
usrmanager.createUser(usrname,"passwd1".toCharArray());
      System.out.println("user (" +usr.getUniqueName() + ") created");

      // modifying user properties
      System.out.println("modifying property
UserProfile.DISPLAY_NAME");
      UserProfile usrprofile = usr.getUserProfile();
      ModProperty mprop = new ModProperty(
UserProfile.DISPLAY_NAME,
                          "modified display name",
                          ModProperty.REPLACE);

      usrprofile.setProperty(mprop);

      System.out.println("get property values
UserProfile.DISPLAY_NAME");
      Property prop = usrprofile.getProperty(UserProfile.DISPLAY_NAME);
      List values = prop.getValues();
      Iterator itr = values.iterator();
      while(itr.hasNext()) {
        System.out.println(UserProfile.DISPLAY_NAME+": "+ itr.next());
      }
      System.out.println();

      // drop user
      System.out.println("Now dropping user "+usrname);
      usrmanager.dropUser(usr);
      System.out.println("user dropped");

    }catch (IMException e)
    {
      e.printStackTrace();
    }
  }
}
```

20.7.3 Example 3: Understanding User Management in a Microsoft Active Directory Store

In this example several user management tasks such as creating, modifying, and dropping an identity are performed in a Microsoft Active Directory store:

- creating a user
- modifying the user's display name
- dropping the user

```
package oracle.security.idm.samples;
```

```
import oracle.security.idm.*;
import oracle.security.idm.providers.ad.*;
import java.util.*;
import java.io.*;

public class CreateModifyDeleteUserAD
{
  public static void main(String args[])
  {
    IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();
    IdentityStoreFactory adFactory = null;
    IdentityStore adStore = null;

    try
    {

      Hashtable factEnv = new Hashtable();
      Hashtable storeEnv = new Hashtable();

      String keystore = "/home/bhusingh/client_keystore.jks";
      System.setProperty("javax.net.ssl.trustStore",keystore);
      System.setProperty("javax.net.ssl.trustStorePassword","welcome1");

      // creating the factory instance
      factEnv.put(ADIdentityStoreFactory.ST_SECURITY_PRINCIPAL,
                  "sramaset@xyzt.com");
      factEnv.put(ADIdentityStoreFactory.ST_SECURITY_CREDENTIALS,
                   "ntrtntrt");
      factEnv.put(ADIdentityStoreFactory.ST_LDAP_URL,
                  "ldaps://mynode.us.mycorp.com:123/");
      factEnv.put("java.naming.security.protocol","SSL");


      adFactory =  builder.getIdentityStoreFactory(
                    "oracle.security.idm.providers.ad.ADIdentityStoreFactory",
                    factEnv);

      // creating the store instance
      storeEnv.put(ADIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                   "dc=upad,dc=us,dc=oracle,dc=com");
      adStore = adFactory.getIdentityStoreInstance(storeEnv);

      //get UserManager
      UserManager usrmanager = adStore.getUserManager();

      // create user
      String usrname = "amyd";
      // delete user if already exists
      try
      {
        User usr = adStore.searchUser(usrname);
        usrmanager.dropUser(usr);
      }catch(IMException ime){}

      System.out.println("creating user "+usrname);
      char[] password = {'w', 'e', 'l', 'c', 'o', 'm','e','3'};
      User usr = usrmanager.createUser(usrname, password);
      System.out.println("user (" +usr.getUniqueName() + ") created with
guid="+usr.getGUID());
```

```
        System.out.println("user name = "+usr.getName() );

        // modifying user properties
        System.out.println("DISPLAY_NAME="+usr.getDisplayName());
        System.out.println("modifying property UserProfile.DISPLAY_NAME");
        UserProfile usrprofile = usr.getUserProfile();
        ModProperty mprop = new ModProperty(UserProfile.DISPLAY_NAME,
                                            "modified display name",
                                            ModProperty.REPLACE);
        usrprofile.setProperty(mprop);

        System.out.println("get property values UserProfile.DISPLAY_NAME");
        Property prop = usrprofile.getProperty(UserProfile.DISPLAY_NAME);
        List values = prop.getValues();
        Iterator itr = values.iterator();
        while(itr.hasNext())
        {
          System.out.println(UserProfile.DISPLAY_NAME+": "+ itr.next());
        }
        System.out.println();

        System.out.println("now verifying the password");
        boolean pass = false;
        try
        {
          usrmanager.authenticateUser(usrname, password);
          pass= true;
        }catch (oracle.security.idm.AuthenticationException e)
        {
          System.out.println(e);
          e.printStackTrace();
        }
        if (pass)
          System.out.println("password verification SUCCESS !!");
        else
          System.out.println("password verification FAILED !!");

        SimpleSearchFilter sf = adStore.getSimpleSearchFilter(
                      UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, usrname);

        SearchParameters params = new SearchParameters();
        params.setFilter(sf);

        // Searching for users
        SearchResponse resp = adStore.searchUsers(params);
        System.out.println("Searched users are:");
        while (resp.hasNext())
        {
          Identity idy = resp.next();
          System.out.println("name: "+idy.getName()+"\tUnique name:
"+idy.getUniqueName());
        }


        // drop user
        System.out.println("Now dropping user "+usrname);
        usrmanager.dropUser(usr);
        System.out.println("user dropped");

      }catch (Exception e)
```

```
    {
      e.printStackTrace();
    }
  }
}
```

20.8 SSL Configuration for LDAP-based User and Role API Providers

This section describes SSL support for the User and Role API. It contains these topics:

- Out-of-the-box Support for SSL
- Customizing SSL Support for the User and Role API

20.8.1 Out-of-the-box Support for SSL

LDAP-based providers for the User and Role API rely on the Sun Java Secure Sockets Extension (JSSE) to provide secure SSL communication with LDAP-based identity stores. JSSE is part of JDK 1.4 and higher.

These LDAP providers are:

- Microsoft Active Directory
- Novell eDirectory
- Oracle Directory Server Enterprise Edition
- Oracle Internet Directory
- OpenLDAP
- Oracle WebLogic Server Embedded LDAP Directory

20.8.1.1 System Properties

To support SSL you must provide the following information in the form of system properties:

```
javax.net.ssl.keyStore

javax.net.ssl.keyStorePassword

javax.net.ssl.trustStore

javax.net.ssl.trustStorePassword
```

See the Java Secure Socket Extension (JSSE) Reference Guide (`http://download.oracle.com/javase/6/docs/technotes/guides/security/jsse/JSSERefGuide.html`) for complete information on JSSE.

20.8.1.2 SSL configuration

You need to provide SSL configuration details during User and Role API configuration.

Provide your keystore location and password as system properties to the JVM:

```
String keystore = "<key store location>";
String keypasswd = "<key store password>";
System.setProperty("javax.net.ssl.trustStore",keystore);
System.setProperty("javax.net.ssl.trustStorePassword", keypasswd);
```

Specify following properties in the environment when creating the `IdentityStoreFactory` instance:

1. Set the SSL URL of the LDAP server, as in this example:

   ```
   factEnv.put(ADIdentityStoreFactory.ST_LDAP_URL,
    "ldaps://ldaphost:sslport/");
   ```

2. Set the security protocol to SSL:

   ```
   factEnv.put("java.naming.security.protocol","SSL");
   ```

20.8.2 Customizing SSL Support for the User and Role API

You can customize SSL support by providing a customized `SSLSocketFactory` to the User and Role API provider.

20.8.2.1 SSL configuration

Specify the following properties when creating the `IdentityStoreFactory` instance:

1. Specify the custom SSL socket factory name:

   ```
   factEnv.put("java.naming.ldap.factory.socket",
   "fully qualified custom socket factory name");
   ```

2. Set the SSL URL of the LDAP server, as in this example:

   ```
   factEnv.put(ADIdentityStoreFactory.ST_LDAP_URL,
    "ldaps://ldaphost:sslport/");
   ```

3. Set the security protocol to SSL:

   ```
   factEnv.put("java.naming.security.protocol","SSL");
   ```

20.9 User and Role API Reference

The User and Role API reference (Javadoc) is available at:

Oracle Fusion Middleware User and Role Java API Reference for Oracle Platform Security Services

20.10 Developing Custom User and Role Providers

This section explains how to develop custom providers that security developers can use to manage identities (users and roles). It contains these topics:

- SPI Overview
- Types of User and Role Providers
- Developing a Read-Only Provider
- Developing a Full-Featured Provider
- Development Guidelines
- Testing and Verification
- Example: Implementing an Identity Provider

20.10.1 SPI Overview

The User and Role API is accompanied by a service provider interface (SPI) that makes it possible to develop custom user/role providers. You can use the service provider interface to develop a custom provider for any identity data repository.

The SPI is bundled as the `oracle.security.idm.spi` package, which is a set of abstract classes. Custom User and Role providers are created by extending this SPI to fit your requirements.

See Also: "The User and Role SPI Reference"

20.10.2 Types of User and Role Providers

The User and Role API offers functions for both search and Create/Read/Update/Delete (CRUD) operations. A User and Role provider based on read-only functions supports only search operations. A full-featured provider supports both search operations and CRUD operations. In other words, the full-featured provider is a superset of a read-only provider.

As a developer you have the choice of creating either read-only or full-functionality providers depending upon the requirements.

It is reasonable to develop a read-only provider in the following situations:

- if the underlying identity repository operates in read-only mode
- if applications consuming the User and Role API do not make any CRUD API calls

For example, it makes sense to develop a read-only provider for use with the SOA identity service.

20.10.3 Developing a Read-Only Provider

This section describes the classes used to implement a provider. Topics include:

- SPI Classes Requiring Extension
- oracle.security.idm.spi.AbstractIdentityStoreFactory
- oracle.security.idm.spi.AbstractIdentityStore
- oracle.security.idm.spi.AbstractRoleManager
- oracle.security.idm.spi.AbstractUserManager
- oracle.security.idm.spi.AbstractRoleProfile
- oracle.security.idm.spi.AbstractUserProfile
- oracle.security.idm.spi.AbstractSimpleSearchFilter
- oracle.security.idm.spi.AbstractComplexSearchFilter
- oracle.security.idm.spi.AbstractSearchResponse

20.10.3.1 SPI Classes Requiring Extension

Table 20–5 shows that SPI classes that must be extended to implement a read-only provider:

Note: All abstract methods must be implemented.

Table 20–5 SPI Classes to Extend for Custom Provider

Class	Usage Notes
oracle.security.idm.spi.AbstractIdentityStoreFactory	The extending class must include a default constructor and a constructor accepting a java.util.Hashtable object.
oracle.security.idm.spi.AbstractIdentityStore	
oracle.security.idm.spi.AbstractRoleManager	
oracle.security.idm.spi.AbstractUserManager	
oracle.security.idm.spi.AbstractRoleProfile	
oracle.security.idm.spi.AbstractUserProfile	
oracle.security.idm.spi.AbstractSimpleSearchFilter	The constructor of the extending class must call the constructor of the abstract (super) class.
oracle.security.idm.spi.AbstractComplexSearchFilter	The constructor of the extending class must call the constructor of the abstract (super) class.
oracle.security.idm.spi.AbstractSearchResponse	

Additional requirements and notes for each class are provided below.

20.10.3.2 oracle.security.idm.spi.AbstractIdentityStoreFactory

The class extending this SPI class must have following constructors:

1. The default constructor (one which has no arguments).
2. A constructor that accepts a `java.util.Hashtable` object as an argument. You can use the hash table to accept any configuration properties required by the provider.

   The configuration properties are passed to this constructor during the user and role configuration phase. The properties are key-value pairs passed in the `Hashtable` argument:

   - The key must be `java.lang.String`.
   - The value can be `java.lang.Object`.

It is recommended that the value be of type `String`. This guarantees that the property can be specified in `jps-config.xml`, which is a text file.

> **See Also:** "The User and Role SPI Reference" for details about the methods that need to be implemented in this class. All listed methods *must* be implemented.

20.10.3.3 oracle.security.idm.spi.AbstractIdentityStore

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Note that:

- Method `getStoreConfiguration()` is optional and can throw `OperationNotSupportedException`.
- Method `getSubjectParser()` can return `null`.

When there are no search results to be returned, all search APIs should throw:

```
oracle.security.idm.ObjectNotFoundException
```

Never return an empty `SearchResponse`.

20.10.3.4 oracle.security.idm.spi.AbstractRoleManager

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Note that only the following methods need concrete/actual implementations:

- `getGrantedRoles()`
- `getOwnedRoles()`
- `getManagedRoles()`
- `isGranted()`
- `isManagedBy()`
- `isOwnedBy()`
- `isDropRoleSupported()` – should always return `false`
- `isCreateRoleSupported()` – should always return `false`
- `isModifyRoleSupported()` – should always return `false`

The remaining methods must throw the following in their respective implementations:

```
oracle.security.idm.OperationNotSupportedException
```

20.10.3.5 oracle.security.idm.spi.AbstractUserManager

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Only the following methods need concrete/actual implementations:

- `authenticateUser(User, char[])`
- `authenticateUser(String, char[])`
- `isDropUserSupported()` – should always return `false`
- `isCreateUserSupported()` – should always return `false`
- `isModifyUserSupported()` – should always return `false`

The remaining methods must throw the following in their respective implementations:

```
oracle.security.idm.OperationNotSupportedException
```

20.10.3.6 oracle.security.idm.spi.AbstractRoleProfile

oracle.security.idm.spi.AbstractRoleProfile is an abstract class that can be used to return a detailed role profile.

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Only the following methods need concrete/actual implementations:

- `getDisplayName()`
- `getGUID()`
- `getName()`

- `getUniqueName()`
- `getPrincipal()`
- `getDescription()`
- `getGrantees()`
- `getManagers()`
- `getOwners()`
- `getProperty()` - If requested property is not set/valid for corresponding role then null should be returned as value.
- isApplicationRole() - must always return false
- isEnterpriseRole() - must always return false
- isSeeded() - must always return false
- getRoleProfile() – should return reference to current object.

The remaining methods must throw the following in their respective implementations:

`oracle.security.idm.OperationNotSupportedException`

20.10.3.7 oracle.security.idm.spi.AbstractUserProfile

`oracle.security.idm.spi.AbstractUserProfile` is an abstract class that can be used to return a detailed user profile.

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Only the following methods need concrete/actual implementations:

- `getDisplayName()`
- `getGUID()`
- `getName()`
- `getUniqueName()`
- `getPrincipal()`
- `getProperty()` - If the requested property is not set/valid for corresponding role then a null value must be returned.
- `getProperties()` – If the requested property is not set/valid for the corresponding user, then a null value must be returned.
- `getAllUserProperties()` – Only the properties set for the corresponding user should be returned.
- `getReportees()`
- `getManagementChain()`
- getUserProfile() – must return reference to current object.

These two methods:

- `setProperty()`
- `setProperties()`

must throw the following in their implementation:

```
oracle.security.idm.OperationNotSupportedException
```

20.10.3.8 oracle.security.idm.spi.AbstractSimpleSearchFilter

`oracle.security.idm.spi.AbstractSimpleSearchFilter` is an abstract class that can be extended to implement a simple search filter.

The implementing class must have a constructor that calls the constructor of the abstract class:

```
AbstractSimpleSearchFilter (
   String attrname, int type, Object value)
```

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Only the following methods need concrete/actual implementations:

- `getNativeRepresentation()` – convert filter into the native representation to be used with the underlying identity repository.
- `getWildCardChar()` – wild card character, for example "*", to be used in searches. The specific character depends on the underlying identity repository.

20.10.3.9 oracle.security.idm.spi.AbstractComplexSearchFilter

`oracle.security.idm.spi.AbstractComplexSearchFilter` is an abstract class that can be extended to implement a search filter of any complexity.

The implementing class must have a constructor that calls the constructor of the abstract class:

```
AbstractComplexSearchFilter (
   oracle.security.idm.SearchFilter[] filters, int oper_type)
```

"The User and Role SPI Reference" provides details about the methods that need to be implemented in this class. Only the following methods need concrete/actual implementations:

- `getNativeRepresentation()` – convert the filter into the native representation to be used with the underlying identity repository.

20.10.3.10 oracle.security.idm.spi.AbstractSearchResponse

The `SearchResponse` object contains search results being returned from a repository. Each result entry corresponds to one user or role in the underlying identity repository, represented by the corresponding UserProfile/RoleProfile class implementation.

The `SearchResponse` object must return one or more results. This means that the `hasNext()` method must return `TRUE` at least once.

Do not use if there are zero results to return. When no results are to be returned, the corresponding search API should throw the following exception:

```
oracle.security.idm.ObjectNotFoundException
```

20.10.4 Developing a Full-Featured Provider

The full-featured provider implements all the functionality supported by a read-only provider, and additionally supports CRUD operations. This requires that the CRUD APIs be implemented in the SPI implementation classes.

In the read-only provider, these APIs were implemented simply by throwing an `OperationNotSupportedException` (see the class descriptions in Section 20.10.3, "Developing a Read-Only Provider").

For a full-featured provider, this needs to be replaced by concrete/actual implementation of the corresponding CRUD operations.

20.10.5 Development Guidelines

This section provides some guidelines for developing providers.

Mapping of Names

Be aware of the usage of naming constants such as UserProfile.NAME, UNIQUE_NAME, UserProfile.USER_NAME, UserProfile.USER_ID.

- NAME – name of the user or role in the underlying repository.
- UNIQUE_NAME – Complete name with which the user or role is represented in the underlying repository.
- USER_NAME – login ID of the user in the underlying repository.
- USER_ID – always same as USER_NAME constant mapping.

Depending on the identity repository, these constants might map to the same underlying identity repository attribute or they might map to different attributes. If the underlying repository is an LDAP v3 server, the mappings are as follows:

- NAME – mapped to naming attribute of user/group entry, for example "cn"
- UNIQUE_NAME - mapped to "DN" of user/group entry
- USER_NAME/USER_ID – mapped to login attribute, for example "uid" or "mail"

Thread Safety

The following objects are likely to be shared among multiple threads:

- IdentityStoreFactory,
- IdentityStore,
- UserManager,
- RoleManager

You should ensure that there are no thread safety-related issues in the corresponding implementation classes of your provider.

20.10.6 Testing and Verification

The User and Role API ships with a test suite to enable you to test the basic operations of providers that you develop.

The test suite can be used to test both read-only and full-featured providers.

Usage

```
java oracle.security.idm.tests.SPITest propertiesfile
```

where *propertiesfile* contains the provider class name and any configuration data for the provider. It also contains information about the tests to be run.

You need to edit this file and update it with correct information before running the tests; the file contents are self-explanatory.

One such file (`ffprovider.properties`) is available with the sample provider discussed in Section 20.10.7.1, "Working with the Sample Provider".

Results

The test will produce the results on-screen. All providers that you develop must pass the "Lookup tests", "Role membership tests" and "Profile tests" in the test suite. Full-featured providers must pass all the tests in the suite including Create/Drop tests.

The log of test results will be output to the file `results.out` in current working directory.

20.10.7 Example: Implementing an Identity Provider

The distribution includes a sample identity provider that you can use to understand how custom providers are built.

This section describes how to access the sample provider, and explains the steps needed to implement a custom provider. The steps rely on the sample for illustration.

- Working with the Sample Provider
- Setting Up the Sample Provider
- Configuring jps-config.xml to Use the Sample Identity Provider
- Configuring Oracle WebLogic Server

20.10.7.1 Working with the Sample Provider

To help you develop providers, a sample provider is available from Oracle Technology Network. To work with this provider:

1. Navigate to the sample content page at:

   `http://www.oracle.com/technetwork/testcontent/index.html`

2. In the Search box, search for sampleprovider-157582.zip.
3. Download and unzip the file. It will generate the following structure:

```
sampleprovider/
      build.xml - ant build file
      ffprovider.properties - properties file required for testing
      jlib - provider jar file location
      out -  location of generated class files
      samples - Folder for samples
      src - provider source code
```

4. Run `ant help` for instructions on building and testing this provider.

The provider relies on an ad-hoc identity repository for fetching identity information and has been tested with Oracle SOA Suite. It is not intended for production use without appropriate testing for your environment.

20.10.7.2 Setting Up the Sample Provider

The sample identity provider used in this example is a custom Identity/Authentication provider that uses an RDBMS as the underlying store. It can be used as both an identity provider and an authentication provider.

Note: The sample provider is intended solely for demonstration purposes, and it is not advisable to use this provider in production without exhaustive testing.

These steps are required to set up the sample provider:

1. Implement the User and Role APIs to access the database repository serving as the identity store. This involves:

   a. Building the sample provider. Run `ant help` for instructions.

   b. Creating the identity store schema in the database.

2. Configure the sample provider as the identity store, as shown in Section 20.10.7.3, "Configuring jps-config.xml to Use the Sample Identity Provider".

3. Set up Weblogic Authenticator to use this provider as SQLAuthenticator, as explained in Section 20.10.7.4, "Configuring Oracle WebLogic Server".

20.10.7.3 Configuring jps-config.xml to Use the Sample Identity Provider

To configure `jps-config.xml` to enable the sample identity provider to be used as the identity store:

1. Add a new provider in the service providers list:

```
<serviceProviders>
      ............
     <serviceProvider type="IDENTITY_STORE" name="custom.provider"
class="oracle.security.jps.internal.idstore.generic.GenericIdentityStoreProvide
r">
          <description>Custom IdStore Provider</description>
    </serviceProvider>
</serviceProviders>
```

2. Add the service instance:

```
<serviceInstances>
........
    <serviceInstance name="idstore.custom" provider="custom.provider"
          location="dumb">
       <description>Custom Identity Store Service Instance</description>
       <property name="idstore.type" value="CUSTOM"/>
       <property name="ADF_IM_FACTORY_CLASS"
         value="custom_provider_identityStoreFactoryClassName"/>
       <property name="DB_SERVER_NAME" value="db_server_name"/>
       <property name="DB_SERVER_PORT" value="db_port"/>
       <property name="DB_DATABASE_NAME" value="db_service_name"/>
       <property name="ST_SECURITY_PRINCIPAL" value="user_name"/>
       <property name="ST_SECURITY_CREDENTIALS" value="password"/>
   </serviceInstance>
........
<serviceInstances>
```

Note: custom_provider_identityStoreFactoryClassName for the sample provider is org.sample.providers.db.DBIdentityStoreFactory

3. Ensure that the default `jpsContext` points to the identity store service instance added in Step 2 above:

```
<jpsContext name="default">
    <serviceInstanceRef ref="credstore"/>
    <serviceInstanceRef ref="keystore"/>
    <serviceInstanceRef ref="policystore.xml"/>
    <serviceInstanceRef ref="audit"/>
    <serviceInstanceRef ref="idstore.custom"/>
</jpsContext>
```

4. Add the path of the custom provider jar to the classpath.
5. Restart the server.

20.10.7.4 Configuring Oracle WebLogic Server

To configure Oracle WebLogic Server to use `SQLAuthenticator`:

1. Log in to the Oracle WebLogic Server console. Select **Security Realms**, then **myrealm**, then **Providers**. Click **New** to add a new provider.
2. Enter a name for the provider and select SQLAuthenticator as the authenticator type.
3. On the **Providers** page, click on the newly created authenticator.
4. Set the Control Flag to `SUFFICIENT`. Click **Save**.
5. Set the control flag to sufficient for all authenticators in the list.
6. Click on the "Provider Specific" tab to enter the details for the authenticator server. Enter the DataSource name that was used to create the schema for the provider. Click **Save**.
7. Return to the Providers tab and reorder the providers so that SQLAuthenticator is at the top of the list.

The User and Role SPI Reference

This section contains the User and Role SPI reference (Javadoc), describing each abstract class in the SPI with package name oracle.security.idm.spi. The classes are:

- oracle.security.idm.spi.AbstractUserProfile
- oracle.security.idm.spi.AbstractUserManager
- oracle.security.idm.spi.AbstractUser
- oracle.security.idm.spi.AbstractSubjectParser
- oracle.security.idm.spi.AbstractStoreConfiguration
- oracle.security.idm.spi. AbstractSimpleSearchFilter
- oracle.security.idm.spi.AbstractSearchResponse
- oracle.security.idm.spi.AbstractRoleProfile
- oracle.security.idm.spi.AbstractRoleManager
- oracle.security.idm.spi.AbstractRole
- oracle.security.idm.spi.AbstractIdentityStoreFactory
- oracle.security.idm.spi.AbstractIdentityStore
- oracle.security.idm.spi.AbstractComplexSearchFilter

oracle.security.idm.spi.AbstractUserProfile

This class represents a detailed user profile and enables you to set or obtain attributes of the user profile.

Constructors

```
public AbstractUserProfile()
```

Methods

```
public void setPassword(char[] oldPasswd, char[] newPasswd)
public byte[] getUserCertificate()
public void setUserCertificate(byte[] cert)
public java.lang.String getEmployeeNumber()
public void setEmployeeNumber(String employeeNumber)
public java.lang.String getBusinessPostalAddr()
public void setBusinessPostalAddr(String addr)
public java.lang.String getBusinessPOBox()
public void setBusinessPOBox(String pobox)
public byte[] getJPEGPhoto()
public void setJPEGPhoto(String imgpath)
public java.lang.String getTimeZone()
public void setTimeZone(String zone)
public java.lang.String getDescription()
public void setDescription(String desc)
public java.lang.String getDepartmentNumber()
public void setDepartmentNumber(String departmentnumber)
public java.lang.String getGivenName()
public void setGivenName(String givenname)
public java.lang.String getBusinessEmail()
public void setBusinessEmail(String email)
public java.lang.String getBusinessPager()
public void setBusinessPager(String pager)
public java.lang.String getOrganization()
public void setOrganization(String org)
public void setName(String name)
public java.lang.String getBusinessCity()
public void setBusinessCity(String city)
public java.lang.String getMaidenName()
public void setMaidenName(String maidenname)
public java.lang.String getDepartment()
public void setDepartment(String dept)
public java.lang.String getBusinessFax()
public void setBusinessFax(String fax)
public java.lang.String getUserName()
public void setUserName(String uname)
public java.lang.String getBusinessMobile()
public void setBusinessMobile(String mobile)
public java.lang.String getDateofHire()
public void setDateofHire(String hiredate)
public java.lang.String getTitle()
public void setTitle(String title)
public java.lang.String getNameSuffix()
public void setNameSuffix(String suffix)
public java.lang.String getMiddleName()
public void setMiddleName(String middlename)
public java.lang.String getHomePhone()
public void setHomePhone(String homephone)
public void setDisplayName(String dispname)
public java.lang.String getEmployeeType()
public void setEmployeeType(String emptype)
public java.lang.String getLastName()
public void setLastName(String lastname)
public java.lang.String getDateofBirth()
public void setDateofBirth(String dob)
public java.lang.String getManager()
public void setManager(String manager)
public java.lang.String getBusinessState()
public void setBusinessState(String state)
public java.lang.String getHomeAddress()
public void setHomeAddress(String homeaddr)
public java.lang.String getBusinessStreet()
public void setBusinessStreet(String street)
public java.lang.String getBusinessPostalCode()
public void setBusinessPostalCode(String postalcode)
public java.lang.String getInitials()
public void setInitials(String initials)
public java.lang.String getUserID()
public void setUserID(String userid)
public java.lang.String getFirstName()
public void setFirstName(String firstname)
public java.lang.String getDefaultGroup()
public void setDefaultGroup(String defgroup)
public java.lang.String getOrganiztionalUnit()
public void setOrganizationalUnit(String ouUnit)
public java.lang.String getWirelessAcctNumber()
public void setWirelessAcctNumber(String wirelessacct)
public java.lang.String getBusinessPhone()
public void setBusinessPhone(String phone)
public java.lang.String getBusinessCountry()
public void setBusinessCountry(String country)
public java.lang.String getPreferredLanguage()
public void setPreferredLanguage(String language)
public java.lang.String getUIAccessMode()
public void setUIAccessMode(String accessMode)
public java.lang.Object getPropertyVal(String prop)
public oracle.security.idm.SearchResponse getReportees(boolean direct)
public java.util.List getManagementChain(int max, String upToManagerName, String
upToTitle)
public oracle.security.idm.PropertySet getAllUserProperties()
```

oracle.security.idm.spi.AbstractUserManager

This class represents a user manager and includes basic authentication methods.

Constructors

```
public AbstractUserManager()
```

Methods

```
public oracle.security.idm.User authenticateUser(
   String user_id, String authProperty, char[] passwd)

public oracle.security.idm.User authenticateUser(
   User user, char[] passwd)
```

oracle.security.idm.spi.AbstractUser

This class represents a user.

Constructors

```
public AbstractUser()
```

Methods

None.

oracle.security.idm.spi.AbstractSubjectParser

This abstract class provides a constructor for a subject parser.

Constructors

```
public AbstractSubjectParser()
```

Methods

None

oracle.security.idm.spi.AbstractStoreConfiguration

This abstract class provides a constructor for identity store configuration.

Constructors

```
public AbstractStoreConfiguration()
```

Methods

None

oracle.security.idm.spi. AbstractSimpleSearchFilter

This abstract class represents a simple search filter that can be used to search the identity store. Each simple filter consists of a search attribute, matching operator type, and value. Search results are filtered based on this condition.

This class is abstract as its actual underlying representation (provided by method `@link #getNativeRepresentation()`) is implementation-specific. A service provider can extend this class by setting up a specific implementation of that method.

Constructors

```
public AbstractSimpleSearchFilter(
   String attrname, int type, Object value)
```

Methods

Table 20–6 lists the methods of `AbstractSimpleSearchFilter`.

Table 20–6 Methods of AbstractSimpleSearchFilter

Method	Description
`public void setAttribute(String name)`	Set attribute name. .
`public void setType(int type)`	Set filter type.
`public void setValue(Object value)`	Set attribute value.
`public java.lang.String getAttributeName()`	Retrieve attribute name.
`public java.lang.Object getValue()`	Retrieve attribute value.
`public int getType()`	Retrieve filter type.
`public void setNegate()`	Negate the current NOT state of the search filter. Behaves like a toggle switch.
`public void negate()`	Negate the current NOT state of the search filter. Behaves like a toggle switch.
`public boolean isNegated()`	Return the current NOT state of the search filter. Returns `true` if the NOT operator is set; `false` otherwise.

oracle.security.idm.spi.AbstractSearchResponse

This is an abstract class that represents search response results.

Constructors

```
public AbstractSearchResponse()
```

Methods

None.

oracle.security.idm.spi.AbstractRoleProfile

This class represents the detailed profile of a role.

Constructors

```
public AbstractRoleProfile()
```

Methods

```
public oracle.security.idm.SearchResponse getManagers(
   SearchFilter filter, boolean direct)

public oracle.security.idm.SearchResponse getManagers(
   SearchFilter filter)

public oracle.security.idm.SearchResponse getOwners(
   SearchFilter filter, boolean direct)

public oracle.security.idm.SearchResponse getOwners(
   SearchFilter filter)

public void addManager(
   Principal principal)

public void removeManager(
   Principal principal)

public void addOwner(
   Principal principal)

public void removeOwner(
   Principal principal)

public boolean isOwnedBy(
   Principal principal)

public boolean isManagedBy(
   Principal principal)

public void addOwner(
   User user)

public void removeOwner(
   User user)

public void setDisplayName(
   String displayName)

public void setDescription(
   String discription)

public java.lang.String getDescription()

public oracle.security.idm.Property getProperty(
   String propName)
```

oracle.security.idm.spi.AbstractRoleManager

This class is an abstract representation of a role manager.

Constructors

```
public AbstractRoleManager()
```

Methods

```
public boolean isOwnedBy(
   Role role, Principal principal)

public boolean isManagedBy(
   Role role, Principal principal)

public oracle.security.idm.SearchResponse getOwnedRoles(
   Principal principal, boolean direct)

public oracle.security.idm.SearchResponse getManagedRoles(
   Principal principal, boolean direct)
```

oracle.security.idm.spi.AbstractRole

This class provides a constructor for a role.

Constructors

```
public AbstractRole()
```

Methods

None

oracle.security.idm.spi.AbstractIdentityStoreFactory

This class represents an identity store factory.

See Also: "IdentityStoreFactory" in Table 20–1.

Constructors

```
public AbstractIdentityStoreFactory()
```

Methods

```
public oracle.security.idm.IdentityStore getIdentityStoreInstance()
```

oracle.security.idm.spi.AbstractIdentityStore

This abstract class represents an identity store.

Constructors

```
public AbstractIdentityStore()
```

Methods

```
public oracle.security.idm.RoleManager getRoleManager() public
oracle.security.idm.UserManager getUserManager() public java.util.List
getMandatoryUserPropertyNames() public java.util.List getUserPropertySchema()
```

oracle.security.idm.spi.AbstractComplexSearchFilter

This class represents a complex search filter. This type of search filter is used to combine multiple SearchFilter instances with a single boolean AND or OR operator. Each of the component search filters can itself be a complex filter, enabling you to form nested search filters with a high degree of complexity.

This class is abstract in that its actual underlying representation, provided by the `@link #getNativeRepresentation()` method, is implementation-specific.

A service provider can extend this class by creating a specific implementation of this method.

See Also: "oracle.security.idm.spi. AbstractSimpleSearchFilter"

Constructors

```
public AbstractComplexSearchFilter(SearchFilter[] filters, int oper_type)
```

Methods

Table 20–7 Methods of Complex Search Filter

Method	Description
public void addFilterComponent(SearchFilter filter)	Add the SearchFilter component to this complex filter's list.
public void setNegate()	Negate the current NOT state of the search filter. Behaves like a toggle switch.
public void negate()	Negate the current NOT state of the search filter. Behaves like a toggle switch.
public boolean isNegated()	Return the current NOT state of the search filter. Returns `true` if the NOT operator is set; `false` otherwise.
public int getOperatorType()	Logical operator type which binds together the SearchFilter components.

21

Developing with the Identity Directory API

This chapter explains how to access and work with identity stores using the Identity Directory API.

This chapter includes the following sections:

21.1 About the Identity Directory API

The Identity Directory API allows applications to access identity information (users and other entities) in a uniform and portable manner regardless of the particular underlying identity repository.

The Identity Directory API:

- is flexible
- is fully configurable by clients supporting a variety of identity stores having standard and specific schemas
- supports retrieving and managing users, and groups
- is extensible, supporting new entity types with relationships defined between those entities
- is robust, with high-availability/fail-over support.

The Identity Directory API uses the Identity Governance framework, providing all the benefits of the framework to enable you to control how identity related information, including Personally Identifiable Information (PII), access entitlements, attributes, and other entities are used, stored, and propagated between organizations.

21.1.1 Feature Overview

This section explains the features supported by the Identity Directory API.

Features for User Entities

The following features are supported for users:

- Perform Create, Read, Update, Delete (CRUD) operations on users
- Perform the following actions:
  - get and set user attributes
  - authenticate the user with the identity store's native authentication mechanism
  - get the group to which the user belongs (optionally, including nested groups)
  - make user a member of a group
- Change user password
- Force user password change
- Get and set user state (enable/disable, lockout, password must change)

Features for Group Entities

The following features are supported for groups:

- Perform CRUD operations on groups
- Perform the following actions:
  - get and set attributes
  - get and search for members of a group
  - get the groups to which a groups belongs (optionally, including nested groups)
  - determine if the group is a member of another group
- Support multi-valued attributes
- Support static and dynamic groups

21.2 Listing the Classes

Table 21–1 lists the classes in the Identity Directory API:

Table 21–1 Classes in the Identity Directory API

Class	Description
Capabilities	Contains an entity's capabilities.
CreateOptions	Contains options for entity creation operations.
DeleteOptions	Contains options for entity deletion operations.
Entity	Generic entity class holding the list of attributes of the entity fetched using search or read methods.
EntityAlreadyExistsException	Returned following an attempt to create an existing entity.
EntityCapabilities	
EntityManager	Handles operations like read, create and search of generic entity.
EntityNotFoundException	Returned when requested entity is not found.
EntityNotUniqueException	Returned when the entity is not uniquely defined.
EntityRelationManager	Handles entity relationship operations like read, create, delete, search relationship.
Group	A generic entity class holding the list of members of the group. It also provides methods to modify group membership.

Table 21–1 (Cont.) Classes in the Identity Directory API

Class	Description
GroupManager	Handles operations like creating, deleting, and searching for groups.
IDSException	Handles exceptions.
IDSPrincipal	Contains the principal related to the exception.
IdentityDirectory	Represents a handle to IdentityDirectory for creation of IdentityDirectory instance. The instance provides handles to User, Group, and generic Entity Manager so that operations on the corresponding entities can be performed.
IdentityDirectoryFactory	A factory class for creating IdentityDirectoryService.
IdentityDirectoryInfo	
InvalidAttributesException	Used for exceptions related to invalid entity attributes.
InvalidFilterException	Used for exceptions generated within Identity Beans
ModAttribute	
ModifyOptions	Extends OperationOptions containing options for entity modify operation
OperationNotSupportedException	Used for exceptions generated within Identity Beans
ReadOptions	Extends OperationOptions containing options for entity read operation. Read options include Locale and Requested Attributes settings.
ResultSet	An interface for the object returned by search interaction with paged results.
SearchFilter	Used to construct simple or complex nested search filters for searching the entities
SearchOptions	Extends ReadOptions containing options for entity search operation.
User	Generic class for User entities.
UserCapabilities	Contains user capability attributes.
UserManager	Contains methods for creating, deleting, and searching for users by various criteria.

21.3 Understanding Identity Directory Configuration

The identity directory configuration is a combination of the logical entity configuration and the physical identity store configuration.

The identity directory with logical entity configuration is stored in:

```
DOMAIN_HOME/config/fmwconfig/ids-config.xml
```

The physical identity store configuration for the default identity directory is located at:

```
DOMAIN_HOME/config/fmwconfig/ovd/default
```

The default identity directory uses the same identity store properties (namely host, port, credentials, search base, create base, name attribute, and so on) configured in OPSS (weblogic authenticator or in `jps-config.xml`). For more information, see Section F.2.4, "LDAP Identity Store Properties".

21.4 Working with the Identity Directory API

This section explains how applications can use the Identity Directory API to view and manage identity store data. It contains these sections:

- Getting an Identity Directory API Instance
- Performing CRUD Operations on Users and Groups

21.4.1 Getting an Identity Directory API Instance

To obtain the identity directory handle from the jps-context and to get a directory instance:

```
JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
JpsContext ctx = ctxFactory.getContext();

//find the JPS IdentityStore service instance
IdentityStoreService idstoreService =
ctx.getServiceInstance(IdentityStoreService.class)

//get the Identity Directory instance
oracle.igf.ids.IdentityDirectory ids = idstoreService.getIdentityStore();
```

21.4.2 Performing CRUD Operations on Users and Groups

You can perform Create, Retrieve, Update, and Delete (CRUD) operations on users and groups.

- User Operations
- Group Operations

21.4.2.1 User Operations

Basic CRUD operations on users are as follows:

Create User

```
Principal UserManager.createUser(List<Attribute> attrVals, CreateOptions opts)
```

Get User

```
User UserManager.getUser(Principal principal, ReadOptions opts)
```

Search for User

```
User UserManager.searchUser(String id, ReadOptions opts)
```

Delete User

```
void UserManager.deleteUser(String id, DeleteOtions opts)
```

Update User

```
void UserManager.modify(List<ModAttribute> attrVals, ModifyOptions opts)
```

Retrieve List of Users

```
ResultSet UserManager.searchUsers(SearchFilter filter, SearchOptions opts)
```

21.4.2.2 Group Operations

Basic CRUD operations on groups are as follows:

Create Group

```
Principal GroupManager.createGroup(List<Attribute> attrVals, CreateOptions opts)
```

Get Group

```
User GroupManager.getGroup(Principal principal, ReadOptions opts)
```

Search for Group

```
User GroupManager.searchGroup(String id, ReadOptions opts)
```

Delete Group

```
void GroupManager.deleteGroup(String id, DeleteOtions opts)
```

Modify Group Attributes

```
void GroupManager.modify(List<ModAttribute> attrVals, ModifyOptions opts)
```

Retrieve List of Groups

```
ResultSet GroupManager.searchGroups(SearchFilter filter, SearchOptions opts)
```

21.5 Using the Identity Directory API

This section contains the following examples of using the Identity Directory API:

- Initializing and Obtaining Identity Directory Handle
- Creating a User
- Obtaining a User
- Modifying a User
- Performing a Simple Search for a User
- Performing a Complex Search for Users
- Creating a Group
- Obtaining a Group
- Obtaining a Group Using a Search Filter
- Deleting a Group
- Adding a Member to a Group
- Deleting a Member from a Group

See Also: Section 25.1, "Using OPSS in Java SE Applications," for details about securing Java SE applications with OPSS.

21.5.1 Initializing and Obtaining Identity Directory Handle

To initialize and obtain a handle to the identity directory:

```
/**
 * This is a sample program for initializing Identity Directory Service with the
configuration
 * that is already persisted in IDS config location.
 * Basic User and Group CRUDS are performed using this IDS instance
 */
```

```
import java.util.ArrayList;
import java.util.List;
import java.util.Iterator;
import java.util.Map;

import java.security.Principal;

import oracle.igf.ids.Entity;
import oracle.igf.ids.User;
import oracle.igf.ids.UserManager;
import oracle.igf.ids.Group;
import oracle.igf.ids.GroupManager;
import oracle.igf.ids.config.OperationalConfig;
import oracle.igf.ids.IdentityDirectoryFactory;
import oracle.igf.ids.IdentityDirectoryInfo;
import oracle.igf.ids.IdentityDirectory;
import oracle.igf.ids.IDSException;
import oracle.igf.ids.ReadOptions;
import oracle.igf.ids.CreateOptions;
import oracle.igf.ids.ModifyOptions;
import oracle.igf.ids.DeleteOptions;
import oracle.igf.ids.SearchOptions;
import oracle.igf.ids.SearchFilter;
import oracle.igf.ids.ResultSet;
import oracle.igf.ids.Attribute;
import oracle.igf.ids.ModAttribute;

import oracle.dms.context.ExecutionContext;


public class Ids1Test {

    private IdentityDirectory ids;
    private UserManager uMgr;
    private GroupManager gMgr;


    /**
     * Get Identity Store Service
     */
    public Ids1Test() throws IDSException {

        // Set Operational Config
        OperationalConfig opConfig = new OperationalConfig();

        // Set the application credentials: this overrides the credentials
        // set in physical ID store configuration
        //opConfig.setApplicationUser("cn=test_user1,l=amer,dc=example,dc=com");
        //opConfig.setApplicationPassword("welcome123".toCharArray());

        // Set search/crate base, name, objclass, etc. config.
        // This overrides default operational configuration in IDS
        opConfig.setEntityProperty("User", opConfig.SEARCH_BASE,
         "l=amer,dc=example,dc=com");
        opConfig.setEntityProperty("User", opConfig.CREATE_BASE,
         "l=amer,dc=example,dc=com");
        opConfig.setEntityProperty("User", opConfig.FILTER_OBJCLASSES, "person");
        opConfig.setEntityProperty("User", opConfig.CREATE_OBJCLASSES,
         "inetorgperson");
        opConfig.setEntityProperty("Group", opConfig.SEARCH_BASE,
         "cn=dlcontainerOCS,dc=example,dc=com");
        opConfig.setEntityProperty("Group", opConfig.CREATE_BASE,
         "cn=dlcontainerOCS,dc=example,dc=com");
        opConfig.setEntityProperty("Group", opConfig.FILTER_OBJCLASSES,
         "groupofuniquenames");
        opConfig.setEntityProperty("Group", opConfig.CREATE_OBJCLASSES,
         "groupofuniquenames,orclgroup");

        // Get IdentityDirectoryService "userrole" configured in IDS config
        IdentityDirectoryFactory factory = new IdentityDirectoryFactory();
        //ids = factory.getDefaultIdentityDirectory(opConfig);
        ids = factory.getIdentityDirectory("userrole", opConfig);

        // Get UserManager and GroupManager handles
        uMgr = ids.getUserManager();
        gMgr = ids.getGroupManager();
    }
```

21.5.2 Creating a User

This sample code creates a user in the identity store:

```
public Principal createUser() {
       Principal principal = null;

       List<Attribute> attrs = new ArrayList<Attribute>();
       attrs.add(new Attribute("commonname", "test1_user1"));
       attrs.add(new Attribute("password", "welcome123".toCharArray()));
       attrs.add(new Attribute("firstname", "test1"));
       attrs.add(new Attribute("lastname", "user1"));
       attrs.add(new Attribute("mail", "test1.user1@example.com"));
       attrs.add(new Attribute("telephone", "1 650 123 0001"));
       attrs.add(new Attribute("title", "Senior Director"));
       attrs.add(new Attribute("uid", "tuser1"));
       // Adding locale specific value
       attrs.add(new Attribute("description", "created test user 1",
        new java.util.Locale("us", "en")));
       try {
           CreateOptions createOpts = new CreateOptions();
           createOpts.setCreateBase("l=apac,dc=example,dc=com");

           principal = uMgr.createUser(attrs, createOpts);

           System.out.println("Created user " + principal.getName());

       } catch (Exception e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
       }

       return principal;
   }
```

21.5.3 Obtaining a User

This sample code obtains a user from the identity store:

```
public User getUser(Principal principal) {
       User user = null;
```

```
        try {
            ReadOptions readOpts = new ReadOptions();
            // Getting specific locale values
            readOpts.setLocale("us-en");

            user = uMgr.getUser(principal, readOpts);

            printEntity(user);

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        return user;
    }
```

21.5.4 Modifying a User

This sample code modifies an existing user by adding a new user attribute:

```
public void modifyUser(User user) {

        try {
            ModifyOptions modifyOpts = new ModifyOptions();

            List<ModAttribute> attrs = new ArrayList<ModAttribute>();
            attrs.add(new ModAttribute("description", "modified test user 1"));
            //attrs.add(new ModAttribute("uid", "testuser1"));

            user.modify(attrs, modifyOpts);

            System.out.println("Modified user " + user.getName());

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
```

21.5.5 Performing a Simple Search for a User

This sample code performs a basic user search:

```
try {
            ReadOptions readOpts = new ReadOptions();
            readOpts.setSearchBase("l=apac");

            User user = uMgr.searchUser("tuser1", readOpts);

            printEntity(user);

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
```

21.5.6 Performing a Complex Search for Users

This sample code uses a complex search filter to return matching users:

```
public void searchUsers() {

   try {
      // Complex search filter with nested AND and OR conditiions
      SearchFilter filter = new SearchFilter(
         SearchFilter.LogicalOp.OR,
         new SearchFilter(SearchFilter.LogicalOp.AND,
         new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH, "ve"),
         new SearchFilter("telephone", SearchFilter.Operator.CONTAINS, "506")),
         new SearchFilter(SearchFilter.LogicalOp.AND,
         new SearchFilter("firstname", SearchFilter.Operator.BEGINS_WITH, "ra"),
         new SearchFilter(SearchFilter.LogicalOp.OR,
         new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH, "ldap"),
         new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH, "sun"),
         new SearchFilter("orgunit", SearchFilter.Operator.BEGINS_WITH,
          "access")),
         new SearchFilter("telephone", SearchFilter.Operator.CONTAINS, "506")));

      // Requesting attributes
      List<String> reqAttrs = new ArrayList<String>();
      reqAttrs.add("jpegphoto");

      SearchOptions searchOpts = new SearchOptions();
      searchOpts.setPageSize(3);
      searchOpts.setRequestedPage(1);
      searchOpts.setRequestedAttrs(reqAttrs);
      searchOpts.setSearchBase("l=amer");

      ResultSet<User> sr = uMgr.searchUsers(filter, searchOpts);
         while (sr.hasMore()) {
            User user = sr.getNext();
            //printEntity(user);
            //System.out.println(" ");
            System.out.println(user.getSubjectName());
            System.out.println("    " + user.getAttributeValue("commonname"));
         }

      } catch (Exception e) {
         System.out.println(e.getMessage());
         e.printStackTrace();
      }
    }
```

21.5.7 Creating a Group

This sample code creates a group:

```
 public Principal createGroup() {
        Principal principal = null;

        List<Attribute> attrs = new ArrayList<Attribute>();
        attrs.add(new Attribute("name", "test1_group1"));
        attrs.add(new Attribute("description", "created test group 1"));
        attrs.add(new Attribute("displayname", "test1 group1"));
        try {
            CreateOptions createOpts = new CreateOptions();
            principal = gMgr.createGroup(attrs, createOpts);

            System.out.println("Created group " + principal.getName());

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        return principal;
    }
```

21.5.8 Obtaining a Group

This sample code returns a specific group:

```
public Group getGroup(Principal principal) {
        Group group = null;

        try {
            ReadOptions readOpts = new ReadOptions();

            group = gMgr.getGroup(principal, readOpts);

            printEntity(group);

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }

        return group;
    }
```

21.5.9 Obtaining a Group Using a Search Filter

This sample code uses a search filter to return groups:

```
public void searchGroups() {

        try {
            SearchFilter filter = new SearchFilter("name",
                                SearchFilter.Operator.BEGINS_WITH, "test");

            SearchOptions searchOpts = new SearchOptions();
            searchOpts.setPageSize(10);

            ResultSet<Group> sr = gMgr.searchGroups(filter, searchOpts);
            while (sr.hasMore()) {
                Group group = sr.getNext();
                System.out.println(group.getSubjectName());
            }

        } catch (Exception e) {
            System.out.println(e.getMessage());
            e.printStackTrace();
        }
    }
```

21.5.10 Deleting a Group

This sample code deletes a group from the store:

```
public void deleteGroup(Principal principal) {

       try {
           DeleteOptions deleteOpts = new DeleteOptions();

           gMgr.deleteGroup(principal, deleteOpts);

           System.out.println("Deleted group " + principal.getName());

       } catch (Exception e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
       }
   }
```

21.5.11 Adding a Member to a Group

This sample code adds a member to a group:

```
public void addMember() {
       try {
           ReadOptions readOpts = new ReadOptions();
           User user = uMgr.searchUser("testuser1", readOpts);
           Group group = gMgr.searchGroup("test1_group1", readOpts);

           ModifyOptions modOpts = new ModifyOptions();
           user.addMemberOf(group, modOpts);

           System.out.println("added testuser1 as member of test1_group1");

       } catch (Exception e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
       }
   }
```

21.5.12 Deleting a Member from a Group

This sample code deletes a member from a group:

```
public void deleteMember() {
       try {
           ReadOptions readOpts = new ReadOptions();
           User user = uMgr.searchUser("testuser1", readOpts);
           Group group = gMgr.searchGroup("test1_group1", readOpts);

           ModifyOptions modOpts = new ModifyOptions();
           group.deleteMember(user, modOpts);

           System.out.println("deleted testuser1 from the group test1_group1");

       } catch (Exception e) {
           System.out.println(e.getMessage());
```

```
                e.printStackTrace();
            }
        }
```

21.6 Configuring SSL Using Identity Directory API

For details about SSL configuration when using the Identity Directory API, see Section 8.5, "Configuring SSL for the Identity Store Service".

22

Developing with the Keystore Service

This chapter explains how to utilize the Keystore Service when developing applications.

Beginning with a survey of the Keystore Service API, the chapter describes what you need to know when invoking the service in your applications; explains how to configure the service; and presents examples of common keystore operations and ways to retrieve keys at runtime.

This chapter contains the following topics:

22.1 About the Keystore Service API

A keystore is used for secure storage of and access to keys and certificates. The Keystore Service API is used to access and perform operations on the keystores.

The Keystore Service:

- enables you to manage keys and certificates securely
- provides an API for storage, retrieval, and maintenance of keys and certificates in different back-end repositories
- supports file-, database-, LDAP-based keystore management

Critical (create, update, delete) functions provided by the Keystore Service API include:

- creating keystores
- deleting keystores
- obtaining a handle to the domain trust store
- obtaining a handle to a keystore
- obtaining the configured properties for a keystore

- obtaining a list of the keystores within an application stripe

Operations on a `KeyStore` are secured by `KeyStoreAccessPermission`, which implements the fine-grained access control model utilized by the Keystore Service.

> **See Also:**
>
> - Chapter 12, "Managing Keys and Certificates with the Keystore Service".

22.2 Overview of Application Development with the Keystore Service

Knowledge of the following areas is helpful in getting your applications to work with the Keystore Service:

- Determining appropriate application stripe and keystore names to use.
- Provisioning Java security policies.

  Policy permissions are set in the policy store, which can be file-based (`system-jazn-data.xml`) or LDAP-based. Setting appropriate permissions to enable application usage without compromising the security of your data requires careful consideration of permission settings.

  > **See Also:** Section 10.2, "Managing the Policy Store".

- Defining the Keystore Service instance in `jps-config.xml`. Here is an example:

  ```
  <serviceInstancelocation="" name="keystore" provider="keystore.provider">
  <description>Default JPS Keystore Service</description>
  <property name="keystore.provider.type" value="file"/>
  <property name="keystore.file.path" value="./"/>
  <property name="keystore.type" value="KSS"/>
  <property name="keystore.csf.map" value="oracle.wsm.security"/>
  <property name="keystore.sig.csf.key" value="<alias>"/>
  <property name="keystore.enc.csf.key" value="<alias>"/>
  </serviceInstance>
  ```

  You will need to define the service instance in `jps-config.xml` only if manually crafting the configuration file.

  > **Note:** The file-based provider is already configured by default, and can be changed to an LDAP-based provider. See Section 9.6, "Migrating the OPSS Security Store".

- Steps to take in setting up the environment.

  The steps are different for stand-alone applications and those that operate in an Oracle WebLogic Server environment.

22.3 Setting the Java Security Policy Permission

The Oracle Platform Security Services keystore provider is set when the server is started. When the keystore provider is file-based, the data is stored in `system-jazn-data.xml`.

Keystore Service supports securing keys in the following ways:

- at the application stripe level
- at the keystore level
- with finer granularity for specific `<application stripe, keystore, key>`

Notes:

- To properly access the Keystore Service APIs, you need to grant Java permissions in the policy store.
- The code invoking Keystore Service APIs needs code source permission. The permissions are typically for specific code jars and not for the complete application.

This section provides guidelines for permission grants to keystore objects, along with several examples:

- Guidelines for Granting Permissions
- Permissions Grant Example 1
- Permissions Grant Example 2
- Permissions Grant Example 3

Note: In the examples, the application jar file name is `AppName.jar`.

22.3.1 Guidelines for Granting Permissions

The Keystore Service relies on Java permissions to grant permissions to keystore or key objects. It is highly recommended that only the requisite permissions be granted, and no more.

WARNING: It is risky and inadvisable to grant unnecessary permissions, particularly permissions to all application stripes and/or keystores.

22.3.2 Permissions Grant Example 1

The Keystore Service stores objects in a hierarchy:

```
application stripe -> keystore(s) -> key(s)/certificate(s)
```

See Also: Section 12.1.1 for details about the object hierarchy in the Keystore Service.

This example grants permissions for a specific application stripe and a specific keystore name within that stripe.

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <!-- This is the location of the jar -->
            <!-- as loaded with the run-time -->
            <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
            <permission>
    <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission</class>
               <name>stripeName=keystoreapp,keystoreName=ks1,alias=*</name>
              <!-- All actions are granted -->
              <actions>*</actions>
            </permission>
        </permissions>
    </grant>
</jazn-policy>
```

where:

- `stripeName` is the name of the application stripe (typically the name of the application) for which you want to grant these permissions (read, write, update, and delete permissions denoted by the wildcarded actions).
- `keystoreName` is the keystore name in use.
- `alias` indicates the key alias within the keystore.

Note: The wildcard indicates the application is granted permission for all aliases.

22.3.3 Permissions Grant Example 2

In this example, permissions are granted for a specific application stripe name and all its keystores.

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
           <permission>
    <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission</class>
              <name>stripeName=keystoreapp,keystoreName=*,alias=*</name>
              <actions>read,write,update,delete</actions>
        </permission>
        </permissions>
    </grant>
</jazn-policy>
```

22.3.4 Permissions Grant Example 3

In this example, read permissions are granted for a specific key alias within an application stripe name and a keystore.

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
           <permission>
 <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission</class>
              <name>stripeName=keystoreapp,keystoreName=ks1,alias=orakey</name>
              <actions>read</actions>
        </permission>
        </permissions>
    </grant>
</jazn-policy>
```

22.4 Configuring the Keystore Service

Pre-defined configuration files containing a Keystore Service instance with the location of the keystore and the provider classes are available in:

```
$DOMAIN_HOME/config/fmwconfig
```

The configuration files are ready to use and are named as follows:

- `jps-config.xml` used by Java EE applications
- `jps-config-jse.xml` used by Java SE applications

See Also: Section 25.1, "Using OPSS in Java SE Applications".

22.5 Using the Keystore Service API

You can use the Keystore Service within Oracle WebLogic Server or in a standalone environment.

- Using the Keystore Service API in Java SE Applications
- Using the Keystore Service API in Java EE Applications

22.5.1 Using the Keystore Service API in Java SE Applications

See Also: Section 25.1, "Using OPSS in Java SE Applications".

To use the keystore service API in Java SE applications, proceed as follows:

1. Set up the classpath. Ensure that the `jps-manifest.jar` file is in your classpath. For details, see Required JAR in Classpath in Section 1.5.3, "Scenario 3: Securing a Java SE Application".
2. Set up the policy; to provide access to the Keystore Service APIs, you need to configure the access permissions in the reference policy store. For examples, see Section 22.3, "Setting the Java Security Policy Permission".
3. Set JVM command-options. See details below.
4. Run the application.

Command-line options include:

`-Doracle.security.jps.config`
specifies the full path to the configuration file `jps-config-jse.xml`, typically located in the directory *domain_home*`/config/fmwconfig.`

`-Djava.security.policy`
specifies the location of the OPSS/Oracle WebLogic Server policy file `weblogic.policy`, typically located in the directory *wl_home*`/server/lib`.

`-Dcommon.components.home`
specifies the location of the oracle_common directory under middleware home.

`-Dopss.version`
specifies the version used in the environment; its value is 12.1.3.

`-Djava.security.debug=all`
is helpful for debugging purposes. Refer to the Java SE documentation on Oracle Technology Network at `http://www.oracle.com/technetwork` for details about this property.

22.5.2 Using the Keystore Service API in Java EE Applications

To use the API in Java EE applications, proceed as follows:

1. Out-of-the-box, the keystore service provider section of the `jps-config.xml` file is configured in the following directory:

   ```
   $DOMAIN_HOME/config/fmwconfig
   ```

   If needed, reassociate to an LDAP or database store.

2. Set up the policy. To provide access to the Keystore Service APIs, you need to configure the access permissions in the reference policy store. For examples, see Section 22.3, "Setting the Java Security Policy Permission".

3. Start Oracle WebLogic Server.

4. Deploy and test the application.

22.6 Example of Keystore Service API Usage

This section provides an example of using the Keystore Service APIs. It contains these topics:

- Java Program for Keystore Service Management Operations
- Reading Keys at Runtime
- Policy Store Setup
- Configuration File
- About Using the Keystore Service in the Java SE Environment

See Also: Section 25.1, "Using OPSS in Java SE Applications".

22.6.1 Java Program for Keystore Service Management Operations

The following Java code demonstrates common Keystore Service operations:

```
import oracle.security.jps.JpsContext;
import oracle.security.jps.JpsContextFactory;
import oracle.security.jps.JpsException;
import oracle.security.jps.service.keystore.KeyStoreProperties;
import oracle.security.jps.service.keystore.KeyStoreService;
import oracle.security.jps.service.keystore.KeyStoreServiceException;

import java.security.AccessController;
import java.security.PrivilegedAction;
```

```
public class KeyStoreTest {

    private static KeyStoreService ks = null;


    public KeyStoreTest() {
        super();
    }

    /*
     * This method performs a non-privileged operation. Either all code
     * in the call stack must have KeyStoreAccessPermission
     * OR
     * the caller must have the KeyStoreAccessPermission only and
     * invoke this operation in doPrivileged block
     */
    public static void doKeyStoreOperation() {
        doOperation();
    }

     /*
     * Since this method performs a privileged operation, only current class or
     * jar containing this class needs KeyStoreAccessPermission
     */
    public static void doPrivilegedKeyStoreOperation() {
        AccessController.doPrivileged(new PrivilegedAction<String>() {
            public String run() {
                doOperation();
                return "done";
            }
        });
    }

    private static void doOperation() {
        try {
            ks.deleteKeyStore("keystoreapp", "ks1", null);
        } catch(KeyStoreServiceException e) {
            e.printStackTrace();
        }

 public static void main(String args[]) throws Exception {

        try {
            new JpsStartup().start();
            JpsContext ctx = JpsContextFactory.getContextFactory().getContext();
            ks = ctx.getServiceInstance(KeyStoreService.class);

            // #1 - this call is in a doPrivileged block
            // #1 - this should succeed.
            doPrivilegedKeyStoreOperation();

            // #2 - this will also pass since granted all application
            // code necessary permission
            // NOTE: Since this call is not in a doPrivileged block,
            // this call would have failed if KeyStoreAccessPermission
            // wasn't granted to this class.

            /*
            doKeyStoreOperation();
```

```
                */

            } catch (JpsException je) {
                je.printStackTrace();
            }

        }
    }
```

See Also:

- Java Keystore API documentation at http://docs.oracle.com/javase/7/docs/api/index.html
- OPSS `KeyStoreService` API in *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services*

22.6.2 Reading Keys at Runtime

This section explains ways in which an application can access keystore artifacts at runtime, in the following topics:

- Getting the Keystore Handle
- Accessing Keystore Artifacts - Method 1
- Accessing Keystore Artifacts - Method 2

22.6.2.1 Getting the Keystore Handle

To access a keystore at runtime and retrieve keys and certificates, an application must first get the keystore handle from the Keystore Service.

There are two approaches to this task:

1. Obtain the handle by explicitly specifying the application stripe and keystore name as separate parameters
2. Obtain the handle by specifying the application stripe and the keystore name as one parameter

Applications are free to use either method. Both methods work exactly in the same way for fetching keys and certificates, the only difference is the way the key store is loaded. Method 1 uses OPSS APIs while method 2 uses JDK KeyStore APIs.

22.6.2.2 Accessing Keystore Artifacts - Method 1

This example demonstrates the first method, in which the application stripe and keystore name are separate parameters.

```
// First get the KeyStoreService handle from JpsContext

JpsContext ctx = JpsContextFactory.getContextFactory().getContext();
KeyStoreService kss = ctx.getServiceInstance(KeyStoreService.class);

// Now get the keystore handle from KeyStoreService. There are two ways to
// do this:
// Method 1: Explicitly specify application stripe and keystore name as
// separate parameters. In this example, the last parameter is null since
// we are demonstrating a permission-protected keystore.

java.security.KeyStore keyStore = kss.getKeyStore("keystoreapp", "ks1", null);
```

```
// If this keystore is password-protected, provide the keystore password as the
// last parameter.
// java.security.KeyStore.ProtectionParameter pwd = new

java.security.KeyStore.PasswordProtection("password".toCharArray());

// java.security.KeyStore keyStore = kss.getKeyStore("keystoreapp", "ks1", pwd);


// Method 2: Specify application stripe and keystore name as one parameter
// using KSS URI. The last parameter is the keystore password, which is
// set to null in this example due to permission protected keystore.
// For password-protected keystore, set it to the keystore password.
// Since we are demonstrating Method 1 in this
// example, method 2 is commented out

// java.security.KeyStore keyStore kss.getKeyStore("kss://keystoreapp/ks1", null);

// Now you have a handle to the JDK KeyStore object. Use this handle to get
// key(s) and/or certificate(s)

// Get the private key or secret key associated with this alias. Pass the
// key password as the second parameter. In this case, it is null due to
// permission protected keystore.

Key key = keyStore.getKey("key-alias", null);

// Get the certificate associated with this alias

Certificate cert = keyStore.getCertificate("cert-alias");
```

Note: Although this code fragment does not show this, calls to `load`, `getKey`, `getCert` are all privileged operations, and they should therefore be wrapped in a `doPrivileged()` block.

22.6.2.3 Accessing Keystore Artifacts - Method 2

This example demonstrates a method in which the application stripe and keystore name are provided as one parameter using the JDK Keystore URI.

```
// Create an object with parameters indicating the keystore to be loaded
FarmKeyStoreLoadStoreParameter param = new FarmKeyStoreLoadStoreParameter();
param.setAppStripeName("keystoreapp");
param.setKeyStoreName("ks1");
// The password is set to null in this example since it assumes this is a
// permission protected keystore.
param.setProtectionParameter(null);
// If the keystore is password protected, use the following lines instead
// java.security.KeyStore.ProtectionParameter pwd = new
java.security.KeyStore.PasswordProtection("password".toCharArray());
// param.setProtectionParameter(pwd);

// Initialize the keystore object by setting keystore type and provider
java.security.KeyStore keyStore = KeyStore.getInstance("KSS", new
FarmKeyStoreProvider());

// Load the keystore. There are two ways to do this:
// Method 1: Explicitly specify application stripe and keystore name as
// separate parameters in param object
// keyStore.load(param);

// Method 2: Specify application stripe and keystore name as one parameter
// using KSS URI. Since we demonstrate method 2, in this
// example, method 1 is commented out

FarmKeyStoreLoadStoreParameter param = new FarmKeyStoreLoadStoreParameter();
param.setKssUri("kss://keystoreapp/ks1");
param.setProtectionParameter(null);
keyStore.load(param);

// Now you have a handle to JDK KeyStore object. Use this handle to get
// key(s) and/or certificate(s)

// Get the private key or secret key associated with this alias
Key key = keyStore.getKey("key-alias", null);
// Get the certificate associated with this alias
Certificate cert = keyStore.getCertificate("cert-alias");
```

Note: Although this code fragment does not show it, calls to `load`, `getKey`, `getCert` are all privileged operations, and they should therefore be wrapped in a `doPrivileged()` block.

22.6.3 Policy Store Setup

See Also: Section 25.1, "Using OPSS in Java SE Applications".

For illustration, the example uses an XML-based policy store file (`system-jazn-data.xml`) which has the appropriate permissions needed to access the given keystore from the store. The file defines the permissions for different combinations of application stripe and keystore name. Other combinations, or attempts to access the store beyond the permissions defined here, will be disallowed.

Note: This grant is added to the default policy store in `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml`.

Here the system property `projectsrc.home` is set to point to the directory containing the Java SE application, and `clientApp.jar` is the application jar file which is present in sub-directory dist.

The corresponding policy grant looks like this:

```
<grant>
   <grantee>
      <codesource>
         <url>file:${projectsrc.home}/dist/clientApp.jar</url>
      </codesource>
   </grantee>
   <permissions>
      <permission>
         <class>oracle.security.jps.service.keystore.KeyStoreAccessPermission
         </class>
         <name>stripeName=keystoreapp,keystoreName=ks1,alias=*</name>
         <actions>*</actions>
      </permission>
   </permissions>
</grant>
```

22.6.4 Configuration File

Here is a sample configuration of the keystore in the file (`jps-config-jse.xml`). The `keystore.file.path` property of the keystore service shows the directory containing the `keystores.xml` file:

Note: For the complete configuration file see the default file shipped with the distribution in the directory `$DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml`.

```
<jpsConfig>
 ...
    <serviceInstances>
        <serviceInstance name="keystore_file_instance"
                         provider="keystore_file_provider">
            <property name="keystore.file.path" value="store" />
            <property name="keystore.provider.type" value="file" />
        </serviceInstance>
    </serviceInstances>
 ...
</jpsConfig>
```

22.6.5 About Using the Keystore Service in the Java SE Environment

In the Java SE environment, the following calls are equivalent:

```
KeyStoreService store =
JpsServiceLocator.getServiceLocator().lookup(KeyStoreService.class);
```

and:

```
KeyStoreService store =
JpsContextFactory.getContextFactory().getContext().getServiceInstance
(KeyStoreService.class);
```

See Also: Section 25.1, "Using OPSS in Java SE Applications".

22.7 Best Practices

In a clustered environment and when the store is file-based, use the Keystore Service Mbean API over the Keystore Service API to create, retrieve, update, and delete keys for an application.

If you are simply reading keys, however, either API can be used.

23 Developing with the Audit Service

This chapter explains how applications can use the Oracle Fusion Middleware Audit Framework to provide auditing capabilities.

Oracle Fusion Middleware Audit Framework contains an audit service that enables you to integrate with the audit framework programmatically to log audit events and generate compliance reports using the same capabilities available to Oracle components.

Using the audit service, applications (also known as audit clients) can:

- create event definitions without the use of custom tables
- register with the audit service when you deploy the application
- change event definitions when redeploying the application
- change audit configuration settings at run-time

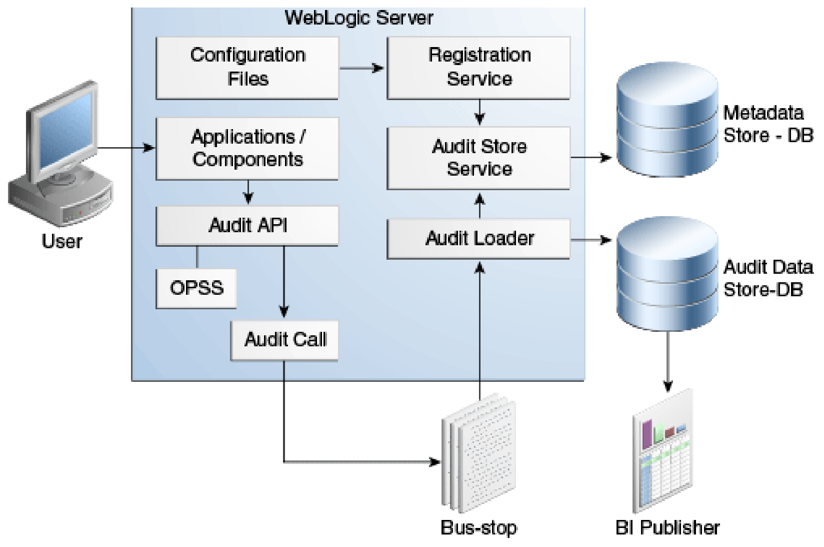
This chapter contains these topics:

See Also: Section 25.1, "Using OPSS in Java SE Applications".

23.1 Understanding How Applications Integrate with Audit Flow

As Figure 23–1 shows, Java EE applications running on Oracle WebLogic Server can integrate with and leverage the audit framework seamlessly:

Figure 23–1 Integrating Applications with the Audit Framework



During application deployment or audit service start-up, a client such as a Java EE application or Oracle component registers with the audit service. The registration service updates the metadata store with the latest audit definitions contained in component_events.xml and related files.

> **See Also:** Section 13.3 for details about the audit flow.

The rest of this chapter explains how you can integrate your applications with the audit framework to log audit events and create audit reports.

23.2 Integrating the Application with the Audit Framework

Take these steps to integrate your application with the audit framework:

1. Complete translation tasks:
   a. Create an audit definition file, `component_events.xml`.
   b. Obtain translations for all the supported languages.
   c. Package the translated files in the `component_events_xlf.jar` file.
2. Package the `component_events.xml` file and `component_events_xlf.jar` in the application `EAR` file.
3. Add the audit event API to the application code to enable it to log audit events.
4. Ensure that the IAU schema and database are set up to upload audit data for reporting.
5. Integrate with a reporting tool such as Oracle Business Intelligence Publisher for reporting.
6. For periodic maintenance, update the audit event definition and redeploy as needed.

The following sections provide more details on these tasks:

- Creating Audit Definition Files
- Registering the Application with the Audit Service

- Adding Application Code to Log Audit Events
- Updating and Maintaining Audit Definitions

> **See Also:** Chapter 7 which describes an alternate, template mode of processing audit artifacts.

23.3 Creating Audit Definition Files

This task involves creating the following files:

- Creating the component-events.xml File
- Creating Translation Files

23.3.1 Creating the component-events.xml File

Create the audit definition component_events.xml file for your application.

For details about the composition of the component_events.xml file, and an example file, see Section 13.5.1.

When creating your audit definition file, you must be aware of certain rules that the registration service uses to create the audit metadata for the application. These include:

- Version numbers. Any change to an audit event definition requires that the version ID be modified by changing the minor and/or major number.

  For more information about versioning, see Section 13.5.3.1.

- Attribute mapping tables for unique mappings between your application's attribute definitions and database columns. Each custom attribute must have a mapping order number in its definition.

  For more information about attribute mapping, see Section 13.5.3.2.

23.3.2 Creating Translation Files

Create the translation files required for your application.

Generate the files in XLIFF format (XML Localisation Interchange File Format) and store them in component_events_xlf.jar; during registration, this information is stored in the audit metadata store along with the component audit event definition.

The steps are as follows:

1. Run a java command similar to the following to generate the XLIFF:

   ```
   java -cp $MW_HOME/oracle_common/modules/oracle.jps_12.1.2/jpsaudit.jar:
   $MW_HOME/oracle_common/modules/oracle.jps_12.1.2/jps-api.jar
   oracle.security.audit.tools.NewXlfGenerator
   -s /tmp/comp_events.xml
   -t /tmp/comp_events.xlf
   ```

2. Translate the generated `xlf` file for the supported languages.

   The generated comp_events.xlf file contains translation units as well as help texts for all categories, events, and attributes. The prefixes of the translation unit id's for each are Category_, Event_ and Attribute_ respectively.

For example, the translation unit id for the CredentialManagement category is Category_CredentialManagement. The source will have the display names and once the file is translated, the target will have the translation for the display name.

3. After translation, package the translated files in a jar file before doing audit registration. For details, see Section 23.4.

23.4 Registering the Application with the Audit Service

This section explains the various mechanisms by which applications can register with the audit service:

- Performing Declarative Audit Registration
- Invoking Registration Programmatically
- Performing Registration at the Command Line
- Using Domain Extension Templates for Audit Artifacts

Note: The use of domain extension templates is the preferred approach to registration; see Section 23.4.4 for details.

23.4.1 Performing Declarative Audit Registration

This section explains the two methods by which declarative registration of audit definitions occurs:

- Default Audit Registration
- Custom Audit Registration

Application Audit Registration

By default, the following registration activity occurs when you deploy, redeploy or undeploy the application:

- Deployment - Registers the audit event definition to the audit metadata store if the application is not yet registered.
- Redeployment - Upgrades the component audit event definition if the component is already registered. See Section 13.5.3.1 for details.
- Undeployment - Removes the application's audit event definition from the audit metadata store.

Custom Application Audit Registration

Custom audit registration parameters are set in the WebLogic deployment descriptor `weblogic-application.xml`. This file is packaged in the META-INF folder of the application EAR files; for information about packaging requirements, see Section 24.5.

Table 23–1 shows the parameters with their options:

Table 23–1 Parameters for Audit Registration Service

Parameter	Option	Description
opss.audit.registration	OVERWRITE	Register component audit definition whether or not it is registered.
	UPGRADE (default option)	Register component audit definition according to versioning support.

Table 23–1 (Cont.) Parameters for Audit Registration Service

Parameter	Option	Description
	DISABLE	Do not register component audit definition.
opss.audit.deregistration	DELETE (default option)	Delete component audit definition from audit store when undeploying applications.
	DISABLE	Keep component audit definition in audit store when undeploying applications.
opss.audit.componentType		Set the custom component type. (Optional)

Examples of weblogic-application.xml

The following sample from a weblogic-application.xml file demonstrates the use of registration and de-registration options:

```
<wls:application-param>
     <wls:param-name>opss.audit.registration</wls:param-name>
     <wls:param-value>DISABLE</wls:param-value>
</wls:application-param>

<wls:application-param>
     <wls:param-name>opss.audit.registration</wls:param-name>
     <wls:param-value>OVERWRITE</wls:param-value>
</wls:application-param>

<wls:application-param>
     <wls:param-name>opss.audit.unregistration</wls:param-name>
     <wls:param-value>DELETE</wls:param-value>
</wls:application-param>

<wls:application-param>
     <wls:param-name>opss.audit.unregistration</wls:param-name>
     <wls:param-value>DISABLE</wls:param-value>
</wls:application-param>
```

23.4.2 Invoking Registration Programmatically

You can register your application to the audit service by invoking audit registration service programmatically. The registration logic is as follows:

```
AuditService auditService =
JpsServiceLocator.getServiceLocator().lookup(AuditService.class);
AuditRegistration auditReg = <instance of AuditRegistration implementation>;
AccessController.doPrivileged(new PrivilegedExceptionAction<Object>() {
            public Object run() throws AuditException, IOException {
                auditService.register(auditReg);
                return null;
            }
        });
```

Invoking auditService.register requires that clients be granted AuditStoreAccessPermission with create, upgrade actions. For details about setting grants, see Section 23.6.2.

Here is an example of registering the audit service programmatically:

```
//
        JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
        JpsContext ctx = ctxFactory.getContext();
        //get audit service
        final AuditService auditService =
ctx.getServiceInstance(AuditService.class);

        //create an audit registration instance
        final AuditRegistration auditReg = new SampleRegistration();

        //call audit service API to register audit event definition
        AccessController.doPrivileged(new PrivilegedExceptionAction<Object>() {
            public Object run() throws AuditException, IOException {
                auditService.register(auditReg);
                return null;
            }
        });
```

23.4.3 Performing Registration at the Command Line

You can register your application with the audit service at the command line through WLST. For details, see registerAudit in the *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

23.4.4 Using Domain Extension Templates for Audit Artifacts

When a domain is created or extended, applications can specify the seeding of application-specific security artifacts such as audit configuration artifacts in the domain security store, as explained in Section 7.1.

Java EE applications can register with the audit service by means of domain extension templates containing the `component_events.xml` and `component_events_xlf.jar` files. The audit registration service will process them automatically when the application is deployed. For details, see Section 7.4.

Note: The filenames *must* be `component_events.xml` and `component_events_xlf.jar` respectively.

23.5 Managing Audit Policies with the Administration Service APIs

The audit framework's Audit Administration Service (admin service) provides a set of APIs to query audit metadata and get/set audit runtime policy. The admin service is invoked as follows:

```
AuditService auditService =
JpsContextFactory.getContextFactory().getContext().getServiceInstance(AuditService
.class);
    AuditAdminService auditAdminService = auditService.getAdminService();
```

This section explains how to use the admin service APIs.

- Querying Audit Metadata
- Viewing and Setting Audit Run-time Policy

23.5.1 Querying Audit Metadata

The APIs to query audit metadata are as follows:

```
Set<String> getComponentNames() throws AuditException;

    Map<String, ? extends AttributeGroup> getGenericAttributeGroups() throws
AuditException;

    Collection<? extends EventCategory> getSystemEvents() throws AuditException;

    ComponentDefinition getComponentDef(String componentType) throws
AuditException;

    AttributesDatabaseMap getAttributesMap(String componentType) throws
AuditException;

    AttributesDatabaseMap getSystemAttributesMap() throws AuditException;
```

The following example demonstrates how to use the APIs to query audit metadata:

```
 //search events and attributes for a component
    Set<String> components = auditAdminService.getComponentNames();
    for (String componentType : components) {
        ComponentDefinition componentDef =
auditAdminService.getComponentDef(componentType);
        //get attributes of a component
        AttributeGroup attrGroup = componentDef.getAttributes();
        for (AuditAttribute attr : attrGroup.getAttributes()) {
            AuditAttribute.DataType type = attr.getAttributeType();
            String attrName = attr.getName();
        }

        //get events of a component
        Collection<? extends EventCategory> events = componentDef.getEvents();
        for (EventCategory category : events) {
            if (category.isComponentSpecific()) { // isComponentSpecific() is true
means the category is belong to a component, otherwise is system category
                Collection<? extends Event> categoryEvents =
category.getAllEvents();

            }
        }
    }
```

23.5.2 Viewing and Setting Audit Run-time Policy

The APIs to retrieve and set audit runtime policy are as follows:

```
AuditPolicy getAuditPolicy(String componentType) throws AuditException;

    void setAuditPolicy(String componentType, AuditPolicy auditPolicy) throws
AuditException;
```

The following example shows how to use the APIs to first obtain and then set the audit policy for a component:

```
 //get runtime policy for component JPS
    final String componentType = "JPS";
    AuditPolicy policy = auditAdminService.getAuditPolicy(componentType);
    String filterLevel = policy.getFilterLevel();
    //set runtim policy for component JPS
    final AuditPolicy newPolicy = new AuditPolicy("None", null, null);
    //setAuditPolicy() requires AuditStoreAccessPermission(<componentType>,
"modify");
    AccessController.doPrivileged(new PrivilegedExceptionAction<Object>(){
        public Object run() throws AuditException {
            auditAdminService.setAuditPolicy(componentType, newPolicy);
            return null;
        }
    });
```

23.6 Adding Application Code to Log Audit Events

Applications can programmatically access the run-time audit service to generate their own audit events using the client API.

23.6.1 Using the Audit Client API

The audit client API is as follows:

> **See Also:** Section 23.6.2 for the prerequisite to using `getAuditor`.

```
Interface AuditService {

Auditor getAuditor(String componentType);

void register(AuditRegistration auditRegistration);

void unregister(AuditRegistration auditRegistration);

}



Interface Auditor {

boolean log(AuditEvent ev);

boolean isEnabled();

boolean isEnabled(String categoryName, String eventType, boolean eventStatus,
Map<String, Object> properties);

}

public class oracle.security.jps.service.audit.AuditEvent {

   public AuditEvent(AuditContext ctx, String eventType,
    String eventCategory, boolean eventStatus, String messageText);
   public void setApplicationName(String applicationName)
   public void setAttribute(String attributeName, Object attributeValue)
   public void setAttributeBoolean(String attributeName, boolean attributeValue)
   public void setAttributeBoolean(String attributeName, Boolean attributeValue)
   public void setAttributeBooleans(String attributeName, boolean[] values)
   public void setAttributeByteArray(String attributeName, byte[] attributeValue)
   public void setAttributeDate(String attributeName, Date attributeValue)
   public void setAttributeDates(String attributeName, Date[] values)
```

```
  public void setAttributeDouble(String attributeName, double attributeValue)
  public void setAttributeDoubles(String attributeName, double[] values)
  public void setAttributeFloat(String attributeName, float attributeValue)
  public void setAttributeFloats(String attributeName, float[] values)
  public void setAttributeInt(String attributeName, int attributeValue)
  public void setAttributeInts(String attributeName, int[] values)
  public void setAttributeLong(String attributeName, long attributeValue)
  public void setAttributeLongs(String attributeName, long[] values)
  public void setAttributeString(String attributeName, String attributeValue)
  public void setAttributeStrings(String attributeName, String[] values)
  public void setComponentName(String componentName)
  public void setComponentType(String componentType)
  public void setContextFields(String contextFields)
  public void setECID(String ecid)
  public void setEventCategory(String category)
  public void setEventDefinition(Object eventDefinition)
  public void setEventStatus(boolean status)
  public void setEventTimestamp(long eventTimestamp)
  public void setEventType(String eventType)
  public void setFailureCode(String failureCode)
  public void setHostId(String hostId)
  public void setHostNetworkAddr(String hostNetworkAddr)
  public void setInitiator(String initiator)
  public void setInstanceId(String instanceId)
  public void setMessageText(String messageText)
  public void setModuleId(String moduleId)
  public void setOracleHome(String oracleHome)
  public void setOracleInstance(String oracleInstance)
  public void setProcessId(String processId)
  public void setRemoteIP(String value)
  public void setResource(String value)
  public void setRID(String rid)
  public void setRoles(String value)
  public void setTarget(String value)
  public void setTargetComponentType(String targetComponentType)
  public void setThreadId(String threadId)
  public void setTransactionId(String transactionId)
}
```

Subsequent sections explain how to obtain permissions and a run-time auditor instance.

23.6.2 Setting System Grants

Audit clients must have AuditStoreAccessPermission to invoke the Audit API. In this example, the grant allows application `MyApp` to call `auditService.getAuditor("MyApp")` in `AccessController.doPrivileged` block:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${oracle.deployed.app.dir}/MyApp${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
 <permissions>
  <permission>
    <class>oracle.security.jps.service.audit.AuditStoreAccessPermission</class>
    <name>comp1</name>
    <actions>action1,action2,….</actions>
  </permission>
 </permissions>
</grant>
```

where the <actions> are a comma-separated list of authorized actions relating to the audit API that the audit client needs to invoke. The possible actions are as follows:

Action	Authorizes client to
create	Call AuditService.register(AuditRegistration) to register a new audit component.
upgrade	Call AuditService.register(AuditRegistration) to upgrade an existing audit component event definition.
delete	Call AuditService.unregister(AuditRegistration) to de-register an audit component.
read	Call AuditService.getAuditor(String componentType) to get an instance of component auditor.
modify	Call AuditAdminService.setAuditPolicy(String componentType, AuditPolicy auditPolicy) to modify audit policy.

23.6.3 Obtaining the Auditor Instance

After your application registers to the audit service, it can get its runtime auditor instance programmatically from the OPSS audit service, as shown in the following sample code fragment:

```
//Gets audit service instance

final AuditService auditService =
JpsServiceLocator.getServiceLocator().lookup(AuditService.class);

//Gets Auditor instance for application 'MyApp'
Auditor auditor = AccessController.doPrivileged(
                            new PrivilegedExceptionAction<Auditor>() {
                                public Auditor run() throws AuditException {
                                    return auditService.getAuditor("MyApp");
                                }
                            });

final String category = "Transaction";
final String eventName = "deposit";

//Check if event 'deposit' is enabled in filtering.
boolean enabled = auditor.isEnabled(category, eventName, "true", null);
if (enabled) {
        AuditContext ctx = new AuditContext();
        String message = "deposit transaction";
        //Creates an audit event
        AuditEvent ev = new AuditEvent(ctx, eventName, category, "true", message);

        //Sets event attributes
        ev.setInitiator("johnsmith");
        ev.setAttributeInt("accounting:AccountNumber", 2134567);
        ev.setAttributeDate("accounting:Date", new Date());
        ev.setAttributeFloat("accounting:Amount", 100.00);

        //Logs audit event
        boolean ret = auditor.log(event);
}
```

23.7 Updating and Maintaining Audit Definitions

As the application's audit requirements evolve, you can update the integration to reflect the changes. The steps are as follows:

1. Update the audit definition file. Be mindful of the versioning rules as you take this step.

   See Also: Section 13.5.3 for a discussion of versioning rules.

2. Redeploy the application EAR file with the updated event definition file. Alternatively, you can notify the audit registration service of the existence of a newer version.
3. Verify your changes.

Note: When creating or extending a WebLogic domain, you can specify audit artifacts in the domain template to facilitate audit definition upgrades. For details, see Section 7.4.

24 Configuring Java EE Applications to Use OPSS

This chapter describes the manual configuration and packaging recommended for Java EE applications that use OPSS but do not use Oracle ADF security. Nevertheless, some topics apply also to Oracle ADF applications. The intended audience are developers that want to configure and package a Java EE application outside Oracle JDeveloper environment. In addition, it describes the API you can use to execute as an asserted user.

For information about Java SE applications, see Chapter 25, "Configuring Java SE Applications to Use OPSS."

This chapter is divided into the following sections:

The files relevant to application management during development, deployment, runtime, and post-deployment are the following:

- `DOMAIN_HOME/config/fmwconfig/jps-config.xml`
- `DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml`
- `jazn-data.xml` (in application EAR file)
- `cwallet.sso` (in application EAR file)
- `web.xml` (in application EAR file)
- `weblogic-application.xml` (in application EAR file)

24.1 Authenticating Java EE Applications

The following documents contain information about developing authentication in Java EE applications:

- For general information about authentication in the Oracle WebLogic Server, see section Authentication in chapter 3 in *Understanding Security for Oracle WebLogic Server*.

- *Developing Applications with the WebLogic Security Service*
  - Chapter 3, Securing Web Applications
  - Chapter 4, Using JAAS Authentication in Java Clients
  - Chapter 5, Using SSL Authentication in Java Clients
- *Developing Security Providers for Oracle WebLogic Server*
  - Chapter 4, Authentication Providers
  - Chapter 5, Identity Assertion Providers
  - Chapter 13, Servlet Authentication Filters
- Custom modules in Java EE applications required to be wrapped in an authenticator provider. For details, see section How to Develop a Custom Authentication Provider in *Developing Security Providers for Oracle WebLogic Server*.
- For login modules and programmatic user authentication used in Java EE applications, see the following topics:
  - Programming Authentication in Java EE Applications
  - Section Login Modules in chapter 4 in *Developing Security Providers for Oracle WebLogic Server*.
  - Section JAAS Authentication Development Environment in Chapter 4 in *Developing Applications with the WebLogic Security Service*.
- For links to all OPSS API Javadocs, see Section G.1, "OPSS API References."

24.2 Programming Authentication in Java EE Applications

The OPSS login modules support programmatic user authentication and assertion in Java EE and SE applications, and the API used to invoke these login modules is common to both Java EE and SE applications. The Oracle WebLogic Server delegates user authentication and assertion to the Oracle WebLogic Server security providers. For details about developing programmatic user authentication with login modules in Java EE applications, see Section 25.3.3, "Login Modules."

24.3 Configuring the Servlet Filter and the EJB Interceptor

OPSS provides a servlet filter, the `JpsFilter`, and an EJB interceptor, the `JpsInterceptor`. The first one is configured in the file `web.xml` packed in a WAR file; the second one is configured in the file `ejb-jar.xml` packed in a JAR file. OPSS also provides a way to configure, in the file `web.xml`, the stripe that application Mbeans should access; for details, see Configuring the Application Stripe for Application MBeans.

On WebLogic, the JpsFilter is out-of-the-box automatically set with default parameter values and need not be explicitly configured in the deployment descriptor; it needs to be configured manually only if a value different from the default value is required. The JpsInterceptor must be manually configured.

Note: Oracle JDeveloper automatically inserts the required servlet filter (`JpsFilter`) and EJB interceptor (`JpsInterceptor`) configurations for Oracle ADF applications.

The manual configurations explained in this section are required *only* if you are packaging or configuring a Java EE application using the OPSS features detailed next *outside* the Oracle JDeveloper environment.

OPSS allows the specification of the application stripe used by MBeans; for details, see Configuring the Application Stripe for Application MBeans.

The servlet filter and the EJB interceptor can be configured using the same set of parameters to customize the following features of a servlet or of an Enterprise Java Bean (EJB):

- Application Name (Stripe)
- Application Roles Support
- Anonymous User and Anonymous Role Support
- Authenticated Role Support
- JAAS Mode

The application name, better referred to as the *application stripe* and optionally specified in the application `web.xml` file, is used at runtime to determine which set of policies are applicable. If the application stripe is not specified, it defaults to the application id (which includes the application name).

An application stripe defines a subset of policies in the policy store. An application wanting to use that subset of policies would define its application stripe with a string identical to that application name. In this way, different applications can use the same subset of policies in the policy store.

The function of the anonymous and authenticated roles is explained in sections About the Anonymous User and Role and About the Authenticated Role.

A servlet specifies the use a filter with the element `<filter-mapping>`. There must be one such element per filter per servlet.

An EJB specifies the use of an interceptor with the element `<interceptor-binding>`. There must be one such element per interceptor per EJB. For more details, see Interceptor Configuration Syntax.

For a summary of the available parameters, see Summary of Filter and Interceptor Parameters.

Application Name (Stripe)

This value is controlled by the following parameter:

```
application.name
```

The specification of this parameter is optional and case sensitive; if unspecified, it defaults to the name of the deployed application. Its value defines the subset of policies in the policy store that the application intents to use.

One way of specifying the application stripe is withing the filter element, as illustrated in the following sample:

```
<filter>
```

```
 <filter-name>JpsFilter</filter-name>
 <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
 <init-param>
  <param-name>application.name</param-name>
  <param-value>stripeid</param-value>
 </init-param>
</filter>
```

Another way to specify it, is to specify it is within the context-param element as illustrated in the following sample:

```
<context-param>
 <description>JPS custom stripe id</description>
 <param-name>application.name</param-name>
 <param-value>stripeid</param-value>
</context-param>
```

This last configuration is required if the application contains MBeans accesssing the application policy store and the application name is different from the application stripe name. For details, see Configuring the Application Stripe for Application MBeans.

Configuration Examples

The following two samples illustrate the configuration of this parameter for a servlet and for an EJB.

The following fragment of a `web.xml` file shows how to configure two different servlets, `MyServlet1` and `MyServlet2`, to be enabled with the filter so that subsequent authorization checks evaluate correctly. Note that servlets in the same WAR file always use the same policy stripe.

```
<filter>
   <filter-name>JpsFilter</filter-name>
   <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
   <init-param>
      <param-name>application.name</param-name>
      <param-value>MyAppName</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>MyServlet1</servlet-name>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
<filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>MyServlet2</servlet-name>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

The following fragment of an `ejb-jar.xml` file illustrates the setting of the application stripe of an interceptor to `MyAppName` and the use of that interceptor by the EJB `MyEjb`:

```
<interceptor>
  <interceptor-class>oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
  <env-entry>
     <env-entry-name>application.name</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>MyAppName</env-entry-value>
```

```
      <injection-target>
        <injection-target-class>
          oracle.security.jps.ee.ejb.JpsInterceptor</injection-target-class>
        <injection-target-name>application_name</injection-target-name>
      </injection-target>
   </env-entry>
</interceptor>
...
<interceptor-binding>
   <ejb-name>MyEjb</ejb-name>
   <interceptor-class>
     oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
</interceptor-binding>
```

Note how the preceding example satisfies the interceptor configuration syntax requirements.

Application Roles Support

The addition of application roles to a subject is controlled by the following parameter, which can be set to true or false:

```
add.application.roles
```

To add application roles to a subject, set the property to true; otherwise, set it to false. The default value is true.

The principal class for the application role is:

```
oracle.security.jps.service.policystore.ApplicationRole
```

Anonymous User and Anonymous Role Support

The use of anonymous for a servlet is controlled by the following parameters, which can be set to true or false:

```
enable.anonymous
remove.anonymous.role
```

For an EJB, only the second parameter above is available, since the use of the anonymous user and role is always enabled for EJBs.

To enable the use of the anonymous user for a servlet, set the first property to true; to disable it, set it to false. The default value is true.

To remove the anonymous role from a subject, set the second property to true; to retain it, set it to false. The default value is false. Typically, one would want to remove the anonymous user and role after authentication, and only in special circumstances would want to retain them after authentication.

The default name and the principal class for the anonymous user are:

```
anonymous
oracle.security.jps.internal.core.principals.JpsAnonymousUserImpl
```

The default name and the principal class for the anonymous role are:

```
anonymous-role
oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl
```

The following fragment of a `web.xml` file illustrates a setting of these parameters and the use of the filter `JpsFilter` by the servlet `MyServlet`:

```
<filter>
```

```
   <filter-name>JpsFilter</filter-name>
   <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
   <init-param>
      <param-name>enable.anonymous</param-name>
      <param-value>true</param-value>
   </init-param>
   <init-param>
      <param-name>remove.anonymous.role</param-name>
      <param-value>false</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>MyServlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
 </filter-mapping>
```

The following fragment of an `ejb-jar.xml` file illustrates the setting of the second parameter to false and the use of the interceptor by the Enterprise Java Bean `MyEjb`:

```
<interceptor>
  <interceptor-class>oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
  <env-entry>
     <env-entry-name>remove.anonymous.role</env-entry-name>
     <env-entry-type>java.lang.Boolean</env-entry-type>
     <env-entry-value>false</env-entry-value>
     <injection-target>
       <injection-target-class>
          oracle.security.jps.ee.ejb.JpsInterceptor</injection-target-class>
       <injection-target-name>remove_anonymous_role/injection-target-name>
     </injection-target>
  </env-entry>
</interceptor>
...
<interceptor-binding>
   <ejb-name>MyEjb</ejb-name>
   <interceptor-class>
        oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
</interceptor-binding>
```

The following fragments illustrate how to access programmatically the anonymous subject, and the anonymous role and anonymous user from a subject:

```
import oracle.security.jps.util.SubjectUtil;

// The next call returns the anonymous subject
javax.security.auth.Subject subj = SubjectUtil.getAnonymousSubject();

// The next call extracts the anonymous role from the subject
java.security.Principal p =
SubjectUtil.getAnonymousRole(javax.security.auth.Subject subj)
// Remove or retain anonymous role
...

// The next call extracts the anonymous user from the subject
java.security.Principal p =
SubjectUtil.getAnonymousUser(javax.security.auth.Subject subj)
// Remove or retain anonymous user
...
```

Authenticated Role Support

The use of the authenticated role is controlled by the following parameter, which can be set to true or false:

```
add.authenticated.role
```

To add the authenticated role to a subject, set the parameter to true; otherwise it, set it to false. The default value is true.

The default name and the principal class for the authenticated role are:

```
authenticated-role
oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl
```

The following fragment of a `web.xml` file illustrates a setting of this parameter and the use of the filter `JpsFilter` by the servlet `MyServlet`:

```
<filter>
   <filter-name>JpsFilter</filter-name>
   <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
   <init-param>
      <param-name>add.authenticated.role</param-name>
      <param-value>false</param-value>
   </init-param>
</filter>
<filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>MyServlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

JAAS Mode

The use of JAAS mode is controlled by the following parameter:

```
oracle.security.jps.jaas.mode
```

This parameter can be set to:

```
doAs
doAsPrivileged
off
undefined
subjectOnly
```

The default value is `doAsPrivileged`. For details on how these values control the behavior of the method `checkPermission`, see Section 17.3.3.1, "Using the Method checkPermission."

The following two samples illustrate configurations of a servlet and an EJB that use this parameter.

The following fragment of a `web.xml` file illustrates a setting of this parameter and the use of the filter `JpsFilter` by the servlet `MyServlet`:

```
<filter>
   <filter-name>JpsFilter</filter-name>
   <filter-class>oracle.security.jps.ee.http.JpsFilter</filter-class>
   <init-param>
      <param-name>oracle.security.jps.jaas.mode</param-name>
      <param-value>doAs</param-value>
   </init-param>
</filter>
```

```
<filter-mapping>
   <filter-name>JpsFilter</filter-name>
   <servlet-name>MyServlet</servlet-name>
   <dispatcher>REQUEST</dispatcher>
</filter-mapping>
```

The following fragment of an `ejb-jar.xml` file illustrates a setting of this parameter to `doAs` and the use of the interceptor `JpsInterceptor` by the Enterprise Java Bean `MyEjb`:

```
<interceptor>
  <interceptor-class>oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
  <env-entry>
     <env-entry-name>oracle.security.jps.jaas.mode</env-entry-name>
     <env-entry-type>java.lang.String</env-entry-type>
     <env-entry-value>doAs</env-entry-value>
     <injection-target>
       <injection-target-class>
          oracle.security.jps.ee.ejb.JpsInterceptor</injection-target-class>
       <injection-target-name>oracle_security_jps_jaas_mode
                </injection-target-name>
     </injection-target>
  </env-entry>
</interceptor>
...
<interceptor-binding>
   <ejb-name>MyEjb</ejb-name>
   <interceptor-class>
        oracle.security.jps.ee.ejb.JpsInterceptor</interceptor-class>
</interceptor-binding>
```

24.3.1 Interceptor Configuration Syntax

The following requirements and characteristics of the specifications apply to all parameters configured for the `JpsInterceptor`:

- The setting of a parameter requires specifying its type (in the element `<env-entry-type>`).
- The setting of a parameter requires the element `<injection-target>`, which specifies the same class as that of the interceptor (in the element `<injection-target-class>`), and the parameter name rewritten as a string where the dots are replaced by underscores (in the element `<injection-target-name>`).
- The binding of an interceptor to an EJB is specified by the EJB name and the interceptor's class, that is, the interceptor is referred to by its class, not by name.

24.3.2 Summary of Filter and Interceptor Parameters

The following table summarizes the description of the parameters used by the JpsFilter and the JpsInterceptor:

Table 24–1 Summary of JpsFilter and JpsInterceptor Parameters

Parameter Name	Values	Default	Function	Notes
application.name	Any valid string. The value is case sensitive.	The name of the deployed application.	To specify the subset of policies that the servlet or EJB is to use.	It should be specified if several servlets or EJBs are to share the same subset of policies in the policy store.
add.application.roles	TRUE or FALSE	TRUE	To add application roles to a Subject.	Since it defaults to TRUE, it must be set (to FALSE) only if the application is not to add application roles to a Subject.
enable.anonymous	TRUE or FALSE	TRUE	To enable or disable the anonymous user in a Subject.	If set to TRUE, it creates a Subject with the anonymous user and the anonymous role.
remove.anonymous.role	TRUE or FALSE	FALSE	To keep or remove the anonymous role from a Subject after authentication.	Available for servlets only. For EJBs, the anonymous role is always removed from a Subject. If set to FALSE, the Subject retains the anonymous role after authentication; if set to TRUE, it is removed after authentication.
add.authenticated.role	TRUE or FALSE	TRUE	To allow addition of the authenticated role in a Subject.	Since it defaults to TRUE, it needs be set (to FALSE) only if the authenticated role is not be included in a Subject.
oracle.security.jps.jaas.mode	doAsPrivileged doAs off undefined subjectOnly	doAsPrivileged	To set the JAAS mode.	

24.3.3 Configuring the Application Stripe for Application MBeans

If your application satisfies the following conditions:

- It contains MBeans that access the policy store and perform authorization checks.
- The application stripe name is not equal to the application name.

then, for the MBean to access the application stripe in the domain security store, the stripe name must be specified by the global parameter (or context parameter) `application.name` in the file `web.xml`, as illustrated in the following sample:

```
<context-param>
 <description>JPS custom stripe id</description>
 <param-name>application.name</param-name>
 <param-value>stripeid</param-value>
</context-param>
```

24.4 Choosing the Appropriate Class for Enterprise Groups and Users

Note: If you are using Oracle JDeveloper, the tool chooses the appropriate classes. Therefore, the configuration explained next is only necessary if policies are entered outside the Oracle JDeveloper environment.

The classes specified in members of an application role must be either other application role class or one of the following:

```
weblogic.security.principal.WLSUserImpl
weblogic.security.principal.WLSGroupImpl
```

The following fragment illustrates the use of these classes in the specification of enterprise groups (in bold face).

Important: Application role names are case *insensitive*; for example, `app_operator` in the following sample.

Enterprise user and group names are case *sensitive*; for example, `Developers` in the following sample.

For related information about case, see Section J.5.2, "Failure to Grant or Revoke Permissions - Case Mismatch."

```
<app-role>
  <name>app_monitor</name>
  <display-name>application role monitor</display-name>
  <class>oracle.security.jps.service.policystore.ApplicationRole</class>
  <members>
    <member>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <name>app_operator</name>
    </member>
    <member>
      <class>weblogic.security.principal.WLSGroupImpl</class>
      <name>Developers</name>
    </member>
  </members>
 </app-role>
```

24.5 Packaging a Java EE Application Manually

This section explains the packaging requirements for a servlet or an EJB (using custom policies and credentials) that is to be deployed on WebLogic Application Server.

Application policies are defined in the file `jazn-data.xml`. The *only* supported way to include this file with an application is to package it in the directory `META-INF` of an EAR file.

Servlets are packaged in a WAR file that contains the configuration file `web.xml`; EJBs are packaged in a WAR file that contains the configuration file `ejb-jar.xml`. The WAR file must include the configuration of the filter `JpsFilter` (for servlets) or of the interceptor `JpsInterceptor` (for EJBs) in the corresponding configuration file.

The description that follows considers the packaging of a servlet and the configuration of the `JpsFilter` in the file `web.xml`, but it applies equally to the packaging of an EJB and the configuration of the `JpsInterceptor` in the file `ejb-jar.xml`.

Important: Currently *all* `JpsFilter` configurations in *all* `web.xml` files in an EAR file *must* have the same configuration. Same constrains apply to the `JpsInterceptor`.

For details about the JpsFilter and the JpsInterceptor, see Configuring the Servlet Filter and the EJB Interceptor.

The packaging requirements and assumptions for a Java EE application that wants to use custom policies and credentials are the following:

- The application to be deployed must be packaged in a single EAR file.
- The EAR file must contain exactly one file `META-INF/jazn-data.xml`, where application policies and roles are specified; these apply equally to all components in the EAR file.
- The EAR file may contain one or more WAR files.
- Each WAR or JAR file in the EAR file must contain exactly one `web.xml` (or `ejb-jar.xml`) where the `JpsFilter` (or `JpsInterceptor`) is configured, and such configurations in all EAR files must be identical.
- Component credentials in `cwallet.sso` files can be packaged in the EAR file. These credentials can be migrated to the credential store when the application is deployed with Oracle Enterprise Manager Fusion Middleware Control.

Note: If a component should require a filter configuration different from that of other components, then it must be packaged in a separate EAR file and deployed separately.

24.5.1 Packaging Policies with Application

Application policies are defined in the file `jazn-data.xml`. The *only* supported way to include this file with an application is to package it in the directory `META-INF` of an EAR file. The EAR file may contain zero or more WAR files, but the policies can be specified only in that XML file located in that EAR directory. To specify particular policies for a component in a WAR file, that component must be packaged in a separate EAR file with its own `jazn-data.xml` file as specified above. No other policy package combination is supported in this release, and policy files other than the top `jazn-data.xml` are disregarded.

24.5.2 Packaging Credentials with Application

Application credentials are defined in a file that must be named `cwallet.sso`. The *only* supported way to include this file with an application is to package it in the directory `META-INF` of an EAR file. The EAR file may contain zero or more WAR files, but credentials can be specified only in that `cwallet.sso` file located in that EAR directory. To specify particular credentials for a component in a WAR file, that component must be packaged in a separate EAR file with its own `cwallet.sso` file as specified above. No other credential package combination is supported in this release, and credential files other than the top `cwallet.sso` are disregarded.

24.6 Configuring Applications to Use OPSS

This section describes several configurations that a developer would perform manually for a Java EE application developed outside the Oracle JDeveloper environment, in the following sections:

- Controlling Policy Migration
- Configuring Parameters According to Behavior
- Controlling Credential Migration
- Credential Parameter Configuration According to Behavior
- Using a Wallet-Based Credential Store
- Using Supported Permission Classes
- Specifying Bootstrap Credentials Manually
- Migrating Identities with migrateSecurityStore
- Example of Configuration File jps-config.xml

Note: Use the system property `jps.deployment.handler.disabled` to disable the migration of application policies and credentials for applications deployed on a WebLogic Server.

When this system property is set to TRUE, the migration of policies and credentials at deployment is disabled for *all* applications regardless of the particular application settings in the application file `weblogic-application.xml`.

24.6.1 Controlling Policy Migration

The migration of application policies at deployment is controlled by several parameters configured in the file `META-INF/weblogic-application.xml`.

The parameters that control migration of policies during application deployment or redeployment, and the removal of policies during undeployment are the following:

- Migration
  - jps.policystore.migration
  - jps.apppolicy.idstoreartifact.migration
  - jps.policystore.removal
- Listener
  - JpsApplicationLifecycleListener
- Principal Validation
  - jps.policystore.migration.validate.principal
- Target of Migration (application stripe)
  - jps.policystore.applicationid

The configuration and function of each of the above is explained next.

Notes: Fusion Middleware Control allows setting of most of these parameters when the application is deployed, redeployed, or undeployed. For details, see Section 6.2.1, "Deploying Java EE and Oracle ADF Applications with Fusion Middleware Control."

The configurations explained next need be entered manually *only if* you are not using Fusion Middleware Control to manage your application.

When deploying an application that is using *file-based* stores to a managed server running in a computer different from that where the administration server is running, do not use the lifecycle listener. Otherwise, the data maintained by the managed server and the administration server would not match, and security may not work as expected. Instead of employing the lifecycle listener, use the WLST command `migrateSecurityStore` to migrate application policies and credentials to the domain stores.

The above remark applies *only* when using file-based stores.

---

jps.policystore.migration

This parameter specifies whether the migration should take place, and, when it does, whether it should merge with or overwrite policies in the target store.

On WebLogic, it is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.policystore.migration</wls:param-name>
  <wls:param-value>Option</wls:param-value>
</wls:application-param>
```

Option stands for one of the following value is MERGE, OVERWRITE, or OFF.

Set to OFF to prevent migration; otherwise, set to MERGE to migrate merging with existing policies, or to OVERWRITE to migrate overwriting existing policies. The default value (at deploy) is MERGE.

jps.policystore.applicationid

This parameter specifies the target stripe into which policies are migrated.

On WebLogic, it is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.policystore.applicationid</wls:param-name>
  <wls:param-value>myApplicationStripe</wls:param-value>
</wls:application-param>
```

This parameter's value can be any valid string; if unspecified, Oracle WebLogic Server picks up a stripe name based on the application name and version, namely, *application_name#version*.

The value of this parameter must match the value of `application.name` specified for the JpsServlet (in the file `web.xml)` or for the JpsInterceptor (in the file `ejb-jar.xml`). For details, see Application Name (Stripe).

The value picked from `weblogic-application.xml` is used at deploy time; the value picked from `web.xml` or `ejb-jar.xml` is used at runtime.

JpsApplicationLifecycleListener

This parameter is supported on WebLogic only, and it must be set as illustrated in the following fragment:

```
<wls:listener>
  <wls:listener-class>
    oracle.security.jps.wls.listeners.JpsApplicationLifecycleListener
  </wls:listener-class>
</wls:listener>
```

jps.apppolicy.idstoreartifact.migration

This parameter is supported on WebLogic only, and it specifies whether the policy migration should exclude migrating references to enterprise users or groups, such as application roles grants to enterprise users or groups, and permission grants to enterprise users or groups; thus it allows the migration of *just* application policies and, when enabled, the migration ignores the mapping of application roles to enterprise groups or users.

It is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.apppolicy.idstoreartifact.migration</wls:param-name>
  <wls:param-value>Option</wls:param-value>
</wls:application-param>
```

Option stands for one of the values TRUE or FALSE. Set to FALSE to exclude the migration of artifacts referencing enterprise users or groups; otherwise, set it to TRUE; if unspecified, it defaults to TRUE.

Important: When an application is deployed with this parameter set to **FALSE** (that is, to exclude the migration of non-application specific policies), before the application can be used in the domain, the administrator should perform the mapping of application roles to enterprise groups or users with Fusion Middleware Control or the WebLogic Administration Console.

Note how this setting allows the administrator further control over application roles.

The following examples show fragments of the same `jazn-data.xml` files. This file, packaged in the application EAR file, describes the application authorization policy.

The file `system-jazn-data.xml` represents the domain file-based policy store into which application policies are migrated (and used in the example for simplicity).

It is assumed that the parameter `jps.apppolicy.idstoreartifact.migration` has been set to FALSE.

```
<!-- Example 1: app role applicationDeveloperRole in jazn-data.xml that references
the enterprise group developers -->
<app-role>
<class>weblogic.security.principal.WLSGroupImpl</class>
  <name>applicationDeveloperRole</name>
  <display-name>application role applicationDeveloperRole</display-name>
  <members>
    <member>
      <class>weblogic.security.principal.WLSGroupImpl</class>
      <name>developers</name>
    </member>
  </members>
</app-role>

<!-- app role applicationDeveloperRole in system-jazn-data.xml after migration:
notice how the role developers has been excluded -->
<app-role>
  <name>applicationDeveloperRole</name>
  <display-name>application role applicationDeveloperRole</display-name>
  <guid>CB3633A0D0E811DDBF08952E56E4544A</guid>
  <class>weblogic.security.principal.WLSGroupImpl</class>
</app-role>

<!-- Example 2: app role viewerApplicationRole in jazn-data.xml makes reference
to the anonymous role -->
<app-role>
  <name>viewerApplicationRole</name>
  <display-name>viewerApplicationRole</display-name>
  <class>weblogic.security.principal.WLSGroupImpl</class>
  <members>
    <member>
      <class>
oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl
      </class>
      <name>anonymous-role</name>
    </member>
  </members>
</app-role>

<!-- app role viewerApplicationRole in system-jazn-data.xml after migration:
notice that references to the anonymous role are never excluded -->
<app-role>
  <name>viewerApplicationRole</name>
  <display-name>viewerApplicationRole</display-name>
  <guid>CB3D86A0D0E811DDBF08952E56E4544A</guid>
  <class>weblogic.security.principal.WLSGroupImpl</class>
  <members>
    <member>
      <class>
oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl
      </class>
      <name>anonymous-role</name>
    </member>
  </members>
</app-role>
```

jps.policystore.removal

This parameter specifies whether the removal of policies at undeployment should *not* take place.

On WebLogic, it is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.policystore.removal</wls:param-name>
  <wls:param-value>OFF</wls:param-value>
</wls:application-param>
```

When set, the parameter's value must be OFF. By default, it is not set.

Set to OFF to prevent the removal of policies; if not set, policies are removed.

The above setting should be considered when multiple applications are sharing the same application stripe. The undeploying application would choose not to remove application policies because other applications may be using the common set of policies.

Note: Deciding to set this parameter to OFF for a given application requires knowing, at the time the application is deployed, whether the application stripe is shared by other applications.

---

jps.policystore.migration.validate.principal

This parameter is supported on WebLogic only, and it specifies whether the check for principals in system and application policies at deployment or redeployment should take place.

It is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.policystore.migration.validate.principal</wls:param-name>
  <wls:param-value>TRUE</wls:param-value>
</wls:application-param>
```

When set, the parameter's value must be TRUE or FALSE.

When set to TRUE the system checks the validity of enterprise users and groups: if a principal (in an application or system policy) refers to an enterprise user or group not found in the identity store, a warning is issued. When set to FALSE, the check is skipped.

If not set, the parameter value defaults to FALSE.

Validation errors are logged in the server log, and they do not terminate the operation.

24.6.2 Configuring Parameters According to Behavior

This section describes the settings required to manage application policies with the following behaviors:

- Skiping Migrating Policies
- Migrating Merging Policies
- Migrating Overwriting Policies
- Removing or Not Removing Policies
- Migrating Policies in a Static Deployment

Any value settings other than the ones described in the following sections are not recommended and may lead to unexpected migration behavior. For more details, see Recommendations.

All behaviors can be specified with Fusion Middleware Control when the application is deployed, redeployed, or undeployed with that tool.

24.6.2.1 Skiping Migrating Policies

The following matrix shows the settings that prevent the migration from taking place:

Table 24–2 Settings to Skip Policy Migration

	Valid at deploy or redeploy
JpsApplicationLifecycleListener	Set
jps.policystore.migration	OFF

Typically, you would skip migrating policies when redeploying the application when you want to keep domain policies as they are, but you would migrate policies when deploying the application for the first time.

24.6.2.2 Migrating Merging Policies

The following matrix shows the setting of required and optional parameters that migrates only policies that are not in the target store (optional parameters are enclosed in between brackets):

Table 24–3 Settings to Migrate Merginh Policies

	Valid at deploy or redeploy
JpsApplicationLifecycleListener	Set
jps.policystore.migration	MERGE
[jps.policystore.applicationid]	Set to the appropriate string. Defaults to servlet or EJB name.
[jps.apppolicy.idstoreartifact.migration]	Set to FALSE to exclude migrating policies that reference enterprise artifacts; otherwise set to TRUE. Defaults to TRUE.
[jps.policystore.migration.validate.principal]	Set to TRUE to validate enterprise users and roles in application and system policies. Set to FALSE, otherwise. If unspecified, it defaults to FALSE.

Typically, you would choose migrating policies with merging at redeploy when the policies have changed and you want to add to the existing policies.

24.6.2.3 Migrating Overwriting Policies

The following matrix shows the setting that migrates all policies overwriting matching target policies (optional parameters are enclosed in between brackets):

Table 24–4 Settings to Migrate Overwriting Policies

	Valid at deploy or redeploy
JpsApplicationLifecycleListener	Set
jps.policystore.migration	OVERWRITE
[jps.policystore.migration.validate.principal]	Set to TRUE to validate enterprise users and roles in application and system policies. Set to FALSE, otherwise. If unspecified, it defaults to FALSE.

Typically, you would choose migrating policies with overwriting at redeploy when a new set of policies should replace existing policies. Note that if the optional parameter `jps.policy.migration.validate.principal` is needed, it must be set manually.

24.6.2.4 Removing or Not Removing Policies

Removing application policies at deployment is limited since code source grants in system policies *cannot* be removed. For details, see example in What Gets Removed and What Remains.

The following matrix shows the setting that removes policies at undeployment:

Table 24–5 Settings to Remove Policies

	Valid at undeploy
JpsApplicationLifecycleListener	Set
jps.policystore.removal	Not set (default)

> **Note:** The policies removed at undeploy are determined by the stripe that the application specified at deploy or redeploy. If an application is redeployed with a stripe specification different than the original one, then policies in that stripe (the original) are not removed.

The following matrix shows the setting that *prevents* the removal of application policies at undeployment:

Table 24–6 Settings to Prevent Removing Policies

	Valid at undeploy
JpsApplicationLifecycleListener	Set
jps.policystore.removal	OFF

> **Note:** Deciding to set this parameter to OFF for a given application requires knowing, at the time the application has been deployed, whether the application stripe is shared by other applications.

What Gets Removed and What Remains

Consider the application `myApp`, which has been configured for automatic migration and removal of policies. The following fragment of the application's `jazn-data.xml` file (packed in the application EAR file) illustrates the application policies that are migrated when the application is deployed with Fusion Middleware Control and those that are and are *not* removed when the application is undeployed with Fusion Middleware Control:

```
<jazn-data>
  <policy-store>
    <applications>
    <!-- The contents of the following element application is migrated
         to the element policy-store in domain system-jazn-data.xml;
         when myApp is undeployed with EM, it is removed from domain store -->
      <application>
        <name>myApp</name>
        <app-roles>
          <app-role>
            <class>oracle.security.jps.service.policystore.SomeRole</class>
            <name>applicationDeveloperRole</name>
            <display-name>application role applicationDeveloperRole</display-name>
            <members>
```

```
              <member>
                <class>oracle.security.somePath.JpsXmlEnterpriseRoleImpl</class>
                <name>developers</name>
              </member>
            </members>
          </app-role>
        </app-roles>
        <jazn-policy>
          <grant>
            <grantee>
              <principals>
                <principal>
            <class>oracle.security.jps.service.policystore.ApplicationRole</class>
                  <name>applicationDeveloperRole</name>
                </principal>
              </principals>
            </grantee>
            <permissions>
              <permission>
                <class>oracle.security.jps.JpsPermission</class>
                <name>loadPolicy</name>
              </permission>
            </permissions>
          </grant>
        </jazn-policy>
      </application>
    </applications>
  </policy-store>

  <jazn-policy>
  <!-- The following codebase grant is migrated to the element
       jazn-policy in domain system-jazn-data.xml; when myApp is undeployed
       with EM, it is not removed from domain store -->
    <grant>
      <grantee>
        <codesource>
          <url>file:${domain.home}/servers/${weblogic.Name}/Foo.ear/-</url>
        </codesource>
      </grantee>
      <permissions>
        <permission>
<class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
          <name>context=SYSTEM,mapName=*,keyName=*</name>
          <actions>*</actions>
        </permission>
      </permissions>
    </grant>
  </jazn-policy>
</jazn-data>
```

To summarize: in regards to what gets removed, the important points to remember are the following:

- All data inside the element `<application>` can be automatically removed at undeployment. In case of an LDAP-based policy store, the application scoped authorization policy data nodes get cleaned up.
- All data inside the element `<jazn-policy>` *cannot* be automatically removed at undeployment.

24.6.2.5 Migrating Policies in a Static Deployment

Table 24–7 shows the setting that migrates application policies when the application is statically deployed. The MERGE or OVERWRITE operation takes place only if the application policies do not already exist in the domain.

Table 24–7 Settings to Migrate Policies with Static Deployments

JpsApplicationLifecycleListener	Set
jps.policystore.migration	MERGE or OVERWRITE

Table 24–8 shows the setting that skip the migration of application policies when the application is statically deployed.

Table 24–8 Settings Not to Migrate Policies with Static Deployments

JpsApplicationLifecycleListener	Set
jps.policystore.migration	OFF

24.6.2.6 Recommendations

Keep in mind the following points:

When a LDAP-based policy store is used and the application is to be deployed to multiple managed servers, then choose to migrate to one of the servers only. The rest of the deployments should choose *not* to migrate policies. This ensures that the policies are migrated only once from the application store to the policy store.

All the deployments must use the same application id.

Attempting policy migration to the same node for the same application multiple times (for example, on different managed servers) can result in policy migration failures. An alternative is to migrate the policy data to the store outside of the deployment process using the WLST command `migrateSecurityStore`.

If, however, the application is deployed to several servers and the policy store is file-based, the deployment must include the administration server for the migration to update the policy file `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml`.

24.6.3 Using a Wallet-Based Credential Store

The content of a wallet-based credential store is defined in a file that must be named `cwallet.sso`. A wallet-based credential store is also referred to as a file-based credential store.

For instructions on how to create a wallet, see section Common Wallet Operations in *Administering Oracle Fusion Middleware*.

The location of the file `cwallet.sso` is specified in the configuration file `jps-config.xml` with the element `<serviceInstance>`, as illustrated in the following example:

```
<serviceInstance name="credstore" provider="credstoressp">
   <property name="location"  value="myCredStorePath"/>
</serviceInstance>
```

For other types of credential storage, see chapter Managing Keystores, Wallets, and Certificates in *Administering Oracle Fusion Middleware*.

24.6.4 Controlling Credential Migration

The migration of application credentials at deployment is controlled by several parameters configured in the file `META-INF/weblogic-application.xml`.

The parameter that controls credential migration is jps.credstore.migration. The listener is JpsApplicationLifecycleListener - Credentials.

jps.credstore.migration

This parameter specifies whether the migration should take place, and, when it does, whether it should merge with or overwrite matching credentials present in the target store.

On WebLogic, it is configured as illustrated in the following fragment:

```
<wls:application-param>
  <wls:param-name>jps.credstore.migration</wls:param-name>
  <wls:param-value>behaviorValue</wls:param-value>
</wls:application-param>
```

If set, this parameter's value must be one of the following: MERGE, OVERWRITE, or OFF. The OVERWRITE value is available on WebLogic only and when the server is running in development mode.

If not set, the migration of credentials takes place with the option MERGE.

JpsApplicationLifecycleListener - Credentials

This listener is supported only on WebLogic and it is configured as illustrated in the following fragment:

```
<wls:listener>
  <wls:listener-class>
     oracle.security.jps.wls.listeners.JpsApplicationLifecycleListener
  </wls:listener-class>
</wls:listener>
```

24.6.5 Credential Parameter Configuration According to Behavior

This section describes the manual settings required to migrate application credentials with the following behaviors:

- Skipping Migrating Credentials
- Migrating Merging Credentials
- Migrating Overwriting Credentials

Any value settings other than the ones described in the following sections are not recommended and may lead to unexpected migration behavior.

If the migration target is an LDAP-based credential store, it is recommended that the application be deployed to just one managed server or cluster. Otherwise, application credentials may not work as expected.

Note: Credentials are not deleted upon an application undeployment. A credential may have started its life as being packaged with an application, but when the application is undeployed credentials are *not* removed.

24.6.5.1 Skipping Migrating Credentials

The following matrix shows the setting that prevents the migration from taking place:

Table 24–9 Settings to Skip Credential Migration

	Valid at deploy or redeploy
jps.credstore.migration	OFF

24.6.5.2 Migrating Merging Credentials

The following matrix shows the setting of required and optional parameters that migrates only credentials that are not present in the target store (optional parameters are enclosed in between brackets):

Table 24–10 Settings to Migrate Merging Credentials

	Valid at deploy or redeploy
JpsApplicationLifecycleListener	Set
jps.credstore.migration	MERGE

24.6.5.3 Migrating Overwriting Credentials

This operation is valid on WebLogic only and when the server is running in development mode. The following matrix shows the setting that migrates all credentials overwriting matching target credentials:

Table 24–11 Settings to Migrate Overwriting Credentials

	Valid at deploy or redeploy
JpsApplicationLifecycleListener	Set
jps.credstore.migration	OVERWRITE
jps.app.credential.overwrite.allowed	This system property must be set to TRUE

24.6.6 Using Supported Permission Classes

The components of a codebase grant are illustrated in the following snippet from a `system-jazn-data.xml` file:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
      </class>
      <name>context=SYSTEM</name>
      <actions>getConfiguredApplications</actions>
    </permission>
    <permission>
      <class>
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
      </class>
      <name>context=APPLICATION,name=*</name>
      <actions>getApplicationPolicy</actions>
    </permission>
  </permissions>
</grant>
```

This section describes the supported values for the elements `<class>`, `<name>`, and `<actions>` within a `<permission>`.

Important: All permission classes used in policies must be included in the class path, so the policy provider can load them when a service instance is initialized.

24.6.6.1 Policy Store Permission

Class name:

```
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
```

When the permission applies to a particular application, use the following pattern for the corresponding element `<name>`:

```
context=APPLICATION,name=appStripeName
```

When the permission applies to all applications, use the following name pattern for the corresponding element `<name>`:

```
context=APPLICATION,name=*
```

When the permission applies to all applications and system policies, use the following name pattern for the corresponding element `<name>`:

```
context=SYTEM
```

The list of values allowed in the corresponding element `<actions>` are the following (* stands for any allowed action):

```
*
createPolicy
getConfiguredApplications
getSystemPolicy
getApplicationPolicy
createApplicationPolicy
deleteApplicationPolicy
grant
revoke
createAppRole
alterAppRole
removeAppRole
addPrincipalToAppRole
removePrincipalFromAppRole
hasPermission
containsAppRole
```

24.6.6.2 Credential Store Permission

Class name:

```
oracle.security.jps.service.credstore.CredentialAccessPermission
```

When the permission applies to a particular map and a particular key in that map, use the following pattern for the corresponding element `<name>`:

```
context=SYSTEM,mapName=myMap,keyName=myKey
```

When the permission applies to a particular map and all keys in that map, use the following pattern for the corresponding element `<name>`:

```
context=SYSTEM,mapName=myMap,keyName=*
```

The list of values allowed in the corresponding element `<actions>` are the following (* stands for any allowed action):

```
*
read
write
update
delete
```

24.6.6.3 Generic Permission

Class name:

```
oracle.security.jps.JpsPermission
```

When the permission applies to an assertion performed by a callback instance of `oracle.security.jps.callback.IdentityCallback`, use the following pattern for the corresponding element `<name>`:

```
IdentityAssertion
```

The only value allowed in the corresponding element `<actions>` is the following:

```
execute
```

24.6.7 Specifying Bootstrap Credentials Manually

This topic is for an administrator who is not using Oracle Fusion Middleware Control to perform reassociation to an LDAP-based store.

The credentials needed for an administrator to connect to and access an LDAP directory must be specified in a separate file named `cwallet.sso` (bootstrap credentials) and configured in the file `jps-config.xml`. These credentials are stored after the LDAP reassociation process. Bootstrap credentials are always file-based.

Every instance of an LDAP-based policy or credential store must specify bootstrap credentials in a `<jpsContex>` element that must be named `bootstrap_credstore_context`, as illustrated in the following excerpt:

```
<serviceInstances>
    ...
  <serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.cred">
      <property value="./bootstrap" name="location"/>
  </serviceInstance>
    ...
</serviceInstances>

<jpsContext name="bootstrap_credstore_context">
    <serviceInstanceRef ref="bootstrap.cred"/>
</jpsContext>
```

In the example above, the bootstrap credential `cwallet.sso` is assumed located in the directory `bootstrap`.

An LDAP-based policy or credential store instance references its credentials using the properties `bootstrap.security.principal.key` and `bootstrap.security.principal.map`, as illustrated in the following instance of an LDAP-based policy store:

```
<serviceInstance provider="ldap.policystore.provider" name="policystore.ldap">
  ...
  <property value="bootstrapKey" name="bootstrap.security.principal.key"/>
  ...
</serviceInstance>
```

If the property `bootstrap.security.principal.map` is not specified in the service instance, its value defaults to `BOOTSTRAP_JPS`.

To modify or add bootstrap credentials with WLST commands, see `modifyBootStrapCredential` and `addBootStrapCredential` in *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.

24.6.8 Migrating Identities with migrateSecurityStore

Identity data can be migrated from a source repository to a target LDAP repository using the WLST command `migrateSecurityStore`. For details, see Section 6.6.1.1, "Migrating Identities with migrateSecurityStore.".

24.6.9 Example of Configuration File jps-config.xml

The following sample shows a complete `jps-config.xml` file that illustrates the configuration of several services and properties; they apply to both Java EE and Java SE applications.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<jpsConfig xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd">
 <property value="off" name="oracle.security.jps.jaas.mode"/>
 <propertySets>
  <propertySet name="saml.trusted.issuers.1">
   <property value="www.oracle.com" name="name"/>
  </propertySet>
 </propertySets>

 <serviceProviders>
  <serviceProvider
class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
name="credstoressp" type="CREDENTIAL_STORE">
   <description>SecretStore-based CSF Provider</description>
  </serviceProvider>
  <serviceProvider
class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider"
name="idstore.ldap.provider" type="IDENTITY_STORE">
   <description>LDAP-based IdentityStore Provider</description>
  </serviceProvider>
  <serviceProvider
class="oracle.security.jps.internal.idstore.xml.XmlIdentityStoreProvider"
name="idstore.xml.provider" type="IDENTITY_STORE">
   <description>XML-based IdentityStore Provider</description>
  </serviceProvider>
  <serviceProvider
class="oracle.security.jps.internal.policystore.xml.XmlPolicyStoreProvider"
```

```
name="policystore.xml.provider" type="POLICY_STORE">
   <description>XML-based PolicyStore Provider</description>
  </serviceProvider>
  <serviceProvider
class="oracle.security.jps.internal.login.jaas.JaasLoginServiceProvider"
name="jaas.login.provider" type="LOGIN">
   <description>JaasLoginServiceProvider to conf loginMod servInsts</description>
  </serviceProvider>
  <serviceProvider class="oracle.security.jps.internal.keystore.KeyStoreProvider"
name="keystore.provider" type="KEY_STORE">
   <description>PKI Based Keystore Provider</description>
   <property value="owsm" name="provider.property.name"/>
  </serviceProvider>
  <serviceProvider class="oracle.security.jps.internal.audit.AuditProvider"
name="audit.provider" type="AUDIT">
   <description>Audit Service</description>
  </serviceProvider>
  <serviceProvider
class="oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider"
name="ldap.credentialstore.provider" type="CREDENTIAL_STORE"/>
  <serviceProvider
class="oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"
name="ldap.policystore.provider" type="POLICY_STORE">
   <property value="OID" name="policystore.type"/>
  </serviceProvider>
 </serviceProviders>

 <serviceInstances>
  <serviceInstance location="./" provider="credstoressp" name="credstore">
   <description>File Based Credential Store Service Instance</description>
  </serviceInstance>
  <serviceInstance provider="idstore.ldap.provider" name="idstore.ldap">
   <property
value="oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfigProvider"
name="idstore.config.provider"/>
  </serviceInstance>
  <serviceInstance location="./system-jazn-data.xml"
provider="idstore.xml.provider" name="idstore.xml">
   <description>File Based Identity Store Service Instance</description>
   <property value="jazn.com" name="subscriber.name"/>
  </serviceInstance>
  <serviceInstance location="./system-jazn-data.xml"
provider="policystore.xml.provider" name="policystore.xml">
   <description>File Based Policy Store Service Instance</description>
  </serviceInstance>
  <serviceInstance location="./default-keystore.jks" provider="keystore.provider"
name="keystore">
   <description>Default JPS Keystore Service</description>
   <property value="JKS" name="keystore.type"/>
   <property value="oracle.wsm.security" name="keystore.csf.map"/>
   <property value="keystore-csf-key" name="keystore.pass.csf.key"/>
   <property value="enc-csf-key" name="keystore.sig.csf.key"/>
   <property value="enc-csf-key" name="keystore.enc.csf.key"/>
 </serviceInstance>
 <serviceInstance provider="audit.provider" name="audit">
   <property value="None" name="audit.filterPreset"/>
   <property value="104857600" name="audit.maxFileSize"/>
   <property value="jdbc/AuditDB" name="audit.loader.jndi"/>
   <property value="15" name="audit.loader.interval"/>
   <property value="File" name="audit.loader.repositoryType"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider" name="saml.loginmodule">
   <description>SAML Login Module</description>
   <property
value="oracle.security.jps.internal.jaas.module.saml.JpsSAMLLoginModule"
name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
   <propertySetRef ref="saml.trusted.issuers.1"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider" name="krb5.loginmodule">
   <description>Kerberos Login Module</description>
   <property value="com.sun.security.auth.module.Krb5LoginModule"
name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
   <property value="true" name="storeKey"/>
   <property value="true" name="useKeyTab"/>
   <property value="true" name="doNotPrompt"/>
   <property value="./krb5.keytab" name="keyTab"/>
   <property value="HOST/localhost@EXAMPLE.COM" name="principal"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider"
name="digest.authenticator.loginmodule">
   <description>Digest Authenticator Login Module</description>
 <property
value="oracle.security.jps.internal.jaas.module.digest.DigestLoginModule"
name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider"
name="certificate.authenticator.loginmodule">
  <description>X509 Certificate Login Module</description>
  <property value="oracle.security.jps.internal.jaas.module.x509.X509LoginModule"
name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider" name="wss.digest.loginmodule">
   <description>WSS Digest Login Module</description>
   <property
value="oracle.security.jps.internal.jaas.module.digest.WSSDigestLoginModule"
name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider"
name="user.authentication.loginmodule">
   <description>User Authentication Login Module</description>
   <property
value="oracle.security.jps.internal.jaas.module.authentication.JpsUserAuthenticati
onLoginModule" name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
 </serviceInstance>
 <serviceInstance provider="jaas.login.provider"
name="user.assertion.loginmodule">
   <description>User Assertion Login Module</description>
   <property
value="oracle.security.jps.internal.jaas.module.assertion.JpsUserAssertionLoginMod
ule" name="loginModuleClassName"/>
   <property value="REQUIRED" name="jaas.login.controlFlag"/>
 </serviceInstance>
 <serviceInstance provider="ldap.credentialstore.provider" name="credstore.ldap">
   <property value="bootstrap" name="bootstrap.security.principal.key"/>
   <property value="cn=wls-jrfServer" name="oracle.security.jps.farm.name"/>
   <property value="cn=jpsTestNode" name="oracle.security.jps.ldap.root.name"/>
   <property value="ldap://myHost.com:1234" name="ldap.url"/>
 </serviceInstance>
 <serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.cred">
   <property value="./bootstrap" name="location"/>
 </serviceInstance>
 <serviceInstance provider="ldap.policystore.provider" name="policystore.ldap">
   <property value="OID" name="policystore.type"/>
   <property value="bootstrap" name="bootstrap.security.principal.key"/>
   <property value="cn=wls-jrfServer" name="oracle.security.jps.farm.name"/>
   <property value="cn=jpsTestNode" name="oracle.security.jps.ldap.root.name"/>
   <property value="ldap://myHost.com:1234" name="ldap.url"/>
 </serviceInstance>
</serviceInstances>

<jpsContexts default="default">
 <jpsContext name="default">
   <serviceInstanceRef ref="keystore"/>
   <serviceInstanceRef ref="audit"/>
   <serviceInstanceRef ref="credstore.ldap"/>
   <serviceInstanceRef ref="policystore.ldap"/>
 </jpsContext>
 <jpsContext name="oracle.security.jps.fmw.authenticator.DigestAuthenticator">
   <serviceInstanceRef ref="digest.authenticator.loginmodule"/>
 </jpsContext>
 <jpsContext name="X509CertificateAuthentication">
   <serviceInstanceRef ref="certificate.authenticator.loginmodule"/>
 </jpsContext>
 <jpsContext name="SAML">
   <serviceInstanceRef ref="saml.loginmodule"/>
 </jpsContext>
 <jpsContext name="bootstrap_credstore_context">
   <serviceInstanceRef ref="bootstrap.cred"/>
 </jpsContext>
</jpsContexts>
</jpsConfig>
```

25

Configuring Java SE Applications to Use OPSS

The information in this chapter applies only to Java SE applications, and the intended audience are developers of Java SE applications.

For similar information about Java EE applications, see Chapter 24, "Configuring Java EE Applications to Use OPSS."

This chapter includes in the following topics:

25.1 Using OPSS in Java SE Applications

In order to use OPSS in a Java SE application, the developer should be aware that:

- The file `jps-manifest.jar` must be in the class path.
- The application must call `JpsStartup.start()` at initialization. For details, see The Class JpsStartup.
- The following system properties must be set:
  - `oracle.security.jps.config`, the file path to the file `jps-config-jse.xml`.
  - `common.components.home`, the path to the oracle common directory.
  - `java.security.policy`, the file path to the file `java.policy`.
  - `opss.audit.logDirectory`, the path to writable directory where audit records for JSE are written

  For other (optional) system properties you can set, see Section F.1, "OPSS System Properties."
- The JVM option `-Djava.security.policy` specifies the location of the Oracle WebLogic Server policy file `weblogic.policy`, typically located in the directory `wl_home/server/lib`.
- The application should use the OPSS class `AppSecurityContext`, as illustrated in the following sample snippet:

Note: The sample snippet is not required if you want to develop and use only CSF/KSS functionality.

```
String appID = AppSecurityContext.getApplicationID();

       try {
            setApplicationID(appName);
            .....
        } finally {
            setApplicationID(appID );
        }
    }

    private static void setApplicationID(final String applicationID) {
                AccessController.doPrivileged(new PrivilegedAction() {
            public Object run() {
                AppSecurityContext.setApplicationID(applicationID);
                return null;
            }
        });
    }
```

The call `AppSecurityContext.setApplicationID` requires a codesource permission like the following:

```
<grant>
    <grantee>
        <codesource>
          <url>myJavaSEapp/-</url>
        </codesource>
    </grantee>
    <permissions>
        <permission>
            <class>oracle.security.jps.JpsPermission</class>
            <name>AppSecurityContext.setApplicationID.myAppStripeID</name>
        </permission>
    </permissions>
</grant>
```

For information about how to configure the OPSS security store, see Chapter 9, "Configuring the OPSS Security Store."

25.1.1 The Class JpsStartup

This section describes the following enhancements to the class `JpsStartup`:

- A set of OPSS states for the method `JpsStartup.start()`
- Several run-time options for the method `JpsStartup.start()`
- A new constructor for the class `JpsStartup`
- The method `JpsStartup.getState()`

For complete information about this class, see *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services*. For examples of use, see OPSS Starting Examples.

This section contains the following information:

25.1.1.1 JpsStartup.start States

The transition states of `JpsStartup.start()` are defined by the following constants:

- UNINITIALIZED - The default state before `JpsStartup.start()` is invoked.
- INITIALIZING - The state to which it transitions after `JpsStartup.start()` has been invoked.
- FAILURE - The state to which it transitions if any failure has occurred during the INITIALIZING state.
- ACTIVE - The state to which it transitions if all services have been started without failures, that is, if no failures have occurred in the INITIALIZING state.
- INACTIVE - The state to which it transitions after `JpsStatup.stop()` is invoked.

25.1.1.2 Run-Time Options to JpsStartup

The run-time options with which you can invoke the class constructor `jpsStartup()` are described in the following table:

Table 25–1 JpsStartup Run-Time Options

Option	Values	Default	Description
ACTIVE_CONTEXT	The name of an active context or the string "default".	"default"	If passed, use the specified context; otherwise, use the "default" context.
ENABLE_JAVA_POLICY_PROVIDER	TRUE or FALSE	TRUE	Set to TRUE to use the Java2 policy provider; otherwise, set to FALSE.
ENABLE_AUDIT_SERVICE_EXT	TRUE or FALSE	TRUE	Set to TRUE to start the audit service with audit monitoring and auditloader; otherwise, set to FALSE.
ENABLE_POLICY_STORE_SERVICE_EXT	TRUE or FALSE	TRUE	Set to TRUE to start the Policy Distribution Point (PDP) with policy change scanner thread; otherwise, set to FALSE.
ENABLE_DEPLOYMENT_HANDLING	TRUE or FALSE	TRUE	Set to TRUE to create deployment handlers; otherwise, set to FALSE.
RUNTIME_MODE	DEFAULT or SCRIPT	DEFAULT	Set to SCRIPT to have the options ENABLE_JAVA_POLICY_PROVIDER , ENABLE_AUDIT_SERVICE_EXT, ENABLE_POLICY_STORE_SERVICE_EXT, and ENABLE_DEPLOYMENT_HANDLING set to FALSE; otherwise set to DEFAULT.
JPS_CONFIG	The path to the file jps.config.xml	The path specified by the property oracle.security.jps.config	If passed, used the specified path; otherwise, use the default path.

25.1.1.3 A New Class Constructor

The following constructor has been added to the class:

```
JpsStatup(java.lang.String platformType,
          java.util.Map<java.lang.String,
          ContextConfiguration>  ctxtCfgs,
          java.util.Map<java.lang.String,?> options)
```

The argument `ContextConfiguration` holds the details about the context within <jpsContext> in the configuration file `jps-config-jse.xml`. To obtain ContextConfiguration before calling JpsStartup, see examples 3 and 4 in OPSS Starting Examples.

To start multiple contexts (in addition to the "default" context), pass multiple context information; see example 4 in OPSS Starting Examples. To start some context (other than the "default" context), then pass the context name; if nothing is passed, then the default context is started.

The argument `options` encloses all the startup properties used to start OPSS; for the list of available options, see Table 25–1.

This constructor is typically invoked when OPSS is to be started with additional `ContextConfiguration` details; see example 3 in OPSS Starting Examples.

25.1.1.4 The Method JpsStartup.getState

This method returns the state OPSS is at the point of invocation, that is, it returns one of INITIALIZING, UNINITIALIZED, FAILURE, ACTIVE or INACTIVE.

25.1.1.5 OPSS Starting Examples

The following code examples illustrate the use of the class `JpsStartup` in typical starting scenarios.

Example 1

The following example illustrates how to start OPSS without any explicit parameters.

```
JpsStartup jpsStartUp = new JpsStartup();
jpsStartUp.start();
jpsStartUp.getState();
jpsStartUp.stop();
```

Example 2

The following example illustrates how to start OPSS with the following configuration:

- `default1` is the default context passed as part of the `contextConfig` map.
- To disable audit runtime services, such as the audit loader and the audit monitoring, then pass the parameter "ENABLE_AUDIT_SERVICE_EXT" set to "FALSE"; note that by default this parameter is enabled.
- To disable runtime services, such as policy changes scanner thread, pass the parameter "ENABLE_POLICY_STORE_SERVICE_EXT" set to "FALSE"; note that by default this parameter is enabled.

```
ConfigurationServiceProvider prov = ConfigurationServiceProvider.newInstance();
ConfigurationService configService = prov.getConfigurationService();
ContextConfiguration configuration =
configService.getContextConfiguration("default1");
Map<String, ContextConfiguration> contextConfig = new HashMap<String,
ContextConfiguration>();
```

```
contextConfig.put(JpsConstants.DEFAULT_CONTEXT_KEY, configuration);

Map<String, Object> option = new HashMap<String, Object>();
option.put(JpsConstants.ENABLE_AUDIT_SERVICE_EXT, "FALSE");
option.put(JpsConstants.ENABLE_POLICY_STORE_SERVICE_EXT, "FALSE");

JpsStartup  jpsStartUp = new JpsStartup("JSE", contextConfig, option);
jpsStartUp.start();
jpsStartUp.stop();
```

Example 3

The following example illustrates how to start OPSS with additional `ContextConfigurations` details.

```
Map<String, Object> startupOption = new HashMap<String, Object>();

ConfigurationServiceProvider prov = ConfigurationServiceProvider.newInstance();
ConfigurationService configService = prov.getConfigurationService();
ContextConfiguration configuration =
configService.getContextConfiguration("default1");

Map<String, ContextConfiguration> contextConfig = new HashMap<String,
ContextConfiguration>();
contextConfig.put(JpsConstants.DEFAULT_CONTEXT_KEY, configuration);

JpsStartup jpsStartUp = new JpsStartup("JSE", contextConfig, startupOption);
jpsStartUp.start();
jpsStartUp.stop();
```

Example 4

The following example illustrates how to start OPSS with multiple contexts. It is assumed that the `jps.config.xml` file contains the following contexts:

```
<jpsContexts default="default">
 <jpsContext name="default">
  <serviceInstanceRef ref="credstore"/> ...
 </jpsContext>

 <jpsContext name="default1">
  <serviceInstanceRef ref="idstore.loginmodule"/> ...
 </jpsContext>

 <jpsContext name="default2">
  <serviceInstanceRef ref="keystore"/> ...
 </jpsContext>

 <jpsContext name="default3">
  <serviceInstanceRef ref="policystore "/> ...
 </jpsContext>

 <jpsContext name="bootstrap_credstore_context">
  <serviceInstanceRef ref="bootstrap.credstore"/>
 </jpsContext>
</jpsContexts>

ConfigurationServiceProvider prov = ConfigurationServiceProvider.newInstance();
ConfigurationService configService = prov.getConfigurationService();
ContextConfiguration configuration1 =
configService.getContextConfiguration("default1");
ContextConfiguration configuration2 =
configService.getContextConfiguration("default2");
ContextConfiguration configuration3 =
configService.getContextConfiguration("default3");

Map<String, ContextConfiguration> contextConfig = new HashMap<String,
ContextConfiguration>();

contextConfig.put(JpsConstants.DEFAULT_CONTEXT_KEY, configuration);
contextConfig.put(("default1", configuration1);
contextConfig.put(("default2", configuration2);
contextConfig.put(("default3", configuration3);

Map<String, Object> startupOption = new HashMap<String, Object>();
startupOption.put(JpsConstants.ACTIVE_CONTEXT, "default");

JpsStartup jpsStartUp = new JpsStartup("JSE", contextConfig, startupOption);
jpsStartUp.start();
jpsStartUp.stop();
```

The parameter "ACTIVE_CONTEXT" specifies the context to use as default. If the JpsConstants.DEFAULT_CONTEXT_KEY is not passed as part of map, then by default the "default" context is started *provided* there is no "ACTIVE_CONTEXT" key is passed as part of `startupOption` map.

Example 5

The following example illustrates how to start OPSS in "SCRIPT" mode. This mode is typically used during migration or upgrading, when runtime services are not required to be started.

```
Map<String, Object> startupOption = new HashMap<String, Object>();
startupOption.put(JpsConstants.RUNTIME_MODE, "SCRIPT");

JpsStartup jpsStartUp = new JpsStartup("JSE", startupOption);
jpsStartUp.start();
jpsStartUp.stop();
```

25.2 Implementing Security Services in Java SE Applications

This sections describes, per service, the OPSS support in Java SE applications.

- Authenticating Java SE Applications
- Chapter 18, "Developing with the Authorization Service"
- Section 19.6.1, "Using the CSF API in Java SE Applications"
- Section 22.5.1, "Using the Keystore Service API in Java SE Applications"

25.3 Authenticating Java SE Applications

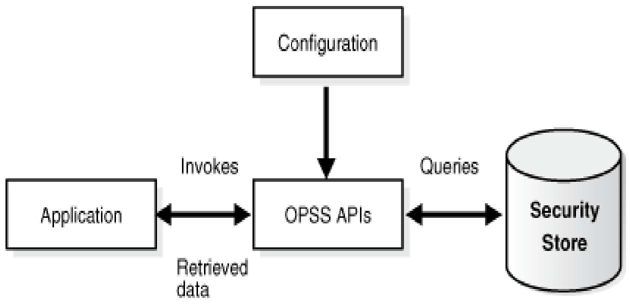
This section explains the identity store support for Java SE applications, and it includes the following sections:

- The Identity Store
- Configuring an LDAP Identity Store in Java SE Applications
- Login Modules
- Using the OPSS API LoginService in Java SE Applications

25.3.1 The Identity Store

Authentication is the mechanism by which callers prove that they are acting on behalf of specific users or system. Using data, such as name-password combinations, authentication answers the question Who are you? The term identity store refers to the storage where identity data is kept, and authentication providers are ways to access an identity store.

An application obtains information from an OPSS security store (identity, policy, or credential store) and manages its contents using the OPSS APIs, as illustrated in the following graphic:



25.3.2 Configuring an LDAP Identity Store in Java SE Applications

A Java SE application can use an LDAP-based identity store configured in the file `jps-config-jse.xml` with the elements `<serviceProvider>`, `<serviceInstance>`, and `<jpsContext>`, as illustrated in the following snippet:

```
<serviceProviders>
  <serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
    <description>Prototype LDAP-based ID store</description>
  </serviceProvider>
</serviceProviders>

<serviceInstances>
 <serviceInstance name="idstore.ldap" provider="idstore.ldap.provider">
   <property name="idstore.type" value="OID"/>
   <property name="max.search.filter.length" value="500"/>
   <extendedProperty>
     <name>user.search.bases</name>
     <values>
       <value>cn=users,dc=us,dc=oracle,dc=com</value>
     </values>
   </extendedProperty>
   <extendedProperty>
     <name>group.search.bases</name>
     <values>
       <value>cn=groups,dc=us,dc=oracle,dc=com</value>
      </values>
   </extendedProperty>
 </serviceInstance>
</serviceInstances>

<jpsContexts default="ldap_idstore">
  <jpsContext name="ldap_idstore">
    <serviceInstanceRef ref="idstore.ldap"/>
  </jpsContext>
  <jpsContext name="bootstrap_credstore_context">
     <serviceInstanceRef ref="bootstrap.cred"/>
   </jpsContext>
</jpsContexts>
```

Note the following points:

- The name of the `<serviceInstance>` (`idstore.ldap` in the example above) can have any value, but it must match the instance referenced in element `<serviceInstanceRef>`.
- The name of the `<serviceProvider>` (`idstore.ldap.provider` in the example above) can have any value, but it must match the provider in element `<serviceInstance>`.
- To add properties to a provider instance with a prescribed script, see Appendix E.1, "Configuring OPSS Service Provider Instances with a Script."

25.3.3 Login Modules

A login module is a component that authenticates users and populates a subject with principals. This process occurs in two distinct phases: during the first phase, the login module attempts to authenticate a user requesting, as necessary, a name and a password or some other credential data; only if this phase succeeds, the second phase is invoked. During the second phase, the login module assigns relevant principals to a subject, which is eventually used to perform some privileged action.

In any of the supported login modules, you can set the following properties:

```
enable.anonymous (default: false)
remove.anonymous.role (default: true)
add.application.roles (default: true)
add.authenticated.role (default: true)
```

For a sample configuration, see "<serviceInstance>"

The supported login modules are the following:

- The Identity Store Login Module
- The User Authentication Login Module
- The User Assertion Login Module
- Executing As an Asserted User

Note 1: The User Authentication login module and the User Assertion login modules can be used in both Java EE and SE applications.

Note 2: In Java SE applications, the User Authentication login module and the User Assertion login modules are supported by the default identity store service.

25.3.3.1 The Identity Store Login Module

A Java SE application can use a stack of login modules to authenticate its users; each module performs its own computations independently from the others in the stack. These and other services are specified in the file `jps-config-jse.xml`.

OPSS APIs includes the interface `oracle.security.jps.service.login.LoginService` which allows a Java SE application to invoke not just all login modules in a stack, but a subset of them in a prescribed order.

The name of the jps context (defined in the configuration file `jps-config-jse.xml`) passed to the method `LoginContext` in the `LoginService` interface determines the stack of login modules that an application uses. The standard JAAS API `LoginContext` can also be used to invoke the login modules defined in the default context

The sequence in which a jps context lists the login modules in a stack is significant, since the authentication algorithm takes this order into account in addition to other data, such as the flag that identifies the module security level (required, sufficient, requisite, or optional).

Out-of-the-box, the identity store service is file-based, its contents being provisioned in the file system-jazn-data.xml; the service can be reconfigured to use an LDAP-based identity store.

OPSS supports the Identity Store login module in Java SE applications, which can be used for authentication or identity assertion, as explained in the following sections:

- Using the Identity Store Login Module for Authentication
- Using the Identity Store Login Module for Assertion

Identity Store Login Module

The class associated with this login module is the following:

```
oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule
```

An instance of this module is configured in the file `jps-config-jse.xml` as illustrated in the following fragment:

```
<serviceInstance name="idstore.loginmodule" provider="jaas.login.provider">
  <description>Identity Store Login Module</description>
  <property name="loginModuleClassName"
value="oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule"/>
  <property name="jaas.login.controlFlag" value="REQUIRED"/>
</serviceInstance>
```

Properties specific to this login module include the following:

```
remove.anonymous.role (defaults to true)
add.application.role (defaults to true)
```

25.3.3.1.1 Using the Identity Store Login Module for Authentication This section illustrates the use of the Identity Store login module for basic username and password authentication.

Invoke IdStoreLoginModule

The following code fragment illustrates how to set a callback handler and a context:

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
```

```
Subject sub = new Subject();
CallbackHandler cbh = new YourCallbackHandler();
LoginContext context = new LoginContext(appName, subject,  cbh);
context.login();
```

The callback handler must be able to handle `NameCallback` and `PasswordCallback`.

Configure jps-config-jse.xml

The following `jps-config-jse.xml` fragment illustrates the configuration of the context `appName`:

```
<jpsContext name="appName">
   <serviceInstanceRef ref="jaaslm.idstore1"/>
</jpsContext>

<serviceProvider type="JAAS_LM" name="jaaslm.idstore"
      class="oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule">
   <description>Identity Store-based LoginModule
   </description>
</serviceProvider>

<serviceInstance name="jaaslm.idstore1" provider="jaaslm.idstore">
   <property name="jaas.login.controlFlag" value="REQUIRED"/>
   <property name="debug" value="true"/>
   <property name="addAllRoles" value="true"/>
</serviceInstance>
```

Write the Callback Handler

The following code snippet illustrates a callback handler able to handle name and password callback:

```
import javax.security.auth.callback.*;
import java.io.IOException;
public class SampleCallbackHandler implements CallbackHandler {
//For name/password callbacks
private String name = null;private char[] password = null;
public SampleCallbackHandler(String name, char[] pwd) {
   if (name == null || name.length() == 0 )
      throw new IllegalArgumentException("Invalid name ");
   else
      this.name = name;
   if (pwd == null || pwd.length == 0)
      throw new IllegalArgumentException("Invalid password ");
   else
      this.password = pwd;
}
public String getName() {
   return name;
   } public char[] getPassword()  {
   return password;
 }
public void handle(Callback[] callbacks)
   throws IOException, UnsupportedCallbackException {
   if (callbacks != null && callbacks.length > 0) {
      for (Callback c : callbacks) {
      if (c instanceof NameCallback) {
            ((NameCallback) c).setName(name);
            }
   else
      if (c instanceof PasswordCallback) {
         ((PasswordCallback) c).setPassword(password);
              }
   else {
      throw new UnsupportedCallbackException(c);
 }
          }
      }
  }
}
```

25.3.3.1.2 Using the Identity Store Login Module for Assertion

To use the Identity Store login module for assertion, a developer must:

- Provide the appropriate permission for the caller to execute the protected method `setIdentity`. This requires granting the permission `oracle.security.jps.JpsPermission` with the name `IdentityAssertion`.
- Implement a callback handler that uses the class `oracle.security.jps.callback.IdentityCallback` as shown in the code sample below.

The above two requirements are illustrated in the following configuration and code samples.

Provisioning the JpsPermission

The following configuration sample illustrates a grant allowing the code `MyApp` the required `JpsPermission` to execute protected methods in the assertion login module:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${soa.oracle.home}/application/myApp.ear</url>
      <--! soa.oracle.home is a system property set when
      the server JVM is started -->
    </codesource>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
    </permission>
  </permissions>
</grant>
```

The following configuration sample illustrates a grant allowing the principal `jdoe` the required `JpsPermission` to execute the assertion login module:

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>weblogic.security.principal.WLSUserImpl</class>
        <name>jdoe</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
    </permission>
  </permissions>
</grant>
```

Implementing the CallbackHandler

The following code fragment illustrates an implementation of the callback handler:

```
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
import javax.security.auth.callback.UnsupportedCallbackException;

import oracle.security.jps.callback.IdentityCallback;

public class CustomCallbackHandler implements CallbackHandler {
    private String name = null;
    private char[] password;

    public CustomCallbackHandler(String name) {
        this.name = name;
    }

    public CustomCallbackHandler(String name, char[] password) {
      this.name  = name;
      this.password = password;
    }

    public void handle(Callback[] callbacks) throws IOException,
                                                UnsupportedCallbackException {
        for (Callback callback : callbacks) {
            if (callback instanceof NameCallback) {
              NameCallback nc = (NameCallback) callback;
              nc.setName(name);
            }
            else if (callback instanceof PasswordCallback) {
              PasswordCallback pc = (PasswordCallback) callback;
              pc.setPassword(password);
            }
            else if (callback instanceof IdentityCallback) {
                IdentityCallback idcb = (IdentityCallback)callback;
                idcb.setIdentity(name);
                idcb.setIdentityAsserted(true);
                idcb.setAuthenticationType("CUSTOM");
            } else {
                //throw exception
                throw new UnsupportedCallbackException(callback);
            }
        }
    }
}
```

The following code fragment illustrates the implementation of a login module:

```
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.login.LoginContext;
import oracle.security.jps.service.JpsServiceLocator;
import oracle.security.jps.service.login.LoginService;

public class LoginModuleExample {
    private static final String CONTEXT_NAME = "JSE_UserAuthnAssertion";

    public LoginModuleExample() {
        super();
    }

    public Subject assertUser(final String username) throws Exception {
        CallbackHandler cbh =
            AccessController.doPrivileged(new
PrivilegedExceptionAction<CallbackHandler>() {
                public CallbackHandler run() throws Exception {
                    return new CustomCallbackHandler(username);
                }
            });
        Subject sub = new Subject();
        LoginService ls =
            JpsServiceLocator.getServiceLocator().lookup(LoginService.class);
        LoginContext context = ls.getLoginContext(sub, cbh);

        context.login();
        Subject s = context.getSubject();

        return s;
    }

    public Subject authenticate(final String username, final char[] password)
throws Exception {
        CallbackHandler cbh = new CustomCallbackHandler(username, password);
        Subject sub = new Subject();
        LoginService ls =
            JpsServiceLocator.getServiceLocator().lookup(LoginService.class);
        LoginContext context = ls.getLoginContext(sub, cbh);

        context.login();
        Subject s = context.getSubject();

        return s;
    }

    public static void main(String[] args) {
        LoginModuleExample loginModuleExample = new LoginModuleExample();
        try {
            System.out.println("authenticated user subject = " +
                               loginModuleExample.authenticate("testUser",
 "welcome1".toCharArray()));
            System.out.println("asserted user subject = " +
                               loginModuleExample.assertUser("testUser"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

25.3.3.2 The User Authentication Login Module

The User Authentication login module can be used by both Java EE and Java SE applications to authenticate a user given a user name and a password. The configuration of this login module is available out-of-the-box. This login module authenticates against the domain identity store.

Here is a code fragment using this module for programmatic authentication:

```
import javax.security.auth.callback.NameCallback;
import javax.security.auth.callback.PasswordCallback;
public class MyCallbackHandler  extends CallbackHandler {
        private String name = null;
        private char[] password = null;

        public MyCallbackHandler(String name, char[] password) {
          this.name = name;
          this.password = password;
        }

        public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
            for (Callback callback : callbacks) {
                if (callback instanceof NameCallback) {
                    NameCallback ncb  = (NameCallback) callback;
                    ncb.setName(name);
                }
                else if (callback instanceof PasswordCallback) {
                    PasswordCallback pcb  = (PasswordCallback) callback;
                    pcb.setPassword(password);
                }
                else {
                    throw UnsupportedCallbackException(callback);
                }
            }
        }
}

JpsLoginModuleFactory factory = JpsLoginModuleFactory.getLoginModuleFactory();
CallbackHandler cbh = new  MyCallbackHandler("name", "password".toCharArray());
LoginContext ctx = factory.getLoginContext(JpsLoginModuleType.USER_AUTHENTICATION,
null, cbh);
ctx.login();
Subject s = ctx.getSubject();
```

25.3.3.3 The User Assertion Login Module

The User Assertion login module can be used by both Java EE and Java SE applications to assert a user identity. This login module asserts against the domain identity store, and it requires implementing the callback `oracle.security.jps.callback.IdentityCallback`. Moreover, execution of this login module requires an IdentityAssertion permission (with execute as action) for `oracle.security.jps.JpsPermission`.

To use the User Assertion login module for programmatic assertion, a developer must:

- Provide the appropriate permission for the caller to execute login module protected methods. This requires granting the permission oracle.security.jps.JpsPermission with name IdentityAssertion (and action execute).

- Implement a callback handler that uses the class `oracle.security.jps.callback.IdentityCallback`.
- Implement the programmatic assertion code.

The above requirements are illustrated in the following configuration and code samples:

- Provisioning the jpsPermission
- Implementing the CallbackHandler
- Implementing the Programmatic Assertion

25.3.3.3.1 Provisioning the jpsPermission The following configuration sample illustrates a grant allowing the principal jdoe the required JpsPermission to execute the User Assertion login module:

```
<grant>
  <grantee>
    <principals>
      <principal>
        <class>weblogic.security.principal.WLSUserImpl</class>
        <name>jdoe</name>
      </principal>
    </principals>
  </grantee>
  <permissions>
    <permission>
      <class>oracle.security.jps.JpsPermission</class>
      <name>IdentityAssertion</name>
      <action>execute</action>
    </permission>
  </permissions>
</grant>
```

For a codesource example, see grant sample in Section 24.6.6, "Using Supported Permission Classes."

25.3.3.3.2 Implementing the CallbackHandler The following code fragment illustrates an implementation of the callback handler:

```
import oracle.security.jps.callback.IdentityCallback;

import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import java.io.IOException;

    public class AssertCallbackHandler implements CallbackHandler {
        private String name = null;

        public AssertCallbackHandler(String name) {
            this.name = name;
        }


        public void handle(Callback[] callbacks) throws IOException,
UnsupportedCallbackException {
            for (Callback callback : callbacks) {
                if (callback instanceof IdentityCallback) {
                    IdentityCallback nc = (IdentityCallback) callback;
                    nc.setIdentity(name);
                }
                else {
                    throw new UnsupportedCallbackException(callback);
                }
            }
        }
    }
```

25.3.3.3.3 Implementing the Programmatic Assertion The following code fragment illustrates how to assert a user:

```
import oracle.security.jps.callback.IdentityCallback;
import oracle.security.jps.internal.api.jaas.module.JpsLoginModuleFactory;
import oracle.security.jps.internal.api.jaas.module.JpsLoginModuleType;

import javax.security.auth.Subject;
import javax.security.auth.callback.Callback;
import javax.security.auth.callback.CallbackHandler;
import javax.security.auth.callback.UnsupportedCallbackException;
import javax.security.auth.login.LoginContext;
import java.io.IOException;
import java.security.AccessController;
import java.security.PrivilegedActionException;
import java.security.PrivilegedExceptionAction;

        Subject subject = AccessController.doPrivileged(new
PrivilegedExceptionAction<Subject>() {
            public Subject run() throws Exception {
                JpsLoginModuleFactory f =
JpsLoginModuleFactory.getLoginModuleFactory();
                CallbackHandler cbh = new AssertCallbackHandler(name);
                LoginContext c = f.getLoginContext(JpsLoginModuleType.USER_
ASSERTION, null, cbh);
                c.login();
                return c.getSubject();
            }
        });
```

25.3.3.4 Executing As an Asserted User

Applications often need to execute actions as if the operation were run by a user, that is, to invoke run-as operations. OPSS solves this problem by allowing Java EE and Java SE applications to use the subjects of an asserted user to execute application logic as if it were executed by that user; this functionality is encapsulated in the OPSS class `SubjectSecurity` and supported on the Oracle WebLogic platform.

This section includes the following topics:

- Use Cases
- Programming Guidelines and Recommendations
- A Code Example

25.3.3.4.1 Use Cases Here are several examples where the class `SubjectSecurity` can be used to solve a programming challenge:

- UC1 - A scheduler allows submitting and executing jobs; these jobs are typically submitted at one time and executed at some future time, therefore the need to assert the user's identity and execute the scheduled job in the user's security context.

- UC2 - A scheduled job is executed by an Entity Bean that, further, may call other Entity Beans to complete the job, and all the code to complete the job must execute in the user's security context; therefore this context needs to be propagated through the call path, across Entity Beans.
- UC3 - The permissions to execute a scheduled job are based on the user's application roles; in this case, the user's security context must be aware of the application roles granted to the user regardless of the code path across Entity Beans.
- UC4 - When an unauthenticated user submits a job, the job metadata needs to persist the anonymous user; eventually, the job must as executed by the anonymous user.
- UC5 - An MBean running in an application is invoked from a UI via an MBean server; the MBean operation invokes a series of diagnostic tests that need to be run as a particular user.
- UC6 - A user identity has been asserted by the user assertion API, and has produced a user security context that can be used to run operations as that user.

25.3.3.4.2 Programming Guidelines and Recommendations The `SubjectSecurity` API allows a user to execute code either when a pre-computed Subject is available (as in use case UC6), or when the user needs to be first asserted. If an application security context has not been set, then, before using the `SubjectSecurity` API, the application must set its id as follows:

```
AppSecurityContext.setApplicationID(applicationID)
```

For example, the application id must be set with `setApplicationID` in use cases UC1, UC2, and UC3, and in the case of a system Mbean as in use case UC5. When an application security context has been set, the call to `setApplicationID` is optional.

The following are general recommendations when using the class `SubjectSecurity`.

- Use `SubjectSecurity.getActionExecutor(subject)` if you have already obtained a subject and want to execute some operation with it. Depending on how the subject was obtained, we have the following cases:
  - If the container authentication was performed and the subject was obtained using `SubjectUtil.getCurrentSubject()`, then that subject works with the container security and OPSS security, and application roles are considered.
  - If the subject was obtained via programmatic authentication using `JpsLoginModuleFactory`, then that subject works with the container security and OPSS security.
- Use `SubjectSecurity.getActionExecutor(userName)` if just the user name is available and want to execute some operation as that user. This method performs privilege operations including identity assertion, which requires the calling code to have the necessary permissions to invoke those operations. Therefore, applications using this method require a codebase permission like the following:

```
<grant>
    <grantee>
        <codesource>
```

```
                <url>file:${domain.home}/servers/${weblogic.Name}/myApp.ear/-</url>
            </codesource>
        </grantee>
        <permissions>
            <permission>
                <class>oracle.security.jps.JpsPermission</class>
                <name>IdentityAssertion</name>
                <actions>execute</actions>
            </permission>
        </permissions>
    </grant>
```

where `<url>` specifies the location of the code where the method is called.

- `ActionExecutor.execute(PrivilegedAction)` invokes `Subject.doAs()` with the user's JAAS subject and the application's `PrivilegedAction`.
- Use `SubjectSecurity.executeAs(Subject subject, PrivilegedAction<T> action)` to execute the action immediately.
- Use `SubjectSecurity.executeAs(Subject subject, PrivilegedExceptionAction<T> action)` to execute the action immediately. This method obtains the action executor using `getActionExecutor(subject)`, and executes the action.

For complete details about the class `oracle.security.jps.runtime.SubjectSecurity` and the interface `oracle.security.jps.runtime.ActionExecutor` see *Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services*.

25.3.3.4.3 A Code Example The following sample code illustrates the use of the `SubjectSecurity` API to run as an asserted user named `jdoe`:

```
// get a SubjectSecurity instance
        final SubjectSecurity subjectSecurity = SubjectSecurity.getInstance();
        String username = "jdoe";

// create ActionExecutor with subjectSecurity getActionExecutor(String username)
        ActionExecutor executor =
        AccessController.doPrivileged(new
PrivilegedExceptionAction<ActionExecutor>() {
            public ActionExecutor run() throws AssertionException {
                    return subjectSecurity.getActionExecutor(username);
            }
        }, null);

// When OPSS subjects are available,
// create ActionExecutor with SubjectSecurity getActionExecutor(Subject subject)
        Subject opssSubject = SubjectUtil.getCurrentSubject();
        ActionExecutor ececutor = subjectSecurity.getActionExecutor(opssSubject);

 //run privilegedAction
        PrivilegedAction action = new MyPrivilegedAction();
        executor.execute(action);

 //run PrivilegedExceptionAction
        PrivilegedExceptionAction exceptionAction = new
MyPrivilegedExceptionAction();
        try {
            executor.execute(exceptionAction);
        } catch (PrivilegedActionException e) {
            // handle PrivilegedActionException

        }
```

25.3.4 Using the OPSS API LoginService in Java SE Applications

To invoke a login module programmatically in Java SE applications, use the method `getLoginContext` of the interface `oracle.security.jps.service.login.LoginService`.

Similar to the method `LoginContext` in the standard JAAS API, `getLoginContext` returns an instance of a LoginContext object that can be used to authenticate a user, but, more generally, it also allows the use of any number of login modules in any order. Authentication is then performed on just those login modules and in the order they were passed.

The following code fragment illustrates user authentication against a subset of login modules in a prescribed order using `getLoginContext`:

```
import oracle.security.jps.service.ServiceLocator;
import oracle.security.jps.service.JpsServiceLocator;
import oracle.security.jps.service.login.LoginService;

//Obtain the login service
ServiceLocator locator = JpsServiceLocator.getServiceLocator();
LoginService loginService = locator.lookup(LoginService.class);

//Create the handler for given name and password
CallbackHandler cbh = new MyCallbackHandler("name", "password".toCharArray());

//Invoke login modules selectively in a given order
selectiveModules = new String[]{"lmName1", "lmName2", "lmName3"};
LoginContext ctx = loginService.getLoginContext(new Subject(), cbh,
selectiveModules);
ctx.login();
Subject s = ctx.getSubject();
```

`selectiveModules` is an array of (login module) names, and the authentication uses precisely those login modules named in the array in the order listed in the array. Each name in the array must be the name of a service instance listed in the default context of the file `jps-config-jse.xml`.

The following fragment illustrates the configuration of a stack of two login modules:

```
<serviceProvider type="LOGIN" name="jaas.login.provider"
class="oracle.security.jps.internal.login.jaas.JaasLoginServiceProvider">
   <description>Common definition for any login module instances</description>
</serviceProvider>

<serviceInstance name="auth.loginmodule" provider="jaas.login.provider">
   <description>User Authentication Login Module</description>
   <property name="loginModuleClassName"
value="oracle.security.jps.internal.jaas.module.authentication.JpsUserAuthenticati
onLoginModule"/>
   <property name="jaas.login.controlFlag" value="REQUIRED"/>
</serviceInstance>

<serviceInstance name="custom.loginmodule" provider="jaas.login.provider">
   <description>My Custom Login Module</description>
   <property name="loginModuleClassName" value="my.custom.MyLoginModuleClass"/>
   <property name="jaas.login.controlFlag" value="REQUIRED"/>
```

```
</serviceInstance>

<jpsContexts default="aJpsContext">
   <jpsContext name="aJpsContext">
     <serviceInstanceRef ref="auth.loginmodule"/>
     <serviceInstanceRef ref="custom.loginmodule"/>
   </jpsContext>
</jpsContexts>
```

25.4 Auditing in Java SE Applications

This section provides the background you need to get started leveraging the Common Audit Framework (CAF) to audit events with your Java SE applications. It explains basic requirements and describes some common audit usage scenarios for Java SE clients. It contains these topics:

- About Auditing in the Java SE Environment
- Configuring the Audit Bus-stop Directory
- Configuring Audit Loaders
- Implementing Common Audit Scenarios in JavaSE

25.4.1 About Auditing in the Java SE Environment

This section explains useful audit concepts.

> **See Also:** For background information about using OPSS in Java SE environments, see Section 25.1.

Audit Configuration Properties

As explained in Section 25.1, the audit configuration resides in the `jps-config-jse.xml` file.

Here is an example fragment from that file showing the audit properties:

```
<serviceInstance provider="audit.provider" name="audit">
 <property value="Medium" name="audit.filterPreset"/>
 <property value="0" name="audit.maxDirSize"/>
 <property value="104857600" name="audit.maxFileSize"/>
 <property value="" name="audit.specialUsers"/>
 <property value="" name="audit.customEvents"/>
 <property value="Db" name="audit.loader.repositoryType"/>
 <property value="file" name="auditstore.type"/>
</serviceInstance>
```

Audit Service and Processes

The audit run-time service enables the application to audit various events dictated by conditions specified in audit metadata (which resided in audit store), and records each event in the bus-stop files.

Other notable processes include the audit loaders which move data from bus-stop files to the database.

Bus-stop Files

These files are explicitly designed to hold the newly generated audit records. Although they are sometimes referred to mistakenly as audit logs, they should not be confused with system log files.

The following sections provide more details about aspects of audit operation including requirements and recommendations for the JavaSE environment.

25.4.2 Configuring the Audit Bus-stop Directory

Note: Starting with 11*g*Release 1 (11.1.1.9), the naming format of bus-stop files has changed to:

```
$hostname_$processId_audit_$majorVersion_$minorVersion.log
```

For example:

```
example.myhost.com_12345_audit_1_0.log
```

Java SE clients must specify a writable directory to which audit bus-stop files will be written. To configure this directory:

- Set it as an audit service property ("audit.logDirectory" in the file `jps-config-jse.xml`) using the `setAuditRepository`() WLST command. For example:

  ```
  setAuditRepository(logDirectory="/Audit")
  ```
  or

- Pass system property "opss.audit.logDirectory" OR "oracle.instance" to the JVM

WARNING: If the runtime process is unable to determine a writable directory, the audit service may stop working.

This directory can be shared between JavaSE client processes if:

- You set the directory with the audit.logDirectory service property in the file `jps-config-jse.xml` and all the Java SE clients/processes in the environment use the same `jps-config-jse.xml`

  or

- the same value for the bus-stop directory is passed as a system property to all Java SE clients.

25.4.3 Configuring Audit Loaders

The audit data loader is a thread/process that pushes the data from the bus-stop file to a database.

Note: Having the audit.loader.repositoryType property set to 'Db' is the default requirement for the audit loader for both JavaEE and JavaSE environments.

There are two kinds of loaders for JavaSE applications:

- The first is the JavaSE audit service loader, a thread started by the runtime OPSS audit service.

  Out-of-the-box, this audit loader is disabled. To enable it, pass "audit.loader.enable=true" to the JVM.

- The second loader is a standalone JavaSE application. It is the recommended loader for this environment, especially if you are using a shared directory as described in Section 25.4.2.

  For details about starting the stand-alone loader, see Section 14.2.4.2 and the example in the final step of Section 25.4.4.1.

25.4.4 Implementing Common Audit Scenarios in JavaSE

This section describes some common scenarios that implement auditing in Java SE.

We consider environments with and without a co-located Oracle WebLogic Server.

25.4.4.1 Auditing by JavaSE Clients, with co-located WebLogic Server

After setting up the domain, take these steps as administrator:

1. Set up the audit loader configuration by either executing the setAuditRepository() WLST command or by using Fusion Middleware Control.

   Note: setAuditRepository is an online command.

   This step also updates appropriate properties in the file `jps-config.xml` and `jps-config-jse.xml`, thus enabling both Java SE and Java EE scenarios. The WLST script is similar to the following example:

```
# logDirectory is the location of the audit log directory
# jdbcSring is the connect string
# user and pass refer to the IAU append schema user and password
# (These get stored in bootstrap credential store)

setAuditRepository(switchToDB = "true",dataSourceName = "jdbc/AuditDB",interval
= "20", timezone="utc", logDirectory="/foo/bar",
jdbcString="jdbc:oracle:thin:@host:1521:sid", dbUser="test_iau_append",
dbPassword="password");
```

   Note: A standalone audit loader is not needed if the writable audit directory for JavaSE processes is set to be the same directory as the log directory for Oracle WebLogic Server (say DOMAIN_HOME/servers/AdminServer/logs). The container's audit loader process takes care of this task.

2. If choosing a separate writable audit directory for JavaSE processes, start a standalone audit loader to push these records to the database. Pass one of the system properties, setting it to the value of the bus-stop directory. In the example below, `oracle.instance` is used:

```
$JAVA_HOME/bin/java
-classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_11.1.1/jps-manifest.jar:
```

```
$COMMON_COMPONENTS_HOME/modules/oracle.jdbc_11.1.1/ojdbc6dms.jar:
$COMMON_COMPONENTS_HOME/modules/oracle.iau_11.1.1./fmw_audit.jar
-Doracle.instance=$ORACLE_INST
-Doracle.security.jps.config=path_to_jps-config-jse.xml
oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

See Also: The final step in Section 25.4.4.2.

25.4.4.2 Auditing by JavaSE Clients, without co-located WebLogic Server

The assumption here is that a middleware home is available but no Oracle WebLogic Server is running. So Fusion Middleware Control or online WLST commands such as setAuditRepository() cannot be used.

The steps are as follows:

1. Out-of-the-box, audit runtime is enabled in JavaSE if a writable directory is provided to the JVM.

   Since audit.logDirectory service property is not available by default in the file `jps-config-jse.xml`, we recommend that you set either opss.audit.logDirectory OR oracle.instance system property to specify the bus-stop directory.

2. As administrator, you then add a bootstrap credential to the bootstrap credential store as defined in the file `jps-config-jse.xml` using the addBootStrapCredential( ) WLST command:

   ```
   addBootStrapCredential(jpsConfigFile='../../../user_projects/domains/base_
   domain/config/fmwconfig/jps-config-jse.xml', map='AuditDbPrincipalMap',
   key='AuditDbPrincipalMap', username='TEST_IAU_APPEND', password='password')
   ```

3. If JavaSE processes use a shared audit directory, start the standalone audit loader to push these records to the database.

4. Finally, ensure that the relevant audit loader is operational. There are two scenarios:

   - The audit directory is not shared (that is, every JavaSE process writes to its own audit directory).

     In this case, simply enable the runtime service's audit loader by specifying audit.loader.enable=true.

   - The audit directory is shared (that is, multiple JavaSE processes write to the same audit directory).

     In this case, start the standalone audit loader as shown in Section 25.4.4.1.

   Note: The standalone audit loader can also be used when the directory is not shared.

25.5 Configuration Examples

This section illustrates the configuration of the following artifacts:

- XML policy and credential stores
- XML and LDAP identity stores
- Login Module Principals

XML Policy and Credential Stores Configuration

The following snippets illustrate the configuration of XML-based policy and credential stores. The contents of an XML-based policy store is specified in the file `system-jazn-data.xml`; the contents of an XML-based credential store is specified in the file `cwallet.sso`.

```
<serviceProviders>
 <serviceProvider
class="oracle.security.jps.internal.policystore.xml.XmlPolicyStoreProvider"
name="policystore.xml.provider" type="POLICY_STORE">
  <description>XML-based PolicyStore Provider</description>
 </serviceProvider>

 <serviceProvider
class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
name="credstoressp" type="CREDENTIAL_STORE">
  <description>SecretStore-based CSF Provider</description>
 </serviceProvider>
</serviceProviders>

<serviceInstances>
 <serviceInstance location="./" provider="credstoressp" name="credstore">
  <description>File-based Credential Store Service Instance</description>
 </serviceInstance>

<serviceInstance location="./system-jazn-data.xml"
provider="policystore.xml.provider" name="policystore.xml">
   <description>File-based Policy Store Service Instance</description>
 </serviceInstance>
</serviceInstances>
```

XML Identity Store Configuration

The following snippets illustrate the configuration of an XML-based identity store. The contents of an XML-based identity store is specified in the file `system-jazn-data.xml`.

```
<serviceProvider
  class="oracle.security.jps.internal.idstore.xml.XmlIdentityStoreProvider"
  name="idstore.xml.provider"
  type="IDENTITY_STORE">
 <description>XML-based Identity Store Service Provider</description>
</serviceProvider>

<serviceInstance
  location="./system-jazn-data.xml" provider="idstore.xml.provider"
  name="idstore.xml">
 <description>File Based Identity Store Service Instance</description>
 <property value="jazn.com" name="subscriber.name"/>
</serviceInstance>
```

LDAP Identity Store Configuration

The snippets below illustrate the configuration of an LDAP-based identity store, which includes the required configuration of the bootstrap credentials to access the LDAP server. The service instance property `idstore.type` can have the following values, according to the LDAP used:

Table 25–2 Idstore Types

Supported LDAP	Idstore.type value
Oracle Internet Directory 10g and 11g	OID
Oracle Virtual Directory 10g and 11g	OVD
Sun Java System Directory Server 6.3	IPLANET
Active Directory 2003, 2008	ACTIVE_DIRECTORY
Novell eDirectory 8.8	EDIRECTORY
Oracle Directory Server Enterprise Edition 11gR1 (11.1.1.3+)	IPLANET
IBM Tivoli DS 6.2	OPEN_LDAP
OpenLDAP 2.2.	OPEN_LDAP

```
<serviceProvider
   class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider"
   name="idstore.ldap.provider" type="IDENTITY_STORE">
 <description>LDAP-based Identity Store Service Provider</description>
</serviceProvider>

<serviceProvider
   class="oracle.security.jps.internal.credstore.ssp.SspCredentialStoreProvider"
   name="credstoressp" type="CREDENTIAL_STORE">
 <description>SecretStore-based CSF Provider</description>
</serviceProvider>

<serviceInstance name="idstore.oid" provider="idstore.ldap.provider">
 <property name="subscriber.name" value="dc=us,dc=oracle,dc=com"/>
 <property name="idstore.type" value="OID"/>
 <property value=ldap://myOID.com:3555 name="ldap.url"/>
 <extendedProperty>
  <name>user.search.bases</name>
  <values>
   <value>cn=users,dc=us,dc=oracle,dc=com</value>
  </values>
 </extendedProperty>
 <extendedProperty>
 <name>group.search.bases</name>
  <values>
   <value>cn=groups,dc=us,dc=oracle,dc=com</value>
  </values>
 </extendedProperty>
 <property name="username.attr" value="uid"/>
 <propperty name="group.attr" value="cn"/>
</serviceInstance>

<serviceInstance location="./bootstrap" provider="credstoressp"
 name="bootstrap.cred">
 <property value="./bootstrap" name="location"/>
</serviceInstance>
```

Login Module Principals

The following properties are set in the out-of-the-box `jps-config-jse.xml`:

```
 <property name="oracle.security.jps.enterprise.user.class"
         value="weblogic.security.principal.WLSUserImpl"/>
```

```
    <property name="oracle.security.jps.enterprise.role.class"
             value="weblogic.security.principal.WLSGroupImpl"/>
```

The above propeties must be used in any login module; this implies that the principals that represent users and groups in the identity store are the following:

```
weblogic.security.principal.WLSUserImpl
weblogic.security.principal.WLSGroupImpl
```

Part V

Appendices

This part contains the following appendices:

A OPSS Configuration File Reference

This appendix describes the element hierarchy and attributes in the file that configures OPSS services. By default, this file is named `jps-config.xml` (in Java EE applications) or `jps-config-jse.xml` (in Java SE applications).

By default, the OPSS configuration file is located in the directory `$DOMAIN_HOME/config/fmwconfig`; in Java SE applications, an alternative location can be specified using the system property `oracle.security.jps.config`.

The configuration file is used to configure the security store, login modules, and the audit service. For a complete example of a configuration file see Section 24.6.9, "Example of Configuration File jps-config.xml."

To configure services programmatically, see Section E.2, "Configuring OPSS Services with MBeans."

This appendix includes the following sections:

- Top- and Second-Level Element Hierarchy
- Lower-Level Elements

A.1 Top- and Second-Level Element Hierarchy

The top element in the file `jps-config.xml` is <jpsConfig>. It contains the following second-level elements:

- `<property>`
- `<propertySets>`
- `<extendedProperty>`
- `<serviceProviders>`
- `<serviceInstances>`
- `<jpsContexts>`

Table A–1 describes the function of these elements. The annotations between curly braces`{}` indicate the number of occurrences the element is allowed. For example, `{0 or more}` indicates that the element can occur 0 or more times; `{1}` indicates that the element must occur once.

These elements are *not* application-specific configurations: all items in the configuration file pertain to an entire domain and apply to all managed servers and applications deployed on the domain.

Table A–1 First- and Second-Level Elements in jps-config.xml

Elements	Description
`<jpsConfig> {1}`	Defines the top-level element in the configuration file.
`<property> {0 or more}`	Defines names and values of properties. It can also appear elsewhere in the hierarchy, such as under the elements `<propertySet>`, `<serviceProvider>`, and `<serviceInstance>`.
`<propertySets> {0 or 1}`	Groups one or more `<propertySet>` elements so that they can referenced as a group.
`<propertySet> {1 or more}`	
`<property> {1 or more}`	
`<extendedProperty> {0 or more}`	Defines a property that has multiple values. It can also appear elsewhere in the hierarchy, such as under the elements extendedProperty and serviceInstance.
`<name> {1}`	
`<values> {1}`	
`<value> {1 or more}`	
`<extendedPropertySets> {0 or 1}`	Groups one or more `<extendedPropertySet>` elements so that they can referenced a group.
`<extendedPropertySet> {1 or more}`	
`<extendedProperty> {1 or more}`	
`<name> {1}`	
`<values> {1}`	
`<value> {1 or more}`	
`<serviceProviders> {0 or 1}`	Groups one or more `<serviceProvider>` elements, each of which defines an implementation of an OPSS service, such as a policy store provider, a credential store provider, or a login module.
`<serviceProvider> {1 or more}`	
`<description> {0 or 1}`	
`<property> {0 or more}`	
`<serviceInstances> {0 or 1}`	Groups one or more `<serviceInstance>` elements, each of which configures and specifies property values for a service provider defined in a `<serviceProvider>` element.
`<serviceInstance> {1 or more}`	
`<description> {0 or 1}`	
`<property> {0 or more}`	
`<propertySetRef> {0 or more}`	
`<extendedProperty> {0 or more}`	
`<name> {1}`	
`<values> {1}`	
`<value> {1 or more}`	
`<extendedPropertySetRef> {0 or more}`	
`<jpsContexts> {1}`	Groups one or more `<jpsContext>` elements, each of which is a collection of service instances that an application can use.
`<jpsContext> {1 or more}`	
`<serviceInstanceRef> {1 or more}`	

A.2 Lower-Level Elements

This section describes, in alphabetical order, the complete set of elements that can occur in under the second-level elements described in the Top- and Second-Level Element Hierarchy.

- <description>
- <extendedProperty>
- <extendedPropertySet>
- <extendedPropertySetRef>
- <extendedPropertySets>
- <jpsConfig>
- <jpsContext>

- <jpsContexts>
- <name>
- <property>
- <propertySet>
- <propertySetRef>
- <propertySets>
- <serviceInstance>
- <serviceInstanceRef>
- <serviceInstances>
- <serviceProvider>
- <serviceProviders>
- <value>
- <values>

<description>

This element describes the corresponding entity (a service instance or service provider).

Parent Elements

<serviceInstance> or <serviceProvider>

Child Element

None.

Occurrence

`<description>` can be a child of <serviceInstance> or <serviceProvider>.

- As a child of <serviceInstance>:

```
<serviceInstances> {0 or 1}
    <serviceInstance> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
        <propertySetRef> {0 or more}
        <extendedProperty> {0 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
        <extendedPropertySetRef> {0 or more}
```

- As a child of <serviceProvider>:

```
<serviceProviders> {0 or 1}
    <serviceProvider> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
```

Example

The following example sets a description for a service provider.

```
<serviceProvider ... >
   <description>XML-based IdStore Provider</description>
   ...
</serviceProvider>
```

<extendedProperty>

This element defines an extended property in the following scenarios:

Table A–2 Scenarios for <extendedProperty>

Location in jps-config.xml	Function
Directly under <jpsConfig>	Defines an extended property for general use. As a child of <jpsConfig>, an extended property can specify, for example, all the base DNs in an LDAP-based authenticators.
Directly under <extendedPropertySet>	Defines an extended property for general use that is part of an extended property set.
Directly under <serviceInstance>	Defines an extended property for a particular service instance.

An extended property typically includes multiple values. Use a <value> element to specify each value. Several LDAP identity store properties are in this category, such as the specification of the following values:

- Object classes used for creating user objects
- Attribute names that must be specified when creating a user
- Base DNs for searching users

Parent Elements

<extendedPropertySet>, <jpsConfig>, or <serviceInstance>

Child Elements

<name> or <values>

Occurrence

`<extendedProperty>` can be a child of <extendedPropertySet>, <jpsConfig>, or <serviceInstance>.

- As a child of <extendedPropertySet>:

```
<extendedPropertySets> {0 or 1}
    <extendedPropertySet> {1 or more}
        <extendedProperty> {1 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
```

- As a child of <jpsConfig>:

```
<jpsConfig>
    <extendedProperty> {0 or more}
        <name> {1}
        <values> {1}
            <value> {1 or more}
```

- As a child of <serviceInstance>:

```
<serviceInstances> {0 or 1}
```

```
<serviceInstance> {1 or more}
    <description> {0 or 1}
    <property> {0 or more}
    <propertySetRef> {0 or more}
    <extendedProperty> {0 or more}
        <name> {1}
        <values> {1}
            <value> {1 or more}
    <extendedPropertySetRef> {0 or more}
```

Example

The following example sets a single value:

```
<extendedProperty>
   <name>user.search.bases</name>
   <values>
      <value>cn=users,dc=us,dc=oracle,dc=com</value>
   </values>
</extendedProperty>
```

<extendedPropertySet>

This element defines a set of extended properties. The extended property set can then be referenced by an `<extendedPropertySetRef>` element to specify the given properties as part of the configuration of a service instance.

Attributes

Name	Description
`name`	Designates a name for the extended property set. No two `<extendedPropertySet>` elements may have the same `name` attribute setting within a configuration file. Values: string Default: n/a (required)

Parent Element

<extendedPropertySets>

Child Element

<extendedProperty>

Occurrence

Required within <extendedPropertySets>, one or more:

```
<extendedPropertySets> {0 or 1}
    <extendedPropertySet> {1 or more}
        <extendedProperty> {1 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
```

<extendedPropertySetRef>

This element configures a service instance by referring to an extended property set defined elsewhere in the file.

Attributes

Name	Description
`ref`	Refers to an extended property set whose extended properties are used for the service instance defined in the <serviceInstance> parent element. The `ref` value of `<extendedPropertySetRef>` must match the `name` value of an <extendedPropertySet> element. Values: string Default: n/a (required)

Parent Element

<serviceInstance>

Child Element

None.

Occurrence

Optional, zero or more.

```
<serviceInstances> {0 or 1}
    <serviceInstance> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
        <propertySetRef> {0 or more}
        <extendedProperty> {0 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
        <extendedPropertySetRef> {0 or more}
```

<extendedPropertySets>

This element specifies a set of properties.

Parent Element

<jpsConfig>

Child Element

<extendedPropertySet>

Occurrence

Optional, zero or one.

```
<jpsConfig>
    <extendedPropertySets> {0 or 1}
        <extendedPropertySet> {1 or more}
            <extendedProperty> {1 or more}
                <name> {1}
                <values> {1}
                    <value> {1 or more}
```

<jpsConfig>

This is the root element of a configuration file.

Parent Element

None.

Child Elements

<extendedProperty>, <extendedPropertySets>, <jpsContexts>, <property>, <propertySets>, <serviceInstances>, or <serviceProviders>

Occurrence

Required, one only.

Example

```
<jpsConfig xmlns="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_1.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://xmlns.oracle.com/oracleas/schema/11/jps-config-11_
1.xsd"
    schema-major-version="11" schema-minor-version="1">
...
</jpsConfig>
```

<jpsContext>

This element declares an OPSS context, a collection of service instances common to a domain, either by referring to a set of service instances that comprise the context (typical usage), or by referring to another context. Each `<jspContext>` in a configuration file must have a distinct name.

Attributes

Name	Description
`name`	Designates a name for the OPSS context. Each context must have a unique name. Values: string Default: n/a (required)

Parent Element

<jpsContexts>

Child Element

<serviceInstanceRef>

Occurrence

There must be at least one `<jpsContext>` element under <jpsContexts>. A `<jpsContext>` element contains the <serviceInstanceRef> element.

```
<jpsContexts> {1}
    <jpsContext> {1 or more}
        <serviceInstanceRef> {1 or more}
```

Example

The following example illustrates the definition of two contexts; the first one, named `default`, is the default context (specified by the attribute `default` in <jpsContexts>), and it references several service instances by name.

The second one, named `anonymous`, is used for unauthenticated users, and it references the `anonymous` and `anonymous.loginmodule` service instances.

```
<serviceInstances>
...
   <serviceInstance provider="credstoressp" name="credstore">
      <description>File Based Default Credential Store Service Instance</description>
      <property name="location" value="${oracle.instance}/config/JpsDataStore/JpsSystemStore"/>
   </serviceInstance>
...
   <serviceInstance provider="anonymous.provider" name="anonymous">
      <property value="anonymous" name="anonymous.user.name"/>
      <property value="anonymous-role" name="anonymous.role.name"/>
   </serviceInstance>
...
   <serviceInstance provider="jaas.login.provider" name="anonymous.loginmodule">
      <description>Anonymous Login Module</description>
      <property value="oracle.security.jps.internal.jaas.module.anonymous.AnonymousLoginModule"
```

```
                name="loginModuleClassName"/>
      <property value="REQUIRED"
                name="jaas.login.controlFlag"/>
   </serviceInstance>
...
</serviceInstances>
...
<jpsContexts default="default">
...
   <jpsContext name="default">
      <!-- This is the default JPS context. All the mandatory services and Login Modules must be
           configured in this default context -->
      <serviceInstanceRef ref="credstore"/>
      <serviceInstanceRef ref="idstore.xml"/>
      <serviceInstanceRef ref="policystore.xml"/>
      <serviceInstanceRef ref="idstore.loginmodule"/>
      <serviceInstanceRef ref="idm"/>
   </jpsContext>
   <jpsContext name="anonymous">
       <serviceInstanceRef ref="anonymous"/>
       <serviceInstanceRef ref="anonymous.loginmodule"/>
   </jpsContext>
...
</jpsContexts>
```

<jpsContexts>

This element specifies a set of contexts.

Attributes

Name	Description
`default`	Specifies the context that is used by an application if none is specified. The `default` value of the `<jpsContexts>` element must match the `name` of a <jpsContext> child element. Values: string Default: n/a (required) **Note**: The default context must configure all mandatory services and login modules.

Parent Element

<jpsConfig>

Child Element

<jpsContext>

Occurrence

Required, one only.

```
<jpsConfig>
    <jpsContexts> {1}
        <jpsContext> {1 or more}
```

Example

See <jpsContext> for an example.

<name>

This element specifies the name of an extended property.

Parent Element

<extendedProperty>

Child Element

None

Occurrence

Required, one only.

```
<extendedProperty> {0 or more}
    <name> {1}
    <values> {1}
        <value> {1 or more}
```

Example

See <extendedProperty> for an example.

<property>

This element defines a property in the following scenarios:

Table A–3 Scenarios for <property>

Location in jps-config.xml	Function
Directly under <jpsConfig>	Defines a one-value property for general use.
Directly under <propertySet>	Defines a multi-value property for general use that is part of a property set.
Directly under <serviceInstance>	Defines a property for use by a particular service instance.
Directly under <serviceProvider>	Defines a property for use by all service instances of a particular service provider.

For a list of properties, see Appendix F, "OPSS System and Configuration Properties".

Attributes

Name	Description
`name`	Specifies the name of the property being set. Values: string Default: n/a (required)
`value`	Specifies the value of the property being set. Values: string Default: n/a (required)

Parent Elements

<jpsConfig>, <propertySet>, <serviceInstance>, or <serviceProvider>

Child Element

None.

Occurrence

Under a`<propertySet>`, it is required, one or more; otherwise, it is optional, zero or more.

- As a child of <jpsConfig>:

```
<jpsConfig>
    <property> {0 or more}
```

- As a child of <propertySet>:

```
<propertySets> {0 or 1}
    <propertySet> {1 or more}
        <property> {1 or more}
```

- As a child of <serviceInstance>:

```
<serviceInstances> {0 or 1}
```

```
        <serviceInstance> {1 or more}
            <description> {0 or 1}
            <property> {0 or more}
            <propertySetRef> {0 or more}
            <extendedProperty> {0 or more}
                <name> {1}
                <values> {1}
                    <value> {1 or more}
            <extendedPropertySetRef> {0 or more}
```

- As a child of <serviceProvider>:

```
    <serviceProviders> {0 or 1}
        <serviceProvider> {1 or more}
            <description> {0 or 1}
            <property> {0 or more}
```

Example

The following example illustrates a property to disable JAAS mode for authorization:

```
<jpsConfig ... >
   ...
   <property name="oracle.security.jps.jaas.mode" value="off"/>
   ...
</jpsConfig>
```

For additional examples, see <propertySet> and <serviceInstance>.

<propertySet>

This element defines a set of properties. Each property set has a name so that it can be referenced by a <propertySetRef> element to include the properties as part of the configuration of a service instance.

Attributes

Name	Description
`name`	Designates a name for the property set. No two `<propertySet>` elements may have the same name within a `jps-config.xml` file. Values: string Default: n/a (required)

Parent Element

<propertySets>

Child Element

<property>

Occurrence

Required within a`<propertySets>`, one or more

```
<propertySets> {0 or 1}
    <propertySet> {1 or more}
        <property> {1 or more}
```

Example

```
<propertySets>
...
   <!-- For property that points to valid Access SDK installation directory -->
   <propertySet name="access.sdk.properties">
      <property name="access.sdk.install.path" value="$ACCESS_SDK_HOME"/>
   </propertySet>
...
</propertySets>

<serviceInstances>
...
   <serviceInstance provider="jaas.login.provider" name="oam.loginmodule">
      <description>Oracle Access Manager Login Module</description>
      <property
          value="oracle.security.jps.internal.jaas.module.oam.OAMLoginModule"
          name="loginModuleClassName"/>
      <property value="REQUIRED" name="jaas.login.controlFlag"/>
      <propertySetRef ref="access.sdk.properties"/>
   </serviceInstance>
...
</serviceInstances>
```

<propertySetRef>

This element configures a service instance by referring to a property set defined elsewhere in the file.

Attributes

Name	Description
`ref`	Refers to a property set whose properties are used by the service instance defined in the <serviceInstance> parent element. The `ref` value of a `<propertySetRef>` element must match the `name` of a <propertySet> element. Values: string Default: n/a (required)

Parent Element

<serviceInstance>

Child Element

None.

Occurrence

Optional, zero or more.

```
<serviceInstances> {0 or 1}
    <serviceInstance> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
        <propertySetRef> {0 or more}
        <extendedProperty> {0 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
        <extendedPropertySetRef> {0 or more}
```

Example

See <propertySet> for an example.

<propertySets>

This element specifies a set of property sets.

Parent Element

<jpsConfig>

Child Element

<propertySet>

Occurrence

Optional. If present, there can be only one `<propertySets>` element.

```
<jpsConfig>
    <propertySets> {0 or 1}
        <propertySet> {1 or more}
            <property> {1 or more}
```

Example

See <propertySet> for an example.

<serviceInstance>

This element defines an instance of a service provider, such as an identity store service instance, policy store service instance, or login module service instance.

Each provider instance specifies the name of the instance, used to refer to the provider within the configuration file; the name of the provider being instantiated; and, possibly, the properties of the instance. Properties include the location of the instance and can be specified directly, within the instance element itself, or indirectly, by referencing a property or a property set. To change the properties of a service instance, you can use the procedure explained in Section E.1, "Configuring OPSS Service Provider Instances with a Script."

Set properties and extended properties of a service instance in the following ways:

- Set properties directly through <property> subelements.
- Set extended properties directly through <extendedProperty> subelements.
- Refer to previously defined sets of properties through <propertySetRef> subelements.
- Refer to previously defined sets of extended properties through <extendedPropertySetRef> subelements.

Attributes

Name	Description
`name`	Designates a name for this service instance. Note that no two `<serviceInstance>` elements may have the same `name` attribute setting within a `jps-config.xml` file. Values: string Default: n/a (required)
`provider`	Indicates which service provider this is an instance of. The `provider` value of a `<serviceInstance>` element must match the `name` value of a <serviceProvider> element. Values: string Default: n/a (required)

Parent Element

<serviceInstances>

Child Elements

<description>, <extendedProperty>, <extendedPropertySetRef>, <property>, or <propertySetRef>

Occurrence

Required within `<serviceInstances>`, one or more.

```
<serviceInstances> {0 or 1}
    <serviceInstance> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
        <propertySetRef> {0 or more}
        <extendedProperty> {0 or more}
            <name> {1}
            <values> {1}
                <value> {1 or more}
        <extendedPropertySetRef> {0 or more}
```

Examples

Example 1

The following example illustrates the configuration of a file-based identity store service. For a file-based identity store, the subscriber name is the default realm. The example sets the lo cation using the `location` property.

```
<serviceInstances>
   <serviceInstance name="idstore.xml" provider="idstore.xml.provider">
      <!-- Subscriber name must be defined for XML Identity Store -->
      <property name="subscriber.name" value="jazn.com"/>
      <!-- This is the location of XML Identity Store -->
      <property name="location" value="./system-jazn-data.xml"/>
   </serviceInstance>
...
</serviceInstances>
```

Example 2

The following example illustrates the configuration a credential store service. It uses the `location` property to set the location of the credential store.

```
<serviceInstances>
   <serviceInstance provider="credstoressp" name="credstore">
      <description>File Based Default Credential Store Service
                   Instance</description>
      <property name="location"
                 value="${oracle.instance}/config/JpsDataStore/JpsSystemStore" />
   </serviceInstance>
...
</serviceInstances>
```

Example 3

The following example illustrates the configuration of an LDAP-based identity store using Oracle Internet Directory:

```
<serviceInstance name="idstore.oid" provider="idstore.ldap.provider">
   <property name="subscriber.name" value="dc=us,dc=oracle,dc=com"/>
   <property name="idstore.type" value="OID"/>
   <property name="security.principal.key" value="ldap.credentials"/>
   <property name="security.principal.alias" value="JPS"/>
   <property name="ldap.url" value="ldap://myServerName.com:389"/>
   <extendedProperty>
      <name>user.search.bases</name>
      <values>
         <value>cn=users,dc=us,dc=oracle,dc=com</value>
      </values>
   </extendedProperty>
   <extendedProperty>
      <name>group.search.bases</name>
      <values>
```

```
         <value>cn=groups,dc=us,dc=oracle,dc=com</value>
      </values>
   </extendedProperty>
   <property name="username.attr" value="uid"/>
   <property name="groupname.attr" value="cn"/>
</serviceInstance>
```

Example 4

The following example illustrates the configuration of an audit provider:

```
<serviceInstances>
   <serviceInstance name="audit" provider="audit.provider">
      <property name="audit.filterPreset" value="Low"/>
      <property name="audit.specialUsers" value ="admin, fmwadmin" />
      <property name="audit.customEvents" value ="JPS:CheckAuthorization,
          CreateCredential, OIF:UserLogin"/>
      <property name="audit.loader.jndi" value="jdbc/AuditDB"/>
      <property name="audit.loader.interval" value="15" />
      <property name="audit.maxFileSize" value="10240" />
      <property name=" audit.loader.repositoryType " value="Db" />
   </serviceInstance>
  </serviceInstances>
```

Example 5

The following example illustrates the configuration of a login module:

```
<serviceInstance name="user.authentication.loginmodule"
provider="jaas.login.provider">
    <description>User Authentication Login Module</description>
    <property name="loginModuleClassName"
value="oracle.security.jps.internal.jaas.module.authentication.JpsUserAuthenticati
onLoginModule"/>
    <property name="jaas.login.controlFlag" value="REQUIRED"/>
    <property name="enable.anonymous" value="true"/>
    <property name="remove.anonymous.role" value="false"/>
</serviceInstance>
```

See Also:

- <serviceProvider>, for related examples defining service providers referenced here.
- <jpsContext>, for a corresponding example of <serviceInstanceRef>.

<serviceInstanceRef>

This element refers to service instances.

Attributes

Name	Description
`ref`	Refers to a service instance that are part of the context defined in the <jpsContext> parent element. The `ref` value of a `<serviceInstanceRef>` element must match the `name` of a <serviceInstance> element. Values: string Default: n/a (required)

Parent Element

<jpsContext>

Child Element

None

Occurrence

Required within a <jpsContext>, one or more.

```
<jpsContexts> {1}
    <jpsContext> {1 or more}
        <serviceInstanceRef> {1 or more}
```

Example

See <jpsContext> for an example.

<serviceInstances>

This element is the parent of a <serviceInstance> element.

Parent Element

<jpsConfig>

Child Element

<serviceInstance>

Occurrence

Optional, zero or one.

```
<jpsConfig>
    <serviceInstances> {0 or 1}
        <serviceInstance> {1 or more}
            <description> {0 or 1}
            <property> {0 or more}
            <propertySetRef> {0 or more}
            <extendedProperty> {0 or more}
                <name> {1}
                <values> {1}
                    <value> {1 or more}
            <extendedPropertySetRef> {0 or more}
```

Example

See <serviceInstance> for an example.

<serviceProvider>

This element defines a service provider. Each provider specifies the type of the provider, such as credential store, authenticators, policy store, or login module; the name of the provider, used to refer to the provider within the configuration file; and the Java class that implements the provider and that is instantiated when the provider is created. Furthermore, the element `property` specifies settings used to instantiate the provider.

It specifies the following data:

- The type of service provider (specified in the `type` attribute)
- A designated name of the service provider (to be referenced in each `<serviceInstance>` element that defines an instance of this service provider)
- The class that implements this service provider and is instantiated for instances of this service provider
- Optionally, properties that are generic to any instances of this service provider

Attributes

Name	Description

`type`	Specifies the type of service provider being declared; it must be either of the following:
	`CREDENTIAL_STORE`
	`IDENTITY_STORE`
	`POLICY_STORE`
	`AUDIT`
	`LOGIN`
	`ANONYMOUS`
	`KEY_STORE`
	`IDM` (for pluggable identity management)
	`CUSTOM`
	The implementation class more specifically defines the type of provider, such as by implementing a file-based identity store or LDAP-based policy store, for example.
	Values: string (a value above)
	Default: n/a (required)

`name`	Designates a name for this service provider. This name is referenced in the `provider` attribute of <serviceInstance> elements to create instances of this provider. No two `<serviceProvider>` elements may have the same `name` attribute setting within a configuration file. Values: string Default: n/a (required)

Name	Description
`class`	Specifies the fully qualified name of the Java class that implements this service provider (and that is instantiated to create instances of the service provider). Values: string Default: n/a (required)

Parent Element

<serviceProviders>

Child Elements

<description> or <property>

Occurrence

Required within the <serviceProviders> element, one or more.

```
<serviceProviders> {0 or 1}
    <serviceProvider> {1 or more}
        <description> {0 or 1}
        <property> {0 or more}
```

Examples

The following example illustrates the specification of a login module service provider:

```
<serviceProviders>
   <serviceProvider type="LOGIN" name="jaas.login.provider"
        class="oracle.security.jps.internal.login.jaas.JaasLoginServiceProvider">
      <description>This is Jaas Login Service Provider and is used to configure
       login module service instances</description>
   </serviceProvider>
</serviceProviders>
```

The following example illustrates the definition of an audit service provider:

```
 <serviceProviders>
     <serviceProvider name="audit.provider" type="AUDIT"
class="oracle.security.jps.internal.audit.AuditProvider">
     </serviceProvider>
    </serviceProviders>
```

See <serviceInstance> for other examples.

<serviceProviders>

This element specifies a set of service providers.

Parent Element

<jpsConfig>

Child Element

<serviceProvider>

Occurrence

Optional, one only.

```
<jpsConfig>
    <serviceProviders> {0 or 1}
        <serviceProvider> {1 or more}
            <description> {0 or 1}
            <property> {0 or more}
```

Example

See <serviceProvider> for an example.

<value>

This element specifies a value of an extended property, which can have multiple values. Each `<value>` element specifies one value.

Parent Element

<values>

Child Element

None.

Occurrence

Required within <values>, one or more.

```
<extendedProperty> {0 or more}
    <name> {1}
    <values> {1}
        <value> {1 or more}
```

Example

See <extendedProperty> for an example.

<values>

This element is the parent element of a <value> element.

Parent Element

<extendedProperty>

Child Element

<value>

Occurrence

Required within `<extendedProperty>`, one only.

```
<extendedProperty> {0 or more}
    <name> {1}
    <values> {1}
        <value> {1 or more}
```

Example

See <extendedProperty> for an example.

B

File-Based Identity and Policy Store Reference

This appendix describes the elements and attributes in `system-jazn-data.xml`, which is the default store for file-based identity and policy stores in Oracle Platform Security Services.

Note: The file-based identity store is supported for Java SE applications only.

This appendix includes the following topics:

- Hierarchy of Elements in system-jazn-data.xml
- Elements and Attributes of system-jazn-data.xml

B.1 Hierarchy of Elements in system-jazn-data.xml

This section shows the element hierarchy of `system-jazn-data.xml`, or an application-specific `jazn-data.xml` file. The direct subelements of the `<jazn-data>` root element are:

- `<jazn-realm>`
- `<policy-store>`
- `<jazn-policy>`

Note: The `<jazn-principal-classes>` and `<jazn-permission-classes>` elements and their subelements may appear in the `system-jazn-data.xml` schema definition as subelements of `<policy-store>`, but are for backward compatibility only.

Table B–1 Hierarchy of Elements in system-jazn-data.xml

Hierarchy	Description
`<jazn-data>`	This is the top-level element in the `system-jazn-data.xml` file.

Table B–1 (Cont.) Hierarchy of Elements in system-jazn-data.xml

Hierarchy	Description

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```
	The `<jazn-realm>` section specifies security realms, and the users and enterprise groups (as opposed to application-level roles) included in each realm.

Table B–1 (Cont.) Hierarchy of Elements in system-jazn-data.xml

Hierarchy

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            |   <app-role> {1 or more}
            |       <name> {1}
            |       <class> {1}
            |       <display-name> {0 or 1}
            |       <description> {0 or 1}
            |       <guid> {0 or 1}
            |       <uniquename> {0 or 1}
            |       <extended-attributes> {0 or 1}
            |       |   <attribute> {1 or more}
            |       |       <name> {1}
            |       |       <values> {1}
            |       |           <value>  {1 or more}
            |       <members> {0 or 1}
            |           <member> {1 or more}
            |               <name> {1}
            |               <class> {1}
            |               <uniquename> {0 or 1}
            |               <guid> {0 or 1}
            <role-categories> {0 or 1}
            |   <role-category> {1 or more}
            |      <name> {1}
            |      <display-name> {0 or 1}
            |      <description> {0 or 1}
            |      <members> {0 or 1}
            |         <role-name-ref> {1}
            <resource-types> {0 or 1}
            |   <resource-type> {1 or more}
            |      <name> {1}
            |      <display-name> {1}
            |      <description> {0 or 1}
            |      <provider-name> {1}
            |      <matcher-class> {1}
            |      <actions-delimiter> {1}
            |      <actions> {0 or more}
            <resources> {0 or 1}
            |    <resource> {1 or more}
            |       <name> {1}
            |       <display-name> {1}
            |       <description> {0 or 1}
            |       <type-name-ref> {1}
            <permission-sets> {0 or 1}
            |   <permission-set> {1 or more}
            |       <name> {1}
            |       <member-resources> {1 or more}
            |          <member-resource> {1 or more}
            |              <resource-name> {1}
            |              <type-name-ref> {1}
            |              <actions> {0 or 1}
            <jazn-policy> {0 or 1}
            |   <grant> {0 or more}
            |       <description> {0 or 1}
            |       <grantee> {0 or 1}
            |       |   <principals> {0 or 1}
            |       |       <principal> {0 or more}
            |       |           <name> {1}
            |       |           <class> {1}
            |       |           <uniquename> {0 or 1}
```

Description

The `<policy-store>` section configures application-level policies. You can define roles at the application level, and members in the roles. Members can be users or roles.

When `<jazn-policy>` is specified under the `<application>` element, it specifies policies at the application level.

`<jazn-policy>` can also appear under the `<jazn-data>` element, in which case it specifies policies at the system level.

Table B–1 (Cont.) Hierarchy of Elements in system-jazn-data.xml

Hierarchy	Description

```
  <jazn-policy> {0 or 1}
    <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
      |   <principals> {0 or 1}
      |      <principal> {0 or more}
      |         <name> {1}
      |         <class> {1}
      |         <uniquename> {0 or 1}
      |         <guid> {0 or 1}
      |   <codesource> {0 or 1}
      |      <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
      <permission-sets>
      |   <permission-set>
      |      <name>
```
	When the `<jazn-policy>` element is located under the `<jazn-data>` element, it specifies policies at the system-level.
	
	`<jazn-policy>` can also appear under the `<application>` element, in which case it specifies policies at the application level.

B.2 Elements and Attributes of system-jazn-data.xml

This section describes the elements and attributes in the `system-jazn-data.xml` file.

Notes:

- You can update most settings in `system-jazn-data.xml` through Oracle Enterprise Manager Fusion Middleware Control.

- <actions>
- <actions-delimiter>
- <app-role>
- <app-roles>
- <application>
- <applications>
- <attribute>
- <class>
- <codesource>
- <credentials>
- <description>
- <display-name>
- <extended-attributes>

- <grant>
- <grantee>
- <guid>
- <jazn-data>
- <jazn-policy>
- <jazn-realm>
- <matcher-class>
- <member>
- <member-resource>
- <member-resources>
- <members>
- <name>
- <owner>
- <owners>
- <permission>
- <permissions>
- <permission-set>
- <permission-sets>
- <policy-store>
- <principal>
- <principals>
- <provider-name>
- <realm>
- <resource>
- <resource-name>
- <resources>
- <resource-type>
- <resource-types>
- <role>
- <role-categories>
- <role-category>
- <role-name-ref>
- <roles>
- <type>
- <type-name-ref>
- <uniquename>
- <url>

- <user>
- <users>
- <value>
- <values>

<actions>

This element specifies the operations permitted by the associated permission class. Values are case-sensitive and are specific to each permission implementation. Examples of actions are "invoke" and "read,write".

Parent Element

`<permission>`

Child Elements

None

Occurrence

Optional, zero or one:

```
 <jazn-policy> {0 or 1}
    <grant> {0 or more}
       <description> {0 or 1}
       <grantee> {0 or 1}
           <principals> {0 or 1}
           ...
           <codesource> {0 or 1}
              <url> {1}
       <permissions> {0 or 1}
           <permission> {1 or more}
              <class> {1}
              <name> {0 or 1}
              <actions> {0 or 1}
```

Examples

See `<jazn-policy>` for examples.

<actions-delimiter>

This element specifies the character used to separate the actions of the associated resource type.

Parent Element

`<resource-types>`

Child Elements

`<name>`, `<display-name>`, `<description>`, `<actions><roles>`, `<users>`

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <resource-types> {0 or 1}
               <resource-type> {1 or more}
                  <name> {1}
                  <display-name> {1}
                  <description> {0 or 1}
                  <provider-name> {1}
                  <matcher-class> {1}
                  <actions-delimiter> {1}
                  <actions> {0 or more}
```

Example

For an example, see <resource-type>.

<app-role>

This element specifies an application role.

Required subelements specify the following:

- `<name>` specifies the name of the application role.
- `<class>` specifies the fully qualified name of the class implementing the application role.

Optional subelements can specify the following:

- `<description>` provides more information about the application role.
- `<display-name>` specifies a display name for the application role, such as for use by GUI interfaces.
- `<guid>` specifies a globally unique identifier to reference the application role. This is for internal use only.
- `<members>` specifies the users, roles, or other application roles that are members of this application role.
- `<uniquename>` specifies a unique name to reference the application role. This is for internal use only.

Parent Element

`<app-roles>`

Child Elements

`<class>`, `<description>`, `<display-name>`, `<guid>`, `<members>`, `<name>`, `<uniquename>`

Occurrence

Required, one or more:

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                                <value>  {1 or more}
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
```

```
<class> {1}
<uniquename> {0 or 1}
<guid> {0 or 1}
```

Examples

See `<policy-store>` for examples.

<app-roles>

This element specifies a set of application roles.

Parent Element

`<application>`

Child Elements

`<app-role>`

Occurrence

Optional, zero or one:

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                ...
```

Example

See `<policy-store>` for examples.

<application>

This element specifies roles and policies for an application.

Required subelements specify the following information for an application:

- `<name>` specifies the name of the application.

Optional subelements can specify the following:

- `<description>` provides information about the application and its roles and policies.
- `<app-roles>` specifies any application-level roles
- `<jazn-policy>` specifies any application-level policies.

Parent Element

`<applications>`

Child Elements

`<app-roles>`, `<description>`,, `<jazn-policy>`, `<name>`, `<permission-sets>`, `<resource-types>`, `<resources>`, `<role-categories>`

Occurrence

Required, one or more:

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                ...
```

Example

See `<policy-store>` for examples.

<applications>

This element specifies a set of applications.

Parent Element

`<policy-store>`

Child Elements

`<application>`

Occurrence

Optional, zero or one

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
        ...
```

Example

See `<policy-store>` for an example.

<attribute>

This element specifies an attribute of an application role.

Parent Element

`<extended-attributes>`

Child Elements

`<name>`, `<values>`

Occurrence

Required, one or more:

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                                <value>  {1 or more}
                            <guid> {0 or 1}
```

<class>

This element specifies several values depending on its location in the configuration file:

- Within the <app-role> element, <class> specifies the fully qualified name of the class implementing the application role.

  ```
  <app-role>
  ...
     <class>oracle.security.jps.service.policystore.ApplicationRole</class>
  ```

- Within the <member> element, <class> specifies the fully qualified name of the class implementing the role member.

  ```
  <app-role>
  ...
     <members>
        <member>
        ...
           <class>
            weblogic.security.principal.WLSUserImpl
           </class>
  ```

- Within the <permission> element (for granting permissions to a principal), <class> specifies the fully qualified name of the class implementing the permission. Values are case-insensitive.

  ```
  <jazn-policy>
     <grant>
     ...
        <permissions>
           <permission>
              <class>java.io.FilePermission</class>
  ```

- Within the <principal> element (for granting permissions to a principal), it specifies the fully qualified name of the principal class, which is the class that is instantiated to represent a principal that is being granted a set of permissions.

  ```
  <jazn-policy>
     <grant>
     ...
        <grantee>
           <principals>
              <principal>
              ...
                 <class>oracle.security.jps.service.policystore.TestUser</class>
  ```

Parent Element

<app-role>, <member>, <principal>, or <permission>

Child Elements

None

Occurrence

Required, one only

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    ...
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}

<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                ...
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` and `<policy-store>` for examples.

<codesource>

This element specifies the URL of the code to which permissions are granted.

The policy configuration can also include a `<principals>` element, in addition to the `<codesource>` element. Both elements are children of a `<grantee>` element and they specify who or what the permissions in question are being granted to.

For variables that can be used in the specification of a `<codesource>` URL, see <url>.

Parent Element

`<grantee>`

Child Elements

`<url>`

Occurrence

Optional, zero or one

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<credentials>

This element specifies the authentication password for a user. The credentials are, by default, in obfuscated form.

Parent Element

`<user>`

Child Elements

None

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
```

Example

See `<jazn-realm>` for examples.

<description>

This element specifies a text string that provides textual information about an item. Depending on the parent element, the item can be an application role, application policy, permission grant, security role, or user.

Parent Element

`<app-role>`, `<application>`, `<grant>`, `<role>`, or `<user>`

Child Elements

None

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                ...
                <description>  {0 or 1}
                ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                ...
                <description>  {0 or 1}
                ...

<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                ...
                    <description> {0 or 1}

<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
```

Example

The `fmwadmin` user might have the following description:

```
<description>User with administrative privileges</description>
```

See `<jazn-realm>` for additional examples.

<display-name>

This element specifies the name of an item typically used by a GUI tool. Depending on the parent element, an item can be an application role, user, or enterprise group.

Parent Element

`<app-role>`, `<role>`, or `<user>`

Child Elements

None

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                ...

<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
```

Example

The `fmwadmin` user might have the following display name:

```
<display-name>Administrator</display-name>
```

See `<jazn-realm>` for additional examples.

<extended-attributes>

This element specifies attributes of an application role.

Parent Element

<app-role>

Child Elements

<attribute>

Occurrence

Optional, zero or one

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                                <value>  {1 or more}
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}
```

Example

```
<app-roles>
   <app-role>
      <name>Knight</name>
      <display-name>Fellowship For the Ring</display-name>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <extended-attributes>
         <attribute>
         <name>SCOPE</name>
         <values>
            <value>Part-I</value>
         </values>
         </attribute>
      </extended-attributes>
   </app-role>
```

<grant>

This element specifies the recipient of the grant - a codesource, or a set of principals, or both- and the permissions assigned to it.

Parent Element

`<jazn-policy>`

Child Elements

`<description>`, `<grantee>`, `<permissions>`, <permission-sets>

Occurrence

Optional, zero or more

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<grantee>

This element, in conjunction with a parallel `<permissions>` element, specifies who or what the permissions are granted to: a set of principals, a codesource, or both.

Parent Element

`<grant>`

Child Elements

`<codesource>`, `<principals>`

Occurrence

Optional, zero or one

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<guid>

This element is for internal use only. It specifies a globally unique identifier (GUID) to reference the item.

Depending on the parent element, the item to be referenced may be an application role, application role member, principal, enterprise group, or user. It is typically used with an LDAP provider to uniquely identity the item (a user, for example). A GUID is sometimes generated and used internally by Oracle Platform Security Services, such as in migrating a user or role to a different security provider. It is not an item that you would set yourself.

Parent Element

`<app-role>`, `<member>`, `<principal>`, `<role>`, or `<user>`

Child Elements

None

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                ...

<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                            <value>  {1 or more}
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}

<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      ...
```

Example

See `<jazn-realm>` for examples.

<jazn-data>

This element specifies the top-level element in the `system-jazn-data.xml` file-based policy store.

Attributes

Name	Description
`schema-major-version`	Specifies the major version number of the `system-jazn-data.xml` XSD. The value of this attribute is fixed at `11` for use with Oracle Fusion Middleware 11*g*.
`schema-minor-version`	Specifies the minor version number of the `system-jazn-data.xml` XSD. The value of this attribute is fixed at `0` for use with the Oracle Fusion Middleware 12.1.3 implementation.

Parent Element

n/a

Child Elements

`<jazn-policy>`, `<jazn-realm>`, `<policy-store>`

Occurrence

Required, one only

```
<jazn-data ... >  {1}
    <jazn-realm>  {0 or 1}
    ...

    <policy-store> {0 or 1}
    ...

    <jazn-policy> {0 or 1}
    ...
```

Example

```
<jazn-data
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:noNamespaceSchemaLocation=
                     "http://xmlns.oracle.com/oracleas/schema/jazn-data-11_0.xsd">
...
</jazn-data
```

<jazn-policy>

This element specifies policy grants that associate grantees (principals or codesources) with permissions.

This element can appear in two different locations in the `system-jazn-data.xml` file:

- Under the `<jazn-data>` element, it specifies global policies.
- Under the `<application>` element, it specifies application-level policies.

Parent Element

`<application>` or `<jazn-data>`

Child Elements

`<grant>`

Occurrence

Optional, zero or one

```
<jazn-data> {1}
    <jazn-policy> {0 or 1}
       <grant> {0 or more}
          <description> {0 or 1}
          <grantee> {0 or 1}
              <principals> {0 or 1}
              ...
              <codesource> {0 or 1}
                 <url> {1}
          <permissions> {0 or 1}
              <permission> {1 or more}
                 <class> {1}
                 <name> {0 or 1}
                 <actions> {0 or 1}
```

Example

Example B–1 <jazn-policy>

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>
                <principal>
                    <class>
                       oracle.security.jps.service.policystore.TestUser
                    </class>
                    <name>jack</name>
                </principal>
                <principal>
                    <class>
                       oracle.security.jps.service.policystore.TestUser
                    </class>
                    <name>jill</name>
                </principal>
            </principals>
            <codesource>
  <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
            </codesource>
        </grantee>
        <permissions>
            <permission>
                <class>oracle.security.jps.JpsPermission</class>
                <name>getContext</name>
            </permission>
            <permission>
                <class>java.io.FilePermission</class>
                <name>/foo</name>
                <actions>read,write</actions>
            </permission>
        </permissions>
    </grant>
</jazn-policy>
```

Example B–2 <jazn-policy>

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>
                <principal>
                    <class>
                       oracle.security.jps.service.policystore.TestAdminRole
                    </class>
                    <name>Farm=farm1,name=FullAdministrator</name>
                </principal>
            </principals>
            <codesource>
                <url>file://some-file-path</url>
            </codesource>
        </grantee>
        <permissions>
             permission>
                <class>javax.management.MBeanPermission</class>
                <name>
          oracle.as.management.topology.mbeans.InstanceOperations#getAttribute
                </name>
                <actions>invoke</actions>
            </permission>
        </permissions>
    </grant>
</jazn-policy>
```

<jazn-realm>

This element specifies security realms and the users and enterprise groups (as opposed to application-level roles) they include, and is the top-level element for user and role information

Attribute

Name	Description
default	Specifies which of the realms defined under this element is the default realm. The value of this attribute must match a <name> value under one of the <realm> subelements. Values: string Default: n/a (required)

Parent Element

<jazn-data>

Child Elements

<realm>

Occurrence

Optional, zero or one

```
<jazn-data>  {1}
    <jazn-realm>  {0 or 1}
        <realm>  {0 or more}
            <name>  {1}
            <users>  {0 or 1}
            ...
            <roles>  {0 or 1}
            ...
```

Example

```
<jazn-data ... >
    ...
    <jazn-realm default="jazn.com">
        <realm>
            <name>jazn.com</name>
            <users>
                <user deactivated="true">
                    <name>anonymous</name>
                    <guid>61FD29C0D47E11DABF9BA765378CF9F3</guid>
                    <description>The default guest/anonymous user</description>
                </user>
                <user>
                    <name>developer1</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>developer2</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>manager1</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>manager2</name>
                    <credentials>!password</credentials>
                </user>
                <!-- these are for testing the admin role hierachy. -->
                <user>
                    <name>farm-admin</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>farm-monitor</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>farm-operator</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>farm-auditor</name>
                    <credentials>!password</credentials>
                </user>
                <user>
                    <name>farm-auditviewer</name>
                    <credentials>!password</credentials>
                </user>
            </users>
            <roles>
                <role>
                    <name>users</name>
                    <guid>31FD29C0D47E11DABF9BA765378CF9F7</guid>
                    <display-name>users</display-name>
                    <description>users role for rmi/ejb access</description>
                </role>
                <role>
                    <name>ascontrol_appadmin</name>
                    <guid>51FD29C0D47E11DABF9BA765378CF9F7</guid>
                    <display-name>ASControl App Admin Role</display-name>
                    <description>
                       Application Administrative role for ASControl
                    </description>
                </role>
                <role>
                    <name>ascontrol_monitor</name>
                    <guid>61FD29C0D47E11DABF9BA765378CF9F7</guid>
                    <display-name>ASControl Monitor Role</display-name>
                    <description>Monitor role for ASControl</description>
                </role>
                <role>
                    <name>developers</name>
                    <members>
                        <member>
                            <type>user</type>
                            <name>developer1</name>
                        </member>
```

```
                        <member>
                            <type>user</type>
                            <name>developer2</name>
                        </member>
                    </members>
                </role>
                <role>
                    <name>managers</name>
                    <members>
                        <member>
                            <type>user</type>
                            <name>manager1</name>
                        </member>
                        <member>
                            <type>user</type>
                            <name>manager2</name>
                        </member>
                    </members>
                </role>
            </roles>
        </realm>
    </jazn-realm>
    ...
</jazn-data>
```

<matcher-class>

This element specifies the fully qualified name of the class within a resource type; queries for resources of this type delegate to this matcher class. Values are case-sensitive.

Parent Element

<resource-type>

Child Elements

None

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <resource-types> {0 or 1}
               <resource-type> {1 or more}
                  <name> {1}
                  <display-name> {1}
                  <description> {0 or 1}
                  <provider-name> {1}
                  <matcher-class> {1}
                  <actions-delimiter> {1}
                  <actions> {1 or more}
```

Example

For an example, see <resource-type>.

<member>

This element specifies the members of a set, such as a `<role>` or an`<app-role>` element:

- When under a `<role>` element, it specifies a member of the enterprise group. A member can be a user or another enterprise group. The `<name>` subelement specifies the name of the member, and the `<type>` subelement specifies whether the member type (a user or an enterprise group).
- When under an `<app-role>` element, it specifies a member of the application role. A member can be a user, an enterprise group, or an application role. The `<name>` subelement specifies the name of the member, and the `<class>` subelement specifies the class that implements it. The member type is determined through the `<class>` element.

  Optional subelements include `<uniquename>` and `<guid>`, which specify a unique name and unique global identifier; these optional subelements are for internal use only.

Parent Element

`<members>`

Child Elements

- When under a `<role>` element, the `<member>` element has the following child elements: `<name>`, `<type>`
- When under an `<app-role>` element, the `<member>` element has the following child elements: `<name>`, `<class>`, `<uniquename>`, `<guid>`

Occurrence

Optional, zero or more

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}

<policy-store> {0 or 1}
```

```
<applications> {0 or 1}
    <application> {1 or more}
        <name> {1}
        <description> {0 or 1}
        <app-roles> {0 or 1}
            <app-role> {1 or more}
                <name> {1}
                <class> {1}
                <display-name> {0 or 1}
                <description> {0 or 1}
                <guid> {0 or 1}
                <uniquename> {0 or 1}
                <extended-attributes> {0 or 1}
                ...
                <members> {0 or 1}
                    <member> {1 or more}
                        <name> {1}
                        <class> {1}
                        <uniquename> {0 or 1}
                        <guid> {0 or 1}
```

Example

See `<jazn-realm>` and `<policy-store>` for examples.

<member-resource>

This element specifies resources for a permission set.

Parent Element

`<member-resources>`

Child Elements

`<resource-name>`, `<type-name-ref>`, `<actions>`

Occurrence

Required within <member-resources>, one or more.

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <permission-sets> {0 or 1}
               <permission-set> {1 or more}
                   <name> {1}
                   <member-resources> {1 or more}
                      <member-resource> {1 or more}
                          <resource-name> {1}
                          <type-name-ref> {1}
                          <actions> {0 or 1}
```

Example

For an example, see <permission-set>.

<member-resources>

This element specifies a set of member resources.

Parent Element

`<permission-set>`

Child Elements

`<member-resource>`

Occurrence

Required within <permission-sets>; one or more.

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <permission-sets> {0 or 1}
               <permission-set> {1 or more}
                   <name> {1}
                   <member-resources> {1 or more}
                      <member-resource> {1 or more}
                          <resource-name> {1}
                          <type-name-ref> {1}
                          <actions> {0 or 1}
```

Example

For an example, see <permission-set>.

<members>

This element specifies a set of members.

Parent Element

<role>, <app-role>

Child Elements

<member>

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}

<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                    ...
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}
```

Example

See `<jazn-realm>` and `<policy-store>` for examples.

<name>

This element has different uses, depending on its location in the file:

- Within the <app-role> element, it specifies the name of an application-level role in the policy configuration. For example:

  ```
  <name>Farm=farm1,name=FullAdministrator</name>
  ```

  Or a simpler example:

  ```
  <name>Myrolename</name>
  ```

- Within the <application> element, it specifies the policy context identifier. Typically, this is the name of the application during deployment.
- Within the <attribute> element, it specifies the name of an additional attribute for the application-level role.
- Within the <member> element, it specifies the name of a member of an enterprise group or application role (depending on where the <member> element is located). For example, if the `fmwadmin` user is to be a member of the role:

  ```
  <name>fmwadmin</name>
  ```

- Within the <owner> element, it specifies the name of an owner of an enterprise group. For example:

  ```
  <name>mygroupowner</name>
  ```

- Within the <permission> element, as applicable, it can specify the name of a permission that is meaningful to the permission class. Values are case-sensitive. For example:

  ```
  <name>
     oracle.as.management.topology.mbeans.InstanceOperations#getAttribute
  </name>
  ```

  Or:

  ```
  <name>getContext</name>
  ```

- Within the <principal> element (for granting permissions to a principal), it specifies the name of a principal within the given realm. For example:

  ```
  <name>Administrators</name>
  ```

- Within the <realm> element, it specifies the name of a realm. For example:

  ```
  <name>jazn.com</name>
  ```

- Within the <role> element, it specifies the name of an enterprise group in a realm. For example:

  ```
  <name>Administrators</name>
  ```

- Within the <user> element, it specifies the name of a user in a realm. For example:

  ```
  <name>fmwadmin</name>
  ```

- Within the `<resource-type>` element, it specifies the name of a resource type and is required. For example:

  ```
  <name>restype1</name>
  ```

Parent Element

`<app-role>`, `<application>`, `<attribute>`, `<member>`, `<owner>`, `<permission>`, `<principal>`, `<realm>`, `<role>`, or `<user>`

Child Elements

None

Occurrence

Required within any parent element other than `<permission>`, one only; optional within `<permission>`, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}

<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
```

```
                                <values> {1}
                                    <value>  {1 or more}
                        <members> {0 or 1}
                            <member> {1 or more}
                                <name> {1}
                                <class> {1}
                                <uniquename> {0 or 1}
                                <guid> {0 or 1}

<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

```
<application>
   <name>peanuts</name>
   <app-roles>
      <app-role>
         <name>snoopy</name>
         <display-name>application role snoopy</display-name>
         <class>oracle.security.jps.service.policystore.ApplicationRole</class>
         <members>
            <member>

.......
```

See `<jazn-policy>`, `<jazn-realm>`, and `<policy-store>` for examples.

<owner>

This element specifies the owner of the enterprise group, where an owner has administrative authority over the role.

An owner is a user or another enterprise group. The `<type>` subelement specifies the owner's type. The concept of role (group) owners specifically relates to BPEL or Oracle Internet Directory functionality. For example, in BPEL, a role owner has the capability to create and update workflow rules for the role.

Note: To create a group owner in Oracle Internet Directory, use the Oracle Delegated Administration Services.

Parent Element

`<owners>`

Child Elements

`<name>`, `<type>`

Occurrence

Optional, zero or more

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```

<owners>

This element specifies a set of owners.

Parent Element

`<role>`

Child Elements

`<owner>`

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```

<permission>

This element specifies the permission to grant to grantees, where a grantee is a set of principals, a codesource, or both, as part of a policy configuration.

Parent Element

`<permissions>`

Child Elements

`<actions>`, `<class>`, `<name>`

Occurrence

Required within parent element, one or more

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<permissions>

This element specifies a set of permissions.

The `<permissions>` element (used in conjunction with a parallel `<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 Oracle Platform Security runtime implementation requires its use within any `<grant>` element.

Parent Element

`<grant>`

Child Elements

`<permission>`

Occurrence

Optional, zero or one

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<permission-set>

A permission set or entitlement specifies a set of permissions.

Parent Element

<permission-sets>

Child Elements

<name>

Occurrence

Optional, zero or more

```
    <policy-store> {0 or 1}
        <applications> {0 or 1}
            <application> {1 or more}
                <name> {1}
                <description> {0 or 1}
                <app-roles> {0 or 1}
                ...
                <role-categories> {0 or 1}
                ...
                <permission-sets> {0 or 1}
                   <permission-set> {1 or more}
                       <name> {1}
                       <member-resources> {1 or more}
                          <member-resource> {1 or more}
                              <resource-name> {1}
                              <type-name-ref> {1}
                              <actions> {0 or 1}
```

Example

The following fragment illustrates the configuration of a permission set (or entitlement):

```
<permission-sets>
  <permission-set>
    <name>permsetName</name>
    <member-resources>
      <member-resource>
         <type-name-ref>TaskFlowResourceType</type-name-ref>
         <resource-name>resource1</resource-name>
         <actions>customize,view</actions>
      </member-resource>
    </member-resources>
  </permission-set>
</permission-sets>
```

Note the following points about a permission set:

- The actions specified in a <member-resource> must match one or more of the actions specified for the resource type that is referenced through <resource-name-ref>.
- A <member-resources> can have multiple <member-resource> elements in it.

- A permission set must have at least one resource.
- Permission sets can exist without necessarily being referenced in any grants, that is, without granting them to any principal.

In addition, the following strings in a permission set entry conform to the case sensitivity rules:

- The name is case insensitive.
- The description string is case insensitive.
- The display name is case insensitive.

<permission-sets>

This element specifies a set of permission sets.

Parent Element

`<application>`

Child Elements

`<permission-set>`

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <permission-sets> {0 or 1}
               <permission-set> {1 or more}
                   <name> {1}
                   <member-resources> {1 or more}
                      <member-resource> {1 or more}
                          <resource-name> {1}
                          <type-name-ref> {1}
                          <actions> {0 or 1}
```

Example

For an example, see <permission-set>.

<policy-store>

This element configures application-level policies, through an `<applications>` subelement. Under the `<applications>` element is an `<application>` subelement for each application that is to have application-level policies. The policies are specified through a `<jazn-policy>` subelement of each `<application>` element.

Note: The `<jazn-principal-classes>` and `<jazn-permission-classes>` elements and their subelements may appear in the `system-jazn-data.xml` schema definition as subelements of `<policy-store>`, but are for backward compatibility only.

Parent Element

`<jazn-data>`

Child Elements

`<applications>`

Occurrence

Optional, zero or one

```
<jazn-data>  {1}
    <policy-store> {0 or 1}
        <applications> {0 or 1}
            <application> {1 or more}
            ...
```

Example

```
<jazn-data ... >
    ...
    <policy-store>
        <!--  application policy -->
        <applications>
            <application>
                <name>policyOnly</name>
                <jazn-policy>
                   ...
                </jazn-policy>
            </application>
            <application>
                <name>roleOnly</name>
                <app-roles>
                    <app-role>
                        <name>Fellowship</name>
                        <display-name>Fellowship of the Ring</display-name>
                        <class>
                           oracle.security.jps.service.policystore.ApplicationRole
                        </class>
                    </app-role>
                    <app-role>
                        <name>King</name>
                        <display-name>Return of the King</display-name>
                        <class>
                           oracle.security.jps.service.policystore.ApplicationRole
                        </class>
                    </app-role>
                </app-roles>
            </application>
            <application>
                <app-roles>
                    <app-role>
                        <name>Farm=farm1,name=FullAdministrator</name>
                        <display-name>farm1.FullAdministrator</display-name>
                        <guid>61FD29C0D47E11DABF9BA765378CF9F2</guid>
                        <class>
                           oracle.security.jps.service.policystore.ApplicationRole
                        </class>
                        <members>
                            <member>
                                <class>
             oracle.security.jps.internal.core.principals.JpsXmlEnterpriseRoleImpl
                                </class>
                                <name>admin</name>
                            </member>
                        </members>
                    </app-role>
                </app-roles>
                <jazn-policy>
                   ...
                </jazn-policy>
            </application>
            ...
        </applications>
    </policy-store
    ....
</jazn-data
```

See <jazn-policy> for examples of that element.

<principal>

This element specifies a principal being granted the permissions specified in a <permissions> element as part of a policy configuration. Required under <principals>.

Subelements specify the name of the principal and the class that implements it, and optionally specify a unique name and unique global identifier (the latter two for internal use only).

For details about how principal names can be compared, see Section 2.7, "About Principal Name Comparison Logic."

Parent Element

`<principals>`

Child Elements

`<class>`, `<guid>`, `<name>`, `<uniquename>`

Occurrence

Optional, zero or more

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<principals>

This element specifies a set of principals.

For policy configuration, a `<principals>` element and/or a `<codesource>` element are used under a `<grantee>` element to specify who or 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.

Parent Element

`<grantee>`

Child Elements

`<principal>`

Occurrence

Optional, zero or one

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

See `<jazn-policy>` for examples.

<provider-name>

This element specifies the name of a resource type provider. The resource resides in a location external to the OPSS policy store. Values are case-insensitive.

Parent Element

`<resource-type>`

Child Elements

None

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <resource-types> {0 or 1}
               <resource-type> {1 or more}
                  <name> {1}
                  <display-name> {1}
                  <description> {0 or 1}
                  <provider-name> {1}
                  <matcher-class> {1}
                  <actions-delimiter> {1}
                  <actions> {0 or more}
```

Example

For an example, see <resource-type>.

<realm>

This element specifies a security realm, and the users and roles that belong to the realm.

Parent Element

`<jazn-realm>`

Child Elements

`<name>`, `<roles>`, `<users>`

Occurrence

Optional, zero or more

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
        ...
```

Example

See `<jazn-realm>` for an example.

<resource>

This element specifies an application resource and contains information about the resource.

Parent Element

`<resources>`

Child Elements

`<name>`, `<description>`, `<display-name>`, <type-name-ref>.

Occurrence

One of more required under <resources>.

```
<resources> (0 or more)
    <resource> (1 or more)
       <name> (1)
       <display-name> (1)
       <description> {0 or 1}
       <type-name-ref> (1)
```

Example

The following fragment illustrates the configuration of a resource (instance):

```
<resources>
  <resource>
    <name>resource1</name>
    <display-name>Resource1DisplayName</display-name>
    <description>Resource1 Description</description>
    <type-name-ref>TaskFlowResourceType</type-name-ref>
  </resource>
</resources>
```

Note the following points about case sensitivity of various strings in a resource entry:

- The name is case sensitive.
- The description string is case insensitive.
- The display name is case insensitive.

<resources>

This element specifies a collection of application resources.

Parent Element

`<application>`

Child Elements

`<resource>`

Occurrence

Optional, zero or more

```
 <resources> (0 or more)
    <resource> (1 or more)
       <name> (1)
       <display-name> (1)
       <description> {0 or 1}
       <type-name-ref> (1)
```

Example

For an example, see <resource>.

<resource-name>

This element specifies a member resource in a permission set. Values are case-sensitive.

Parent Element

`<member-resource>`

Child Elements

None

Occurrence

Optional, zero or more

```
    <policy-store> {0 or 1}
        <applications> {0 or 1}
            <application> {1 or more}
                <name> {1}
                <description> {0 or 1}
                <app-roles> {0 or 1}
                ...
                <role-categories> {0 or 1}
                ...
                <permission-sets> {0 or 1}
                   <permission-set> {1 or more}
                       <name> {1}
                       <member-resources> {1 or more}
                          <member-resource> {1 or more}
                              <resource-name> {1}
                              <type-name-ref> {1}
                              <actions> {0 or 1}
```

Example

For an example, see <permission-set>.

<resource-type>

This element specifies the type of a secured artifact, such as a flow, a job, or a web service. Values are case-insensitive.

Parent Element

`<resource-types>`

Child Elements

`<name>`, `<display-name>`, `<description>`, `<actions>`, `<actions-delimiter>`, `<matcher-class>`, `<provider-name>`.

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <resource-types> {0 or 1}
               <resource-type> {1 or more}
                  <name> {1}
                  <display-name> {1}
                  <description> {0 or 1}
                  <provider-name> {1}
                  <matcher-class> {1}
                  <actions-delimiter> {1}
                  <actions> {0 or more}
```

Example

The following fragment illustrates the configuration of a resource type:

```
<resource-types>
  <resource-type>
    <name>TaskFlowResourceType</name>
    <display-name>TaskFlowResourceType_disp</display-name>
    <description>Resource Type for Task Flow</description>
    <provider-name>resTypeProv</provider-name>
    <matcher-class>
oracle.adf.controller.security.TaskFlowPermission</matcher-class>
    <actions-delimiter>,</actions-delimiter>
    <actions>customize,view</actions>
  </resource-type>
</resource-types>
```

The following points apply to the specification of a resource type:

- The name is required and case insensitive.

- The provider name is optional and case insensitive. A provider is typically used when there are resources managed in an external store, that is, in a store other than the OPSS domain policy store.

  When specified, the class in a <provider-name> element is used as a resource finder; queries for resources of this type (via the `ResourceManager` search APIs) delegate to this matcher class instead of using the built-in resource finder against the OPSS domain policy store.
- The matcher class name is required and case sensitive.
- The description string is optional and case insensitive.
- The display name is optional and case insensitive.
- The action string is optional and case sensitive. The list of actions in a resource type can be empty. An empty action list indicates that the actions on instances of the resource type are determined externally and are opaque to OPSS.

<resource-types>

This element specifies a set of resource types.

Parent Element

`<application>`

Child Elements

`<resource-type>`

Occurrence

Optional, zero or more

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
            ...
            <role-categories> {0 or 1}
            ...
            <resource-types> {0 or 1}
               <resource-type> {1 or more}
                  <name> {1}
                  <display-name> {1}
                  <description> {0 or 1}
                  <provider-name> {1}
                  <matcher-class> {1}
                  <actions-delimiter> {1}
                  <actions> {0 or more}
```

Example

For an example, see <resource-type>.

<role>

This element specifies an enterprise security role, as opposed to an application-level role, and the members (and optionally owners) of that role.

Parent Element

<roles>

Child Elements

<description>, <display-name>, <guid>, <members>, <name>, <owners>

Occurrence

Optional, zero or more

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```

Example

See <jazn-realm> for examples.

<role-categories>

This element specifies the parent element of <role-category> elements.

Parent Element

<application>

Child Elements

<role-category>

Occurrence

Optional, zero or one

```
 <application> {1 or more}
               <name> {1}
               <description> {0 or 1}
               <app-roles> {0 or 1}
                  <app-role> {1 or more}
                     <name> {1}
                     <class> {1}
                      <display-name> {0 or 1}
                      <description> {0 or 1}
                      <guid> {0 or 1}
                      <uniquename> {0 or 1}
                      <extended-attributes> {0 or 1}
                         <attribute> {1 or more}
                             <name> {1}
                             <values> {1}
                                 <value>  {1 or more}
                      <members> {0 or 1}
                          <member> {1 or more}
                              <name> {1}
                              <class> {1}
                              <uniquename> {0 or 1}
                              <guid> {0 or 1}
               <role-categories> {0 or 1
                  <role-category> {1 or more}
                     <name> {1}
                     <description> {0 or 1}
                     <display-name> {0 or 1}
```

Example

See Section 17.3.3.1, "Using the Method checkPermission" for an example.

<role-category>

This element specifies a category, that is, a flat set of application roles.

Parent Element

<role-categories>

Child Elements

<name>, <display-name>, <description>, <members>

Occurrence

Optional, zero or one

```
<application> {1 or more}
               <name> {1}
               <description> {0 or 1}
               <app-roles> {0 or 1}
                  <app-role> {1 or more}
                     <name> {1}
                     <class> {1}
                      <display-name> {0 or 1}
                      <description> {0 or 1}
                      <guid> {0 or 1}
                      <uniquename> {0 or 1}
                      <extended-attributes> {0 or 1}
                         <attribute> {1 or more}
                             <name> {1}
                             <values> {1}
                                 <value>  {1 or more}
                      <members> {0 or 1}
                          <member> {1 or more}
                              <name> {1}
                              <class> {1}
                              <uniquename> {0 or 1}
                              <guid> {0 or 1}
               <role-categories> {0 or 1}
                  <role-category> {1 or more}
                     <name> {1}
                     <description>{0 or 1}
                     <display-name>{0 or 1}
                     <members> {0 or 1}
```

Example

See Section 17.3.3.1, "Using the Method checkPermission" for an example.

<role-name-ref>

This element specifies an application role within a role category.

Parent Element

<members>

Child Elements

None

Occurrence

Optional, zero or one

```
<application> {1 or more}
               <name> {1}
               <description> {0 or 1}
               <app-roles> {0 or 1}
                  <app-role> {1 or more}
                     <name> {1}
                     <class> {1}
                      <display-name> {0 or 1}
                      <description> {0 or 1}
                      <guid> {0 or 1}
                      <uniquename> {0 or 1}
                      <extended-attributes> {0 or 1}
                         <attribute> {1 or more}
                             <name> {1}
                             <values> {1}
                                 <value>  {1 or more}
                      <members> {0 or 1}
                          <member> {1 or more}
                              <name> {1}
                              <class> {1}
                              <uniquename> {0 or 1}
                              <guid> {0 or 1}
               <role-categories> {0 or 1}
                  <role-category> {1 or more}
                     <name> {1}
                     <description> {0 or 1}
                     <members> {0 or 1}
                        <role-name-ref> {1}
```

<roles>

This element specifies a set of enterprise security roles that belong to a security realm.

Parent Element

`<realm>`

Child Elements

`<role>`

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```

Example

See `<jazn-realm>` for an example.

<type>

This element specifies the type of an enterprise group member or role owner: specifically, whether the member or owner is a user or another role:

```
<type>user</type>
```

Or:

```
<type>role</type>
```

Parent Element

`<member>` or `<owner>`

Child Elements

None

Occurrence

Required, one only

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
        ...
        <roles>  {0 or 1}
            <role>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <members>  {0 or 1}
                    <member>  {0 or more}
                        <type>  {1}
                        <name>  {1}
                <owners>  {0 or 1}
                    <owner>  {0 or more}
                        <type>  {1}
                        <name>  {1}
```

Example

See `<jazn-realm>` for examples.

<type-name-ref>

This element specifies the resource type of a resource.

Parent Element

`<member-resource>, <resource>`

Child Elements

None

Occurrence

One only. Required within <resource> or <member-resource>.

```
<resources> (0 or more)
   <resource> (1 or more)
      <name> (1)
      <display-name> (1)
      <description> {0 or 1}
      <type-name-ref> (1)
```

Example

For an example, see <resource>.

<uniquename>

This element, for internal use, takes a string value to specify a unique name to reference the item. (The `JpsPrincipal` class can use a GUID and unique name, both computed by the underlying policy provisioning APIs, to uniquely identify a principal.) Depending on the parent element, the item could be an application role, application role member (not an enterprise group member), or principal. It is typically used with an LDAP provider to uniquely identity the item (an application role member, for example). A unique name is sometimes generated and used internally by Oracle Platform Security.

The unique name for an application role would be: `"appid=application_name, name=actual_rolename"`. For example:

```
<principal>
   <class>
      oracle.security.jps.service.policystore.adminroles.AdminRolePrincipal
   </class>
   <uniquename>
      APPID=App1,name="FARM=D.1.2.3,APPLICATION=PolicyServlet,TYPE=OPERATOR"
   </uniquename>
</principal>
```

Parent Element

`<app-role>`, `<member>`, or `<principal>`

Child Elements

None

Occurrence

Optional, zero or one

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                    ...
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}

<jazn-policy> {0 or 1}
```

```
<grant> {0 or more}
   <description> {0 or 1}
   <grantee> {0 or 1}
       <principals> {0 or 1}
          <principal> {0 or more}
             <name> {1}
             <class> {1}
             <uniquename> {0 or 1}
             <guid> {0 or 1}
       <codesource> {0 or 1}
          <url> {1}
   <permissions> {0 or 1}
       <permission> {1 or more}
          <class> {1}
          <name> {0 or 1}
          <actions> {0 or 1}
```

<url>

This element specifies the URL of the code that is granted permissions.

Note the following points:

- URL values cannot be restricted to a single class.
- URL values with ".jar" suffix match the JAR files in the specified directory.
- URL values with "/" suffix match all class files (not JAR files) in the specified directory.
- URL values with "/*" suffix match all files (both class and JAR files) in the specified directory.
- URL values with "/-" suffix match all files (both class and JAR files) in the specified directory and, recursively, all files in subdirectories.
- The system variables `oracle.deployed.app.dir` and `oracle.deployed.app.ext` can be used to specify a URL independent of the platform.

Parent Element

`<codesource>`

Child Elements

None

Occurrence

Required within parent element, one only

```
<jazn-policy> {0 or 1}
   <grant> {0 or more}
      <description> {0 or 1}
      <grantee> {0 or 1}
          <principals> {0 or 1}
             <principal> {0 or more}
                <name> {1}
                <class> {1}
                <uniquename> {0 or 1}
                <guid> {0 or 1}
          <codesource> {0 or 1}
             <url> {1}
      <permissions> {0 or 1}
          <permission> {1 or more}
             <class> {1}
             <name> {0 or 1}
             <actions> {0 or 1}
```

Example

The following example illustrates the use of the system variables `oracle.deployed.app.dir` and `oracle.deployed.app.ext` to specify URLs independent of the server platform.

Suppose an application grant requires a codesource URL that differs with the server platform:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${domain.home}/servers/${weblogic.Name}/tmp/_WL_user/myApp/-</url>
    </codesource>
  </grantee>
  <permissions> ... </permissions>
</grant>
```

Then, using the following system variable settings:

```
-Doracle.deployed.app.dir=${DOMAIN_HOME}/servers/${SERVER_NAME}/tmp/_WL_user
-Doracle.deployed.app.ext=/-
```

the following specification is possible:

```
<grant>
  <grantee>
    <codesource>
      <url>file:${oracle.deployed.app.dir}/<MyApp>${oracle.deployed.app.ext}</url>
    </codesource>
  </grantee>
  <permissions> ... </permissions>
</grant>
```

<user>

This element specifies a user within a realm.

Attributes

Name	Description
`deactivated`	Specifies whether the user is valid or not. 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. Values: `true` or `false` Default: `false`

Parent Element

`<users>`

Child Elements

`<name>`, `<display-name>`, `<description>`, `<guid>`, `<credentials>`

Occurrence

Optional, zero or more

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
        ...
```

Example

See `<jazn-realm>` for examples.

<users>

This element specifies the set of users belonging to a realm.

Parent Element

`<realm>`

Child Elements

`<user>`

Occurrence

Optional, zero or one

```
<jazn-realm>  {0 or 1}
    <realm>  {0 or more}
        <name>  {1}
        <users>  {0 or 1}
            <user>  {0 or more}
                <name>  {1}
                <display-name>  {0 or 1}
                <description>  {0 or 1}
                <guid>  {0 or 1}
                <credentials>  {0 or 1}
        <roles>  {0 or 1}
        ...
```

Example

See `<jazn-realm>` for an example.

<value>

This element specifies a value for an attribute. You can specify additional attributes for application-level roles using the `<extended-attributes>` element.

Parent Element

`<attribute>`

Child Elements

None

Occurrence

Required within the parent element, one only

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                                <value>  {1 or more}
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}
```

Example

```
<app-roles>
   <app-role>
      <name>Knight</name>
      <display-name>Fellowship of the Ring</display-name>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <extended-attributes>
         <attribute>
         <name>SCOPE</name>
         <values>
            <value>Part-I</value>
         </values>
         </attribute>
      </extended-attributes>
   </app-role>
```

<values>

This element specifies a set of values, each of which specify the value of an attribute. An attribute can have more than one value.

Parent Element

`<attribute>`

Child Elements

`<value>`

Occurrence

Required within the parent element, one only

```
<policy-store> {0 or 1}
    <applications> {0 or 1}
        <application> {1 or more}
            <name> {1}
            <description> {0 or 1}
            <app-roles> {0 or 1}
                <app-role> {1 or more}
                    <name> {1}
                    <class> {1}
                    <display-name> {0 or 1}
                    <description> {0 or 1}
                    <guid> {0 or 1}
                    <uniquename> {0 or 1}
                    <extended-attributes> {0 or 1}
                        <attribute> {1 or more}
                            <name> {1}
                            <values> {1}
                                <value>  {1 or more}
                    <members> {0 or 1}
                        <member> {1 or more}
                            <name> {1}
                            <class> {1}
                            <uniquename> {0 or 1}
                            <guid> {0 or 1}
```

Example

```
<app-roles>
   <app-role>
      <name>Knight</name>
      <display-name>Fellowship of the Ring</display-name>
      <class>oracle.security.jps.service.policystore.ApplicationRole</class>
      <extended-attributes>
         <attribute>
         <name>SCOPE</name>
         <values>
            <value>Part-I</value>
         </values>
         </attribute>
      </extended-attributes>
   </app-role>
```

C Oracle Fusion Middleware Audit Framework Reference

This appendix provides reference information for audit reports in the Oracle Fusion Middleware Audit Framework.

Use the information in this chapter for audit record administration and to develop reports from your audit data.

This chapter contains these topics:

Note: This appendix covers reports based on the report template model with Oracle Business Intelligence Publisher 10*g*. A different approach is used for audit based on the dynamic metadata model; see Chapter 15 for details.

C.1 Audit Events

This section describes the components that are audited and the types of events that can be audited.

C.1.1 What Components Can be Audited?

The Oracle Fusion Middleware Audit Framework provides the foundation for auditing by Oracle Fusion Middleware components and applications. In 12*c* (12.1.2), a number of Java and system components can generate audit records; they are known as audit-aware components.

Some examples of Java components that utilize the Fusion Middleware Audit Framework are:

- Directory Integration Platform Server
- Oracle Platform Security Services
- Oracle Web Services Manager
- Oracle Web Services

- Reports Server

Some system components that utilize the Fusion Middleware Audit Framework are:

- Oracle HTTP Server
- Oracle Internet Directory

This appendix provides audit information only for events generated by Oracle Platform Security Services. For details about auditing in other components and applications, refer to the respective administration guides.

C.1.2 What Events can be Audited?

The set of tables in this section shows what event types can be audited:

- Oracle Platform Security Services Events and their Attributes

C.1.2.1 Oracle Platform Security Services Events and their Attributes

This section contains the following tables:

- Table C–2, " Core Oracle Platform Security Services Events"
- Table C–3, " Identity Directory Service Events"
- Table C–4, " Identity Virtualization Library Events"

Table C–1 System Categories and Events

Category	Event	Description
UserSession This set of events is for creating and using user sessions on the system. Common attribute for these events: AuthenticationMethod	UserLogin User Logins	In multi-tier applications, inner tiers often use some special user id (an end user or an administrator) to log in to the next tier. To make audit reports more meaningful, logins by these special users are considered in a separate category - Internal Logins. The User Logins/Logouts events only records actions by regular users (including administrators).
	UserLogout User Logouts	An end user or administrator logs out.
	Authentication	Authentication is very similar to UserLogin/InternalLogin, except that no session is created, so there is no corresponding UserLogout/InternalLogout. This event is usually generated by lower layers, while login is generated by higher layers.
	InternalLogin Internal Login	This is an internal login between two tiers.
	InternalLogout Internal Logout	This is an internal logout between two tiers.
	QuerySession Query Session	Query the attributes within a session object for a logged-in user.
	ModifySession	Modify the attributes within a session object for a logged-in user.

Table C–1 (Cont.) System Categories and Events

Category	Event	Description
Authorization	CheckAuthorization	
This set of events is for authorization.	Check Authorization	
Data Access	CreateDataItem	Create a data item, for example a file.
This set of events is for data access.	Create a data item	
	DeleteDataItem	Delete a data item.
	QueryDataItemAttributes	Query the attributes associated with a data item.
	ModifyDataItemAttributes	Modify the attributes associated with a data item, for example access.
AccountManagement	ChangePassword	Change a user's password.
This set of events is for the management of principal accounts.		
	CreateAccount	Create a user, or group, or any other principal account.
	DeleteAccount	Delete an account for a user, or group, or any other principal.
	EnableAccount	Enable an account for a user, or group, or any other principal
	DisableAccount	Disable an account for a user, or group, or any other principal.
	QueryAccount	Query the user's account.
	ModifyAccount	Modify the account attributes.
ServiceManagement	InstallService	Install or upgrade a service or an application.
This set of events relate to management of system services and applications.		
	RemoveService	De-install a service or an application.
	QueryServiceConfig	Query the configuration of a service or application.
	ModifyServiceConfig	Modify the configuration of a service or application.
	DisableService	Shut down or disable a service or application.
	EnableService	Start up or enable a service or application.
ServiceUtilize	InvokeService	Invoke a service or an application. For example, execute a command-line script.
These events relate to the use of a service or application. They typically map to the execution of a program or procedure, and manipulation of the processing environment.		

Table C–1 (Cont.) System Categories and Events

Category	Event	Description
	TerminateService	Terminate a service or an application, either at the request of the application itself or by intervention of the domain in response to user or administrative action.
	QueryProcessContext	Query the attributes associated with the current processing context.
	ModifyProcessContext	Modify the attributes associated with the current processing context.
PeerAssocManagement This set of events creates and works with communication channels between system components.	CreatePeerAssoc	Creates a communication channel between system components.
	TerminatePeerAssoc	Terminates a communication channel between system components.
	QueryAssocContext	Query attributes associated with a communication channel between system components.
	ModifyAssocContext	Modify attributes associated with a communication channel between system components
		a communication channel between system components
	ReceiveDataViaAssoc	Receive data from an associated peer.
	SendDataViaAssoc	Send data to an associated peer.
DataItemContentAccess This set of events is to form an association between a service or application and a data item or resource element to use its content or services; for example a file or directory, a device special file, a memory segment, communication port, and so on.	CreateDataItemAssoc	Open a data item, for example a file.
	TerminateDataItemAssoc	Close a data item, for example a file.
	QueryDataItemAssocConte xt	Query attributes of a data item, for example mode of access, size limits, access paths, and so on.
	ModifyDataItemAssocConte xt	Modify attributes of a data item.
	QueryDataItemContents	Read the data item.
	ModifyDataItemContent	Write or append to the data item.
Exceptional These events are considered to be outside the generalized events.	StartSystem	Boot a system host.
	ShutdownSystem	Shut down the system.

Table C–1 (Cont.) System Categories and Events

Category	Event	Description
	ResourceExhausted	Resources like data storage or communication endpoints have been exhausted.
	ResourceCorrupted	Resources like data storage have integrity failures.
	BackupDatastore	Make a backup copy of a data store.
	RecoverDatastore	Recover a data store from a backup copy.
AuditService This set of events applies to audit service configuration. Common attribute for these events: TransactionId	ConfigureAuditPolicy	Modify parameters that control auditing, for example the audit event filtering.
	ConfigureAuditRepository	Configure the audit storage type, for example to change from file-based storage to a database.

See Also: Section 13.4.3 for background about system categories and events.

Table C–2 Core Oracle Platform Security Services Events

Event Category	Event Type	Attributes used by Event
Authorization	CheckPermission	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, Subject, PermissionAction, PermissionTarget, PermissionClass
	CheckSubject	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, Subject
	IsAccessAllowed	

Table C–2 (Cont.) Core Oracle Platform Security Services Events

Event Category	Event Type	Attributes used by Event
CredentialManagement	CreateCredential	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, mapName, key, CodeSource, Principals, InitiatorGUID
	DeleteCredential	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, mapName, key, CodeSource, Principals, InitiatorGUID
	AccessCredential	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, mapName, key, CodeSource, Principals, InitiatorGUID
	ModifyCredential	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, mapName, key, CodeSource, Principals, InitiatorGUID
PolicyManagement	PolicyGrant	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, PermissionAction, PermissionTarget, PermissionClass, PermissionScope

Table C–2 (Cont.) Core Oracle Platform Security Services Events

Event Category	Event Type	Attributes used by Event
	PolicyRevoke	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, PermissionAction, PermissionTarget, PermissionClass, PermissionScope
RoleManagement	RoleMembershipAdd	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, ApplicationRole, EnterpriseRoles, PermissionScope
	RoleMembershipRemove	ComponentType, InstanceId, HostId, HostNwaddr, ModuleId, ProcessId, OracleHome, HomeInstance, ECID, RID, ContextFields, SessionId, TargetComponentType, ApplicationName, EventType, EventCategory, EventStatus, TstzOriginating, ThreadId, ComponentName, Initiator, MessageText, FailureCode, RemoteIP, Target, Resource, Roles, CodeSource, Principals, InitiatorGUID, ApplicationRole, EnterpriseRoles, PermissionScope
,RolePolicy Management	RolePolicyCreation	CodeSource, Principals, InitiatorGUID, InitiatorDN, ManagedApplication, PolicyName, PolicyApplicationRolePrincipals, RoleMembers, PolicyRules, ResourceNames, ResourceNameExpressions, PolicyApplicationRolePrincipalsOld, RoleMembersOld, PolicyRulesOld, ResourceNamesOld, ResourceNameExpressionsOld
	,RolePolicyModification	CodeSource, Principals, InitiatorGUID, InitiatorDN, ManagedApplication, PolicyName, PolicyApplicationRolePrincipals, RoleMembers, PolicyRules, ResourceNames, ResourceNameExpressions, PolicyApplicationRolePrincipalsOld, RoleMembersOld, PolicyRulesOld, ResourceNamesOld, ResourceNameExpressionsOld

Table C–2 (Cont.) Core Oracle Platform Security Services Events

Event Category	Event Type	Attributes used by Event
	RolePolicyDeletion	CodeSource, Principals, InitiatorGUID, InitiatorDN, ManagedApplication, PolicyName, PolicyApplicationRolePrincipals, RoleMembers, PolicyRules, ResourceNames, ResourceNameExpressions, PolicyApplicationRolePrincipalsOld, RoleMembersOld, PolicyRulesOld, ResourceNamesOld, ResourceNameExpressionsOld
ResourceManagement	ResourceCreation	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, ResName, ResTypeName, PolicyDomainName, ResourceAttributes, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceAttributesOld, SqlPredicate, SqlPredicateOld, XmlExpression, XmlExpressionOld,
	ResourceDeletion	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, ResName, ResTypeName, PolicyDomainName, ResourceAttributes, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceAttributesOld, SqlPredicate, SqlPredicateOld, XmlExpression, XmlExpressionOld,
	ResourceModification	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, ResName, ResTypeName, PolicyDomainName, ResourceAttributes, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceAttributesOld, SqlPredicate, SqlPredicateOld, XmlExpression, XmlExpressionOld,
,KeyStoreManagement	CreateKeyStore	stripeName, keystoreName, alias, operation, CodeSource, Principals, InitiatorGUID
	DeleteKeyStore	stripeName, keystoreName, alias, operation, CodeSource, Principals, InitiatorGUID
	ModifyKeyStore	stripeName, keystoreName, alias, operation, CodeSource, Principals, InitiatorGUID

Table C–2 (Cont.) Core Oracle Platform Security Services Events

Event Category	Event Type	Attributes used by Event
,PermissionSet Management	PermissionSetCreation	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, PermissionSetName, PolicyDomainName, ResourceActions, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceActionsOld
	PermissionSetDeletion	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, PermissionSetName, PolicyDomainName, ResourceActions, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceActionsOld
	PermissionSetModification	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, PermissionSetName, PolicyDomainName, ResourceActions, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceActionsOld

Table C–3 Identity Directory Service Events

Event Category	Event Type	Attributes used by Event
UserSession	Authentication	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
DataAccess	CreateDataItem	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
	DeleteDataItem	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
	ModifyDataItemAttributes	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod

Table C–4 Identity Virtualization Library Events

Event Category	Eventy Type	Attributes used by Event
LDAPEntryAccess	Add	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource Roles, SessionId, Target, ThreadId, AuthenticationMethod
	Delete	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
	Modify	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
	Rename	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
UserSession	UserLogin.FAILURESONLY	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod
DataAccess	QueryDataItemAttributes	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resourc,e Roles, SessionId, Target, ThreadId, AuthenticationMethod
	ModifyDataItemAttributes	Initiator, EventType, EventStatus, MessageText, ApplicationName, AuditService:TransactionId, ContextFields, ECID, EventCategory, FailureCode, MajorVersion, MinorVersion, RID, RemoteIP, Resource, Roles, SessionId, Target, ThreadId, AuthenticationMethod

C.1.3 OPSS Event Attribute Descriptions

Table C–5 lists all attributes for OPSS audited events. Use this table to learn about the attributes used in the event of interest.

Table C–5 Attributes of OPSS Audit Events

Namespace	Attribute Name	Description
common	ApplicationName	The Java EE application name.

Table C–5 (Cont.) Attributes of OPSS Audit Events

Namespace	Attribute Name	Description
	AuditUser	Identifies the user name of the user who is running the application.
	ComponentData	Where component-specific data are stored when there is no component-specific table in the schema.
	ComponentName	The name of this component.
	ComponentType	Type of the component.
	ContextFields	This attribute contains the context fields extracted from the dms context.
	DomainName	The WebLogic Server or IBM WebSphere Domain.
	ECID	Identifies the thread of execution in which the originating component participates.
	EventCategory	The category of the audit event.
	EventStatus	The outcome of the audit event - success or failure.
	EventType	The type of the audit event. Use the wlst listAuditEvents command to list out all the events.
	FailureCode	The error code in case EventStatus = failure
	HomeInstance	The ORACLE_INSTANCE directory of the component.
	HostId	DNS hostname of originating host.
	HostNwaddr	The IP or other network address of originating host.
	Initiator	Identifies the UID of the user who is doing the operation.
	InstanceId	The name of the Oracle instance to which this component belongs.
	MajorVersion	The major version of a component.
	MessageText	Description of the audit event.
	MinorVersion	The minor version of a component.
	ModuleId	The ID of the module that originated the message. Interpretation is unique within Component ID.
	OracleHome	The ORACLE_HOME directory of the component.
	ProcessId	The ID of the process that originated the message.
	RemoteIP	The IP address of the client initiating this event.
	Resource	Identifies a resource that is being accessed. A resource can be many things - web page, file, directory share, web service, XML document, a portlet. The resource can be named as a combination of a host name, and a URI.

Table C–5 (Cont.) Attributes of OPSS Audit Events

Namespace	Attribute Name	Description
	RID	This is the relationship identifier; it is used to provide the full and correct calling relationships between threads and processes.
	Roles	The roles that the user was granted at the time of login.
	ServerName	The name of the server.
	SessionId	The ID of the login session.
	Target	Identifies the UID of the user on whom the operation is being done. For example, if Alice changes Bob's password, then Alice is the initiator and Bob is the target.
	TargetComponentType	The target component type.
	TstzOriginating	Date and time when the audit event was generated.
	ThreadId	The ID of the thread that generated this event.
	TenantId	The tenant ID.
	TransactionId	The transaction ID.
	UserTenantId	The user tenant ID.
AuditService	TransactionId	The transaction ID.
UserSession	AuthenticationMethod	The Authentication method, namely password, SSL, Kerberos and so on.

See Also: Section 13.4.2 for details about attribute groups and attributes.

C.2 The Audit Schema

You can create custom reports using your choice of reporting tools. For example, while the pre-built reports use a subset of the event attributes, you can make use of the entire audit attribute set for an event in creating custom reports.

Table C–6 and Table C–7 describe the audit schema, which is useful when building custom reports. The `IAU_ID` column in the schema is indexed to enhance query performance.

Table C–6 The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
BASE TABLE	IAU_ID	NUMBER	Yes	1
	IAU_ORGID	VARCHAR2(255 Bytes)	Yes	2
	IAU_COMPONENTID	VARCHAR2(255 Bytes)	Yes	3
	IAU_COMPONENTTYPE	VARCHAR2(255 Bytes)	Yes	4

Table C–6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
	IAU_INSTANCEID	VARCHAR2(255 Bytes)	Yes	5
	IAU_HOSTINGCLIENTID	VARCHAR2(255 Bytes)	Yes	6
	IAU_HOSTID	VARCHAR2(255 Bytes)	Yes	7
	IAU_HOSTNWADDR	VARCHAR2(255 Bytes)	Yes	8
	IAU_MODULEID	VARCHAR2(255 Bytes)	Yes	9
	IAU_PROCESSID	VARCHAR2(255 Bytes)	Yes	10
	IAU_ORACLEHOME	VARCHAR2(255 Bytes)	Yes	11
	IAU_HOMEINSTANCE	VARCHAR2(255 Bytes)	Yes	12
	IAU_UPSTREAMCOMPONENTID	VARCHAR2(255 Bytes)	Yes	13
	IAU_ DOWNSTREAMCOMPONENTID	VARCHAR2(255 Bytes)	Yes	14
	IAU_ECID	VARCHAR2(255 Bytes)	Yes	15
	IAU_RID	VARCHAR2(255 Bytes)	Yes	16
	IAU_CONTEXTFIELDS	VARCHAR2(2000 Bytes)	Yes	17
	IAU_SESSIONID	VARCHAR2(255 Bytes)	Yes	18
	IAU_SECONDARYSESSIONID	VARCHAR2(255 Bytes)	Yes	19
	IAU_APPLICATIONNAME	VARCHAR2(255 Bytes)	Yes	20
	IAU_TARGETCOMPONENTTYPE	VARCHAR2(255 Bytes)	Yes	21
	IAU_EVENTTYPE	VARCHAR2(255 Bytes)	Yes	22
	IAU_EVENTCATEGORY	VARCHAR2(255 Bytes)	Yes	23
	IAU_EVENTSTATUS	NUMBER	Yes	24
	IAU_TSTZORIGINATING	TIMESTAMP(6)	Yes	25
	IAU_THREADID	VARCHAR2(255 Bytes)	Yes	26
	IAU_COMPONENTNAME	VARCHAR2(255 Bytes)	Yes	27
	IAU_INITIATOR	VARCHAR2(255 Bytes)	Yes	28
	IAU_MESSAGETEXT	VARCHAR2(255 Bytes)	Yes	29

Table C–6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
	IAU_FAILURECODE	VARCHAR2(255 Bytes)	Yes	30
	IAU_REMOTEIP	VARCHAR2(255 Bytes)	Yes	31
	IAU_TARGET	VARCHAR2(255 Bytes)	Yes	32
	IAU_RESOURCE	VARCHAR2(255 Bytes)	Yes	33
	IAU_ROLES	VARCHAR2(255 Bytes)	Yes	34
	IAU_AUTHENTICATIONMETHOD	VARCHAR2(255 Bytes)	Yes	35
	IAU_TRANSACTIONID	VARCHAR2(255 Bytes)	Yes	36
	IAU_DOMAINNAME	VARCHAR2(255 Bytes)	Yes	37
	IAU_COMPONENTDATA	clob	yes	38
DIP	IAU_ID	NUMBER	Yes	1
	IAU_TSTZORIGINATING	TIMESTAMP(6)	Yes	2
	IAU_EVENTTYPE	VARCHAR2(255 Bytes)	Yes	3
	IAU_EVENTCATEGORY	VARCHAR2(255 Bytes)	Yes	4
	IAU_ASSOCIATEPROFILENAME	VARCHAR2(512 Bytes)	Yes	5
	IAU_PROFILENAME	VARCHAR2(512 Bytes)	Yes	6
	IAU_ENTRYDN	VARCHAR2(1024 Bytes)	Yes	7
	IAU_PROVEVENT	VARCHAR2(2048 Bytes)	Yes	8
	IAU_JOBNAME	VARCHAR2(128 Bytes)	Yes	9
	IAU_JOBTYPE	VARCHAR2(128 Bytes)	Yes	10
IAU_DISP_ NAME_TL	IAU_LOCALE_STR	VARCHAR2(7 Bytes)		1
	IAU_DISP_NAME_KEY	VARCHAR2(255 Bytes)		2
	IAU_COMPONENT_TYPE	VARCHAR2(255 Bytes)		3
	IAU_DISP_NAME_KEY_TYPE	VARCHAR2(255 Bytes)		4
	IAU_DISP_NAME_TRANS	VARCHAR2(4000 Bytes)	Yes	5

Table C–6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
IAU_LOCALE_MAP_TL	IAU_LOC_LANG	VARCHAR2(2 Bytes)	Yes	1
	IAU_LOC_CNTRY	VARCHAR2(3 Bytes)	Yes	2
	IAU_LOC_STR	VARCHAR2(7 Bytes)	Yes	3

Table C–7 shows additional tables in the audit schema; these tables support the dynamic metadata model.

Table C–7 Additional Audit Schema Tables

Table Name	Column Name	Data Type
IAU_COMMON	IAU_ID	NUMBER
	IAU_OrgId	VARCHAR(255)
	IAU_ComponentId	VARCHAR(255)
	IAU_ComponentType	VARCHAR(255)
	IAU_MajorVersion	VARCHAR(255)
	IAU_MinorVersion	VARCHAR(255)
	IAU_InstanceId	VARCHAR(255)
	IAU_HostingClientId	VARCHAR(255)
	IAU_HostId	VARCHAR(255)
	IAU_HostNwaddr	VARCHAR(255)
	IAU_ModuleId	VARCHAR(255)
	IAU_ProcessId	VARCHAR(255)
	IAU_OracleHome	VARCHAR(255)
	IAU_HomeInstance	VARCHAR(255)
	IAU_UpstreamComponentId	VARCHAR(255)
	IAU_DownstreamComponentId	VARCHAR(255)
	IAU_ECID	VARCHAR(255)
	IAU_RID	VARCHAR(255
	IAU_ContextFields	VARCHAR(2000)
	IAU_SessionId	VARCHAR(255)
	IAU_SecondarySessionId	VARCHAR(255)
	IAU_ApplicationName	VARCHAR(255)
	IAU_TargetComponentType	VARCHAR(255)
	IAU_EventType	VARCHAR(255)
	IAU_EventCategory	VARCHAR(255)
	IAU_EventStatus	NUMBER
	IAU_TstzOriginating	TIMESTAMP

Table C–7 (Cont.) Additional Audit Schema Tables

Table Name	Column Name	Data Type
	IAU_ThreadId	VARCHAR(255)
	IAU_ComponentName	VARCHAR(255)
	IAU_Initiator	VARCHAR(255)
	IAU_MessageText	VARCHAR(2000)
	IAU_FailureCode	VARCHAR(255)
	IAU_RemoteIP	VARCHAR(255)
	IAU_Target	VARCHAR(255)
	IAU_Resource	VARCHAR(255)
	IAU_Roles	VARCHAR(255)
	IAU_AuthenticationMethod	VARCHAR(255)
	IAU_TransactionId	VARCHAR(255)
	IAU_DomainName	VARCHAR(255)
	IAU_ComponentVersion	VARCHAR(255)
	IAU_ComponentData	CLOB
IAU_CUSTOM	IAU_ID	NUMBER
	IAU_BOOLEAN_001 through IAU_BOOLEAN_050	NUMBER
	IAU_INT_001 through IAU_INT_050	NUMBER
	IAU_LONG_001 through IAU_LONG_050	NUMBER
	IAU_FLOAT_001 through IAU_FLOAT_050	NUMBER
	IAU_DOUBLE_001 through IAU_DOUBLE_050	NUMBER
	IAU_STRING_001 through IAU_STRING_100	VARCHAR(2048)
	IAU_DATETIME_001 through IAU_DATETIME_050	TIMESTAMP
	IAU_LONGSTRING_001 through IAU_LONGSTRING_050	CLOB
	IAU_BINARY_001 through IAU_BINARY_050	BLOB
IAU_AuditService	IAU_ID	NUMBER

Table C–7 (Cont.) Additional Audit Schema Tables

Table Name	Column Name	Data Type
	IAU_TransactionId	VARCHAR(255)
IAU_USERSESSION	IAU_ID	NUMBER
	IAU_AuthenticationMethod	VARCHAR(255)

C.3 WLST Commands for Auditing

Oracle WebLogic Server scripts are used at the command line to administer various features. `WLST` is the command-line utility for administration of Oracle Fusion Middleware components and applications in the Oracle WebLogic Server environment. It provides another option for administration in addition to Oracle Enterprise Manager Fusion Middleware Control.

For details about the WLST commands to view and manage audit policies and the audit store configuration, see "Audit Configuration Commands" in the Oracle Fusion Middleware Infrastructure Security WLST Command Reference.

Note: When running audit commands, you must invoke the `WLST` script from the Oracle Common home. See "Using Custom WLST Commands" in the *Oracle Fusion Middleware Administrator's Guide* for more information.

C.4 Audit Filter Expression Syntax

When you select a custom audit policy, you have the option of specifying a filter expression along with an event.

For example, you can use the following expression:

```
Host Id -eq "myhost123"
```

to enable the audit event for a particular host only.

You enter this expression either through the Fusion Middleware Control Edit Filter Dialog or through the `setAuditPolicy` command.

See Also:

- Section 14.3.1, "Managing Audit Policies for Java Components with Fusion Middleware Control"
- Section 14.3.2, "Managing Audit Policies for System Components with Fusion Middleware Control"

There are some syntax rules you should follow when creating a filter expression.

The expression can either be a Boolean expression or a literal.

```
<Expr> ::= <BooleanExpression> | <BooleanLiteral>
```

A boolean expression can use combinations of RelationalExpression with –and, -or, -not and parenthesis. For example, (`Host Id -eq "stadl17" -or "`).

```
<BooleanExpression> ::=  <RelationalExpression>
   | "(" <BooleanExpression> ")"
   | <BooleanExpression> "-and" <BooleanExpression>
   | <BooleanExpression> "-or" <BooleanExpression>
   | "-not" <BooleanExpression>
```

A relational expression compares an attribute name (on the left hand side) with a literal (on the right-hand side). The literal and the operator must be of the correct data type for the attribute.

```
<RelationalExpression> ::= <AttributeName> <RelationalOperator> <Literal>
```

Relational operators are particular to data types:

- -eq, -ne can be used with all data types
- -contains, -startswith, -endswith can be only used with strings
- -contains_case, -startswith_case and -endswith_case are case sensitive versions of the above three functions
- -lt, -le, -gt, -ge can be used with numeric and datetime

```
<RelationalOperator> : = "-eq" | "-ne" | "-lt" | "-le" | "-gt" | "-ge"
   | "-contains" | "-contains_case"
   | "-startswith" | "-startswith_case"
   | "-endswith" | "-endswith_case"
```

Rules for literals are as follows:

- Boolean literals are true or false, without quotes.
- Date time literals have to be in double quotes and can be in many different formats; "June 25, 2006", "06/26/2006 2:00 pm" are all valid.
- String literals have to be quotes, back-slash can be used to escape an embedded double quote.
- Numeric literals are in their usual format.

For example:

```
<Literal> ::=  <NumericLiteral> | <BooleanLiteral> | <DateTimeLiteral> |
<StringLiteral>
<BooleanLiteral> ::= "true" | "false"
```

C.5 Naming and Logging Format of Audit Files

This section explains the rules that are used to maintain audit files.

For Java components (both Java EE and Java SE) the `audit.log` file contains audit records and comprises the bus-stop file.

When that file fills up (it reaches the configured maximum audit file size which is 100MB), it is renamed to `audit1.log` and records written to a new audit.log. When this file fills up, the audit.log file is renamed to "audit2.log" and the cycle starts with a new audit.log.

When you configure a database audit store, the audit loader reads these files and transfers the records to the database in batches. After reading a complete audit<n>.log file, it deletes the file.

Note: The audit loader never deletes the "current" file, that is, audit.log; it only deletes archive files audit<n>.log.

System components follow the same model, except the file name is slightly different. The process ID is embedded in the file name; thus, if the process id is 11925 the current file is called `audit-pid11925.log`, and after rotation it is called `audit-pid11925-1.log`.

For applications with audit definitions in the dynamic model, the file name format is audit_*major version number*_*minor version number*.log; for example, `audit_1_2.log`.

Here is a sample audit.log file:

```
#Fields:Date Time Initiator EventType EventStatus MessageText AuditUser
ApplicationName AuditService:TransactionId ContextFields DomainName ECID
EventCategory FailureCode HomeInstance HostId HostNwaddr JPS:AccessResult
JPS:AclParameters JPS:AclParametersOld JPS:ActionCollName JPS:ActionCollRefs
JPS:ActionCollRefsOld JPS:ActionConstraint JPS:ActionConstraintOld
JPS:AdminRoleName JPS:Advices JPS:AdvicesOld JPS:AnonymousRole JPS:AnonymousUser
JPS:AppContext JPS:ApplicableDBRes JPS:ApplicableDBResOld JPS:ApplicationRole
JPS:AttrCollName JPS:AuthenticatedRole JPS:Cascade JPS:CodeSource
JPS:CodeSourceTarget JPS:CodeSourceTargetOld JPS:CombiningAlgorithmID
JPS:Condition JPS:ConditionOld JPS:ConfigurationId JPS:Direction JPS:Efect
JPS:EfectOld JPS:EnterpriseRoles JPS:EnvironmentConstraint
JPS:EnvironmentConstraintOld JPS:Flush JPS:GUID JPS:HandlerFunction
JPS:HandlerFunctionOld JPS:ImpliedActions JPS:ImpliedActionsOld JPS:InitiatorDN
JPS:InitiatorGUID JPS:ManagedApplication JPS:ModifiedAttributeName
JPS:ModifiedAttributeValue JPS:ModifiedAttributeValueOld JPS:Obligations
JPS:ObligationsOld JPS:PdpAddress JPS:PermSets JPS:PermSetsOld JPS:Permission
JPS:PermissionAction JPS:PermissionCheckResult JPS:PermissionClass
JPS:PermissionScope JPS:PermissionSetName JPS:PermissionTarget
JPS:PoliciesAndPolicySets JPS:PoliciesAndPolicySetsOld
JPS:PolicyApplicationRolePrincipals JPS:PolicyApplicationRolePrincipalsOld
JPS:PolicyCategory JPS:PolicyCodeSource JPS:PolicyCodeSourceOld
JPS:PolicyCombiningAlgorithmID JPS:PolicyCombiningAlgorithmIDOld
JPS:PolicyDefaults JPS:PolicyDefaultsOld JPS:PolicyDomainName JPS:PolicyIssuer
JPS:PolicyIssuerOld JPS:PolicyName JPS:PolicyPrincipals JPS:PolicyPrincipalsOld
JPS:PolicyRuleName JPS:PolicyRules JPS:PolicyRulesOld JPS:PolicySemantic
JPS:PolicySetDefaults JPS:PolicySetName JPS:PolicySetRef JPS:PolicyType
JPS:PrincipalConstraint JPS:PrincipalConstraintOld JPS:Principals
JPS:PrincipalsTarget JPS:PrincipalsTargetOld JPS:PurgeTime JPS:ResName
JPS:ResTypeName JPS:ResourceActions JPS:ResourceActionsOld JPS:ResourceAttributes
JPS:ResourceAttributesOld JPS:ResourceConstraint JPS:ResourceConstraintOld
JPS:ResourceNameExpressions JPS:ResourceNameExpressionsOld JPS:ResourceNames
JPS:ResourceNamesOld JPS:ResourceType JPS:RoleMembers JPS:RoleMembersOld
JPS:RuleCombiningAlgorithmID JPS:RuleCombiningAlgorithmIDOld JPS:RuntimeAction
JPS:RuntimeResource JPS:SqlPredicate JPS:SqlPredicateOld JPS:Subject JPS:Version
JPS:VersionOld JPS:XmlExpression JPS:XmlExpressionOld JPS:alias
JPS:credStoreContext JPS:key JPS:keystoreName JPS:mapName JPS:operation
JPS:stripeName MajorVersion MinorVersion RID RemoteIP Resource Roles ServerName
SessionId Target TargetComponentType TenantId ThreadId TransactionId
UserSession:AuthenticationMethod UserTenantId
#Remark Values:ComponentType="JPS" ReleaseVersion="MAIN"
2014-04-11 19:17:04.450  "biauthoruser1" "CreateKeyStore" true "Created keystore
kk in stripe ss" "biauthoruser1" "opsscactus" - - "jrfServer_domain"
"824656f5-097d-4d78-a29d-30c16132a56e-00000065,0" "KeyStoreManagement" - -
"testnd01" "10.240.97.183" - - - - - - - - - - - - - - - - - - - - -
"file:/scratch/example/view_storage/example_
e51/work/utp/testout/functional/jps/wls-jrfServer/servers/jrfServer_admin/tmp/_WL_
```

```
user/opsscactus/tfuchd/war/WEB-INF/lib/_wl_cls_gen.jar" - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - "" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- "kk" - - "ss" "1" "2" - - - - "jrfServer_admin" - - - "cisco" "88" - - -
2014-04-11 19:17:04.470  "biauthoruser1" "AccessKeyStore" true "Successfully got
the handle to keystore castore in stripe system" "biauthoruser1" "opsscactus" - -
"jrfServer_domain" "824656f5-097d-4d78-a29d-30c16132a56e-00000065,0"
"KeyStoreManagement" - - "testnd01" "10.240.97.183" - - - - - - - - - - - - - - -
- - - - - "file:/scratch/example/view_storage/example_
e51/work/utp/testout/functional/jps/wls-jrfServer/servers/jrfServer_admin/tmp/_WL_
user/opsscactus/tfuchd/war/WEB-INF/lib/_wl_cls_gen.jar" - - - - - - - - - - - - -
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- - - - - - - - - "" - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
- "castore" - - "system" "1" "2" - - - - "jrfServer_admin" - - - "cisco" "88" - -
```

This file follows the W3C extended logging format, which is a very common log format that is used by many Web Servers including Apache and IIS:

- The first line is a "#Fields" line; it specifies all the fields in the rest of the file.
- The second line is a comment like "#Remark" which has a comment indicating some common attributes like the ComponentType.
- All subsequent lines are data lines; they follow the exact format defined in the "#Fields" line. All attributes are separated by spaces, mussing attributes are indicated by a dash.

D

User and Role API Reference

This appendix contains reference information that you will need when developing applications for LDAP directories based on the User and Role APIs.

Note: The User and Role APIs are deprecated. Applications using the User and Role APIs should migrate to Identity Directory Service API. For details, see Chapter 21.

This appendix includes the following sections:

- Mapping User Attributes to LDAP Directories
- Mapping Role Attributes to LDAP Directories
- Default Configuration Parameters
- Secure Connections for Microsoft Active Directory

See Also: Chapter 20, "Developing with the User and Role API"

Note: IBM Tivoli directory parameters are the same as those specified for openLDAP.

Microsoft ADAM parameters are the same as those specified for Microsoft Active Directory.

D.1 Mapping User Attributes to LDAP Directories

Table D–1 lists each user attribute in UserProfile.property and its corresponding attribute in the different directory servers.

Table D–1 User Attributes in UserProfile.Property

Attribute	Oracle Internet Directory	Oracle WebLogic Server Embedded LDAP	Microsoft Active Directory	Oracle Directory Server Enterprise Edition	Novell eDirectory	OpenLDAP
GUID	orclguid	uid	objectguid	nsuniqueid	guid	entryuuid
USER_ID	username (see Note below)	uid	uid	uid	uid	uid
DISPLAY_NAME	displayname	displayname	displayname	displayname	displayname	displayname

Table D–1 (Cont.) User Attributes in UserProfile.Property

Attribute	Oracle Internet Directory	Oracle WebLogic Server Embedded LDAP	Microsoft Active Directory	Oracle Directory Server Enterprise Edition	Novell eDirectory	OpenLDAP
BUSINESS_EMAIL	mail	mail	mail	mail	mail	mail
DESCRIPTION	description	description	description	description	description	description
EMPLOYEE_TYPE	employeeType	employeeType	employeeType	employeeType	employeeType	employeeType
DEPARTMENT	departmentnumber	departmentnumber	departmentnumber	departmentnumber	departmentnumber	departmentnumber
DATE_OF_BIRTH	orcldateofbirth	-	-	-	-	-
BUSINESS_FAX	facsimiletelephonenumber	facsimiletelephonenumber	facsimiletelephonenumber	facsimiletelephonenumber	facsimiletelephonenumber	facsimiletelephonenumber
BUSINESS_CITY	l	l	l	l	l	l
BUSINESS_COUNTRY	c	c	c	c	c	c
DATE_OF_HIRE	orclhiredate	-	-	-	-	-
NAME	cn	uid	cn	uid	cn	cn
PREFERRED_LANGUAGE	Preferredlanguage	preferredlanguage	preferredlanguage	preferredlanguage	preferredlanguage	preferredlanguage
BUSINESS_POSTAL_ADDR	postaladdress	postaladdress	postaladdress	postaladdress	postaladdress	postaladdress
MIDDLE_NAME	orclmiddlename	-	-	-	-	-
ORGANIZATIONAL_UNIT	ou	ou	ou	ou	ou	ou
WIRELESS_ACCT_NUMBER	orclwirelessaccountnumber	-	-	-	-	-
BUSINESS_PO_BOX	postofficebox	postofficebox	postofficebox	postofficebox	postofficebox	postofficebox
BUSINESS_STATE	St	st	st	st	st	st
HOME_ADDRESS	Homepostaladdress	homepostaladdress	homepostaladdress	homepostaladdress	homepostaladdress	homepostaladdress
NAME_SUFFIX	Generationqualifier	generationqualifier	generationqualifier	generationqualifier	generation qualifier	generationqualifier
BUSINESS_STREET	street	street	street	street	street	street
INITIALS	initials	initials	initials	initials	initials	initials
USER_NAME	username (see Note below)	uid	samaccountname	uid	uid	uid

Table D–1 (Cont.) User Attributes in UserProfile.Property

Attribute	Oracle Internet Directory	Oracle WebLogic Server Embedded LDAP	Microsoft Active Directory	Oracle Directory Server Enterprise Edition	Novell eDirectory	OpenLDAP
BUSINESS_ POSTAL_ CODE	postalcode	postalcode	postalcode	postalcode	postalcode	postalcode
BUSINESS_ PAGER	pager	pager	pager	pager	pager	pager
LAST_ NAME	sn	sn	sn	sn	sn	sn
BUSINESS_ PHONE	telephonenu mber	telephonenu mber	telephonenum ber	telephonenu mber	telephonen umber	telephonenu mber
FIRST_ NAME	givenname	givenname	givenname	givenname	givenname	givenname
TIME_ ZONE	orcltimezone	-	-	-	-	-
MAIDEN_ NAME	orclmaidenn ame	-	-	-	-	-
PASSWOR D	userpassswo rd	userpassswo rd	userpasssword	userpassswo rd	userpasssw ord	userpassswo rd
DEFAULT_ GROUP	orcldefaultpr ofilegroup	-	-	-	-	-
ORGANIZ ATION	o	o	o	o	o	o
HOME_ PHONE	homephone	homephone	homephone	homephone	homephon e	homephone
BUSINESS_ MOBILE	mobile	mobile	mobile	mobile	mobile	mobile
UI_ ACCESS_ MODE	orcluiaccessi bilitymode	-	-	-	-	-
JPEG_ PHOTO	jpegphoto	jpegphoto	jpegphoto	jpegphoto	jpegphoto	jpegphoto
MANAGE R	manager	manager	manager	manager	manager	manager
TITLE	title	title	title	title	title	title
EMPLOYE E_ NUMBER	employeenu mber	employeenu mber	employeenum ber	employeenu mber	employeen umber	employeenu mber
LDUser.PA SSWORD	userpasswor d	userpasswor d	userpassword	userpasswor d	userpasswo rd	userpasswor d

Note: username* : typically uid, but technically, the attribute designated by the orclCommonNicknameAttribute in the subscriber's oraclecontext products common entry.

D.2 Mapping Role Attributes to LDAP Directories

Table D–2 lists each role attribute in UserProfile.property and its corresponding attribute in different directory servers.

Table D–2 Role Attribute Values in LDAP Directories

Role Attribute	Oracle Internet Directory	Oracle WebLogic Server Embedded LDAP	Microsoft Active Directory	Oracle Directory Server Enterprise Edition	Novell eDirectory	OpenLDAP
DISPLAY_NAME	displayname	-	displayname	displayname	displayname	displayname
MANAGER	-	-	-	-	-	-
NAME	cn	cn	cn	cn	cn	cn
OWNER	owner	owner	-	Owner	-	owner
GUID	orclguid	cn	objectguid	NSuniqueid	guid	entryuuid

D.3 Default Configuration Parameters

This section lists parameters for which the APIs can use default configuration values, and the source of the value in different directory servers.

Table D–3 lists the source for Oracle Internet Directory and Microsoft Active Directory.

Table D–3 Default Values - Oracle Internet Directory and Microsoft Active Directory

Parameter	Oracle Internet Directory	Active Directory
RT_USER_OBJECT_CLASSES	#config	{"user" }
RT_USER_MANDATORY_ATTRS	#schema	#schema
RT_USER_CREATE_BASES	#config	cn=users,<subscriberDN>
RT_USER_SEARCH_BASES	#config	<subscriberDN>
RT_USER_FILTER_OBJECT_CLASSES	#config	{"user"}
RT_USER_SELECTED_CREATE_BASE	#config	cn=users,<subscriberDN>
RT_GROUP_OBJECT_CLASSES	#config	{"group" }
RT_GROUP_MANDATORY_ATTRS	#schema	#schema
RT_GROUP_CREATE_BASES	#config	<subscriberDN>
RT_GROUP_SEARCH_BASES	#config	<subscriberDN>
RT_GROUP_FILTER_OBJECT_CLASSES	#config	{"group"}
RT_GROUP_MEMBER_ATTRS	"uniquemember", "member"	"member"
RT_GROUP_SELECTED_CREATE_BASE	#config	<subscriberDN>
RT_GROUP_GENERIC_SEARCH_BASE	<subscriber-DN>	<subscriberDN>
RT_SEARCH_TYPE	#config	#config
ST_SUBSCRIBER_NAME	#config	NULL

Table D–3 (Cont.) Default Values - Oracle Internet Directory and Microsoft Active

Parameter	Oracle Internet Directory	Active Directory
ST_USER_NAME_ATTR	#config	cn
ST_USER_LOGIN_ATTR	#config	samaccountname
ST_GROUP_NAME_ATTR	#config	cn
ST_MAX_SEARCHFILTER_LENGTH	500	500
ST_BINARY_ATTRIBUTES	Choose a Binary Basic Attribute (BBA) See note below about BBAs.	Binary Basic Attribute (BBA)+ { "objectguid" , "unicodepwd" } See note below about BBAs.
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Notes:

- The Basic Binary Attributes include: {"photo", "personalsignature", "audio","jpegphoto", "Java SErializeddata", "thumbnailphoto", "thumbnaillogo", "userpassword", "usercertificate", "cacertificate", "authorityrevocationlist", "certificaterevocationlist", "crosscertificatepair", "x500UniqueIdentifier"}
- #config is extracted from the meta information present in the directory
- #schema is extracted from the schema in the directory

Table D–4 lists the source for Oracle Directory Server Enterprise Edition and Novell eDirectory.

Table D–4 Default Values - Oracle Directory Server Enterprise Edition and Novell eDirectory

Parameter	Oracle Directory Server Enterprise Edition	Novell eDirectory
RT_USER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson" }	{"person","inetorgperson", "organizationalPerson", "ndsloginproperties" }
RT_USER_MANDATORY_ATTRS	#schema	#schema
RT_USER_CREATE_BASES	ou=people,<subscriberDN>	ou=users,<subscriberDN>
RT_USER_SEARCH_BASES	<subscriberDN>	<subscriberDN>
RT_USER_FILTER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson" }	{"person","inetorgperson", "organizationalPerson", "ndsloginproperties" }
RT_USER_SELECTED_CREATE_BASE	ou=people,<subscriberDN>	ou=users,<subscriberDN>
RT_GROUP_OBJECT_CLASSES	"groupofuniquenames"	{"group" }

Table D–4 (Cont.) Default Values - Oracle Directory Server Enterprise Edition and Novell eDirectory

Parameter	Oracle Directory Server Enterprise Edition	Novell eDirectory
RT_GROUP_MANDATORY_ATTRS	#schema	#schema
RT_GROUP_CREATE_BASES	ou=groups,<subscriberDN>	ou=groups,<subscriberDN>
RT_GROUP_SEARCH_BASES	<subscriberDN>	<subscriberDN>
RT_GROUP_FILTER_OBJECT_CLASSES	{"groupofuniquenames"}	{"group"}
RT_GROUP_MEMBER_ATTRS	"uniquemember"	"member"
RT_GROUP_SELECTED_CREATE_BASE	ou=groups,<subscriberDN>	ou=groups,<subscriberDN>
RT_GROUP_GENERIC_SEARCH_BASE	<subscriber-DN>	<subscriberDN>
RT_SEARCH_TYPE	#config	#config
ST_SUBSCRIBER_NAME	NULL	NULL
ST_USER_NAME_ATTR	uid	cn
ST_USER_LOGIN_ATTR	uid	cn
ST_GROUP_NAME_ATTR	cn	cn
ST_MAX_SEARCHFILTER_LENGTH	500	500
ST_BINARY_ATTRIBUTES	Choose a Binary Basic Attribute (BBA) See note below about BBAs.	Binary Basic Attribute (BBA)+ { "guid"} See note below about BBAs.
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Notes:

- The Basic Binary Attributes include: {"photo", "personalsignature", "audio","jpegphoto", "Java SErializeddata", "thumbnailphoto", "thumbnaillogo", "userpassword", "usercertificate", "cacertificate", "authorityrevocationlist", "certificaterevocationlist", "crosscertificatepair", "x500UniqueIdentifier"}
- #config is extracted from the metainformation present in the directory
- #schema is extracted from the schema in the directory

Table Table D–5 lists the parameters for OpenLDAP and Oracle Virtual Directory.

Table D–5 Default Values - OpenLDAP and Oracle Virtual Directory

Parameter	OpenLDAP	Oracle Virtual Directory
RT_USER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson" }	{"inetorgperson"}
RT_USER_MANDATORY_ATTRS	#schema	#schema
RT_USER_CREATE_BASES	ou=people,<subscriberDN>	<subscriberDN>
RT_USER_SEARCH_BASES	<subscriberDN>	<subscriberDN>
RT_USER_FILTER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson" }	{"inetorgperson"}
RT_USER_SELECTED_CREATE_BASE	ou=people,<subscriberDN>	<subscriberDN>
RT_GROUP_OBJECT_CLASSES	"groupofuniquenames"	{"groupofuniquenames"}
RT_GROUP_MANDATORY_ATTRS	#schema	#schema
RT_GROUP_CREATE_BASES	ou=groups,<subscriberDN>	<subscriberDN>
RT_GROUP_SEARCH_BASES	<subscriberDN>	<subscriberDN>
RT_GROUP_FILTER_OBJECT_CLASSES	"groupofuniquenames"	{"groupofuniquenames"}
RT_GROUP_MEMBER_ATTRS	"uniquemember"	"uniquemember"
RT_GROUP_SELECTED_CREATE_BASE	ou=groups,<subscriberDN>	<subscriberDN>
RT_GROUP_GENERIC_SEARCH_BASE	<subscriber-DN>	<subscriberDN>
RT_SEARCH_TYPE	#config	#config
ST_SUBSCRIBER_NAME	NULL	#config (namingcontexts)
ST_USER_NAME_ATTR	uid	cn
ST_USER_LOGIN_ATTR	uid	cn
ST_GROUP_NAME_ATTR	cn	cn
ST_MAX_SEARCHFILTER_LENGTH	500	500
ST_BINARY_ATTRIBUTES	Choose a Binary Basic Attribute (BBA) See note below about BBAs.	Binary Basic Attribute (BBA)+ { "guid"} See note below about BBAs.
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Notes:

- The Basic Binary Attributes include: {"photo", "personalsignature", "audio","jpegphoto", "Java SErializeddata", "thumbnailphoto", "thumbnaillogo", "userpassword", "usercertificate", "cacertificate", "authorityrevocationlist", "certificaterevocationlist", "crosscertificatepair", "x500UniqueIdentifier"}
- #config is extracted from the meta information present in the directory
- #schema is extracted from the schema in the directory

Table D–6 lists the parameters for Oracle WebLogic Server LDAP.

Table D–6 Default Values - Oracle WebLogic Server LDAP

Parameter	Oracle WebLogic Server Embedded LDAP
RT_USER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson", "wlsUser"}
RT_USER_MANDATORY_ATTRS	#schema
RT_USER_CREATE_BASES	{"ou=people,<subscriberDN>"}
RT_USER_SEARCH_BASES	{"ou=people,<subscriberDN>"}
RT_USER_FILTER_OBJECT_CLASSES	{"inetorgperson", "wlsUser"}
RT_USER_SELECTED_CREATE_BASE	ou=people,<subscriberDN>
RT_GROUP_OBJECT_CLASSES	{"top","groupofuniquenames","groupOfURLs"}
RT_GROUP_MANDATORY_ATTRS	#schema
RT_GROUP_CREATE_BASES	{"ou=groups,<subscriberDN>"}
RT_GROUP_SEARCH_BASES	{"ou=groups,<subscriberDN>"}
RT_GROUP_FILTER_OBJECT_CLASSES	{"top","groupofuniquenames","groupOfURLs"}
RT_GROUP_MEMBER_ATTRS	"uniquemember"
RT_GROUP_SELECTED_CREATE_BASE	ou=groups,<subscriberDN>
RT_GROUP_GENERIC_SEARCH_BASE	<subscriberDN>
RT_SEARCH_TYPE	#config
ST_SUBSCRIBER_NAME	#config (namingcontexts)
ST_USER_NAME_ATTR	uid
ST_USER_LOGIN_ATTR	uid
ST_GROUP_NAME_ATTR	cn
ST_MAX_SEARCHFILTER_LENGTH	500

Table D–6 (Cont.) Default Values - Oracle WebLogic Server LDAP

Parameter	Oracle WebLogic Server Embedded LDAP
ST_BINARY_ATTRIBUTES	*(BBA) See note below about BBAs.
ST_LOGGER_NAME	oracle.idm.userrole

D.4 Secure Connections for Microsoft Active Directory

Active Directory requires connections to be SSL-enabled when setting sensitive information like passwords. Therefore, operations like creating a user (which set the password) will not succeed if the connection is not SSL-enabled.

E Administration with Scripting and MBean Programming

This appendix describes advanced administrative tasks carried out with WLST commands and MBean programming.

It includes the following sections:

- Configuring OPSS Service Provider Instances with a Script
- Configuring OPSS Services with MBeans
- Restricting Access

E.1 Configuring OPSS Service Provider Instances with a Script

If your application uses the User and Role API and must access an authenticator user attribute *different* from the default attribute (which is cn), then using the WebLogic Administration Console, you would configure the authenticator to use the desired user attribute. But for the User and Role API to use an attribute different from the default, the authenticator must be, in addition, properly initialized.

The procedure below explains how to use a script to change the authenticator initialization, so that the User and Role API uses the configured user attribute to access data in the configured authenticator.

For details about WebLogic scripting, see *Understanding the WebLogic Scripting Tool*.

To add or update custom properties of a service instance, proceed as follows:

1. Create a py script file with the following content:

```
import sys
connect('userName', 'userPassword', 'url', 'adminServerName')
domainRuntime()

val = None
key = None
si = None
for  i in range(len(sys.argv)):
    if sys.argv[i] == "-si":
        si = sys.argv[i+1]
    if sys.argv[i] == "-key":
        key  = sys.argv[i+1]
    if sys.argv[i] == "-value":
        val = sys.argv[i+1]

on = ObjectName("com.oracle.jps:type=JpsConfig")
```

```
sign = ["java.lang.String","java.lang.String","java.lang.String"]
params = [si,key,val]
mbs.invoke(on, "updateServiceInstanceProperty", params, sign)
mbs.invoke(on, "persist", None, None)
```

2. In the produced script, replace *userName*, *userPass*, *localHost*, and *portNumber* by the appropriate strings to connect to the administration server in the domain you are interested. Note that the use of `connect` requires that the server to which you want to connect be up and running when the script is invoked.

   Let's assume that the script is saved in the file `/tmp/updateServiceInstanceProperty.py`.

3. Change to the directory `$ORACLE_HOME/common/bin`, which should contain the file `wlst.sh`:

   ```
   >cd $ORACLE_HOME/common/bin
   ```

4. Run the following command:

   ```
   >wlst.sh /tmp/updateServiceInstanceProperty.py -si servInstName -key propKey
   -value propValue
   ```

   Where:

   - *servInstName* is the name of the service instance provider whose properties are to be modified.
   - *propKey* identifies the name of the property to insert or modify.
   - *propValue* is the name of the value to add or update.

   Any argument containing a space character must be enclosed with double quotes.

   Each invocation of the above command modifies the domain configuration file `$DOMAIN_HOME/config/fmwconfig/jps-config.xml` by adding or updating a property to the passed instance provider. If the passed key matches the name of an existing property, then that property is updated with the passed value.

5. Restart the Oracle WebLogic server: the changes in configuration are not in effect until the server has been restarted.

Example of Use

Assume that the domain configuration file contains an authenticator named `idstore.ldap`. Then the following invocation:

```
wlst.sh /tmp/updateServiceInstanceProperty.py -si idstore.ldap -key "myPropName"
-value "myValue"
```

adds (or updates) the specified property of that instance provider as illustrated in the following snippet:

```
<serviceInstance provider="idstore.ldap.provider" name="idstore.ldap">
   ...
   <property name="myPropName" value="myValue"/>
   ...
</serviceInstance>
```

When the authenticator is initialized with the above configuration, the User and Role API can use the user attribute `mail` to access user information in this authenticator.

E.2 Configuring OPSS Services with MBeans

Oracle Platform Security Services provides a set of JMX-compliant Java EE Beans that are used by Oracle Enterprise Manager Fusion Middleware Control and OPSS security scripts to manage, configure, and monitor Oracle Platform Security Services.

The use of MBeans is recommended in Java EE applications only.

Links to OPSS API javadocs, including the OPSS MBeans API javadoc, are available in Section G.1, "OPSS API References."

This section addresses the following topics:

- List of Supported OPSS MBeans
- Invoking an OPSS MBean
- Programming with OPSS MBeans

E.2.1 List of Supported OPSS MBeans

Table E–1 lists the supported MBeans, their basic function, and the object name to use in custom scripts or Java SE programs to perform a task:

Table E–1 List of OPSS MBeans

MBean	Function	MBeanServer Connection Name
Jps Configuration	Manages domain configuration data, that is in the file `jps-config.xml`. This MBean provides the only way to modify configuration data. Update or write operations require server restart to effect changes.	`com.oracle.jps:type=JpsConfig`
Credential Store	Manages credential data, that is, the store service configured in the default context. Update or write operations do not require server restart to effect changes. All changes are effected immediately. Access is restricted to administrators only.	`com.oracle.jps:type=JpsCredentialStore`
Global Policy Store	Manages global policies in the policy store configured in the default context. Update or write operations do not require server restart to effect changes. All changes are effected immediately.	`com.oracle.jps:type=JpsGlobalPolicyStore`
Application Policy Store	Manages application policies in the policy store configured in the default context. Update or write operations do not require server restart to effect changes. All changes are effected immediately.	`com.oracle.jps:type=JpsApplicationPolicyStore`
Administration Policy Store	Validates whether a user logged into the current JMX context belongs to a particular role. It does not facilitate any configuration modifications.	`com.oracle.jps:type=JpsAdminPolicyStore`

E.2.2 Invoking an OPSS MBean

There are two basic ways to invoke an OPSS MBean:

- To write a script and run it using the Oracle WebLogic scripting tool; for details, see section Navigating MBeans (WLST Online) in *Understanding the WebLogic Scripting Tool*.
- To write a Java program; Section E.2.3, "Programming with OPSS MBeans" contains a sample program illustrating this approach.

Note: An alternative way to invoke an MBean is using the MBean browser in Fusion Middleware Control. This approach, however, allows only a limited number of operations and it involves composite data creation.

To access this browser, login to Fusion Middleware Control and then proceed as follows:

1. Select the menu item **AdminServer > System MBean Browser**, in the appropriate domain, to display the page **System MBean Browser**.
2. In the pane where the hierarchy is displayed, expand the nodes **Application Defined MBeans**, **com.oracle.jps**, and **Domain**: *myDomain* (where *myDomain* stands for the name of your domain); this last one has under it one node per OPSS MBean.
3. After expanding any of those nodes, select an item, that is an MBean, and user the tabs **Attributes**, **Operations**, and **Notifications** in the right pane to inspect current attribute values or to invoke methods in the selected MBean.

For example, the Jps Configuration MBean is found at the following location in this hierarchy:

```
Application Defined
MBeans/com.oracle.jps/Domain:myDomain/JpsConfig/JpsConfig
```

For complete details about this browser, see the Fusion Middleware Control online help system.

E.2.3 Programming with OPSS MBeans

The following code sample illustrates how to invoke the Jps Configuration MBean over the WebLogic Server t3 protocol; in this sample, note the following important points:

- It assumes that the following JAR files are in the class path:
  - `$ORACLE_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-api.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.jps_12.1.3/jps-mbeans.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.jmx_12.1.3/jmxframework.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.idm_12.1.3/identitystore.jar`
  - `$WEBLOGIC_HOME/server/lib/wljmxclient.jar`
- The connection is established by the method `init`.
- Any update operation is followed by a call to persist.

```
import java.io.IOException;
import java.net.MalformedURLException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.HashMap;
import java.util.List;

import javax.management.InstanceNotFoundException;
import javax.management.MBeanException;
import javax.management.MBeanServerConnection;
import javax.management.MalformedObjectNameException;
import javax.management.ObjectName;
import javax.management.ReflectionException;
import javax.management.openmbean.CompositeData;
import javax.management.remote.JMXConnector;
import javax.management.remote.JMXConnectorFactory;
import javax.management.remote.JMXServiceURL;
import javax.naming.Context;

import oracle.security.jps.mas.mgmt.jmx.credstore.PortableCredential;
import oracle.security.jps.mas.mgmt.jmx.credstore.PortablePasswordCredential;
import oracle.security.jps.mas.mgmt.jmx.policy.PortableApplicationRole;
import oracle.security.jps.mas.mgmt.jmx.policy.PortableCodeSource;
import oracle.security.jps.mas.mgmt.jmx.policy.PortableGrant;
import oracle.security.jps.mas.mgmt.jmx.policy.PortableGrantee;
import oracle.security.jps.mas.mgmt.jmx.policy.PortablePermission;
import oracle.security.jps.mas.mgmt.jmx.policy.PortablePrincipal;
import oracle.security.jps.mas.mgmt.jmx.policy.PortableRoleMember;
import oracle.security.jps.mas.mgmt.jmx.util.JpsJmxConstants;

public class InvokeJpsMbeans {
    private static JMXConnector connector;
    private static MBeanServerConnection wlsMBeanConn;
    private static ObjectName configName;
    private static ObjectName credName;
    private static ObjectName appPolName;
    private static ObjectName gloPolName;
    private static ObjectName adminPolName;

    private final static String STR_NAME =String.class.getName();

    public static void main(String args[]) {
        // Intialize connection and retrieve connection object
        init();

        //Check registration
        if (isRegistered(configName))
            System.out.println("Jps Config MBean is registered");
        if (isRegistered(credName))
            System.out.println("Jps Credential Mbean is registered");
        if (isRegistered(appPolName))
            System.out.println("Jps Application policy Mbean is registered");
        if (isRegistered(gloPolName))
            System.out.println("Jps Global policy Mbean is registered");
        if (isRegistered(adminPolName))
           System.out.println("Jps Admin Policy Mbean is registered");

        //invoke MBeans
        invokeConfigMBeanMethods();
        invokeCredentialMBeanMethods();
        invokeApplicationPolicyMBeanMethods();
        invokeGlobalPolicyMBeanMethods();
```

```
            invokeAdminPolicyMBeanMethhods();
        }

        private static void invokeConfigMBeanMethods() {
            String KEY = "myKey";
            String VALUE = "myValue";
            String strVal;
            try {
                strVal = (String) wlsMBeanConn.invoke(configName, "updateProperty",
                         new Object[] { KEY, VALUE },
                         new String[] { STR_NAME, STR_NAME });
                wlsMBeanConn.invoke(configName,"persist",null,null);

                strVal = (String) wlsMBeanConn.invoke(configName, "getProperty",
                         new Object[] { KEY }, new String[] { STR_NAME });
                System.out.println("Updated the property: " + strVal.equals(strVal));

                strVal = (String) wlsMBeanConn.invoke(configName, "removeProperty",
                         new Object[] { KEY }, new String[] { STR_NAME });
                wlsMBeanConn.invoke(configName,"persist",null,null);
            } catch (InstanceNotFoundException e) {
                // auto-generated catch block
                e.printStackTrace();
            } catch (MBeanException e) {
                // auto-generated catch block
                e.printStackTrace();
            } catch (ReflectionException e) {
                // auto-generated catch block
                e.printStackTrace();
            } catch (IOException e) {
                // auto-generated catch block
                e.printStackTrace();
            }
        }

    private static void  invokeCredentialMBeanMethods() {

            String USER = "jdoe";
            String PASSWORD = "welcome1";
            String ALIAS = "mapName";
     String KEY = "keyValue";

     PortableCredential cred = new PortablePasswordCredential(USER,
    PASSWORD.toCharArray());

      try {
           //seed a password credential
           wlsMBeanConn.invoke(credName, "setPortableCredential", new Object[] {
    ALIAS, KEY, cred.toCompositeData(null) }, new String[] { STR_NAME, STR_NAME,
    CompositeData.class.getName() });
            boolean bContainsMap = (Boolean) wlsMBeanConn.invoke(credName,
    "containsMap", new Object[] { ALIAS }, new String[] { STR_NAME });
            System.out.println("Credstore contains map: " + ALIAS + " - "
    +bContainsMap);

            boolean bContainsCred = (Boolean) wlsMBeanConn.invoke(credName,
    "containsCredential", new Object[] { ALIAS, KEY }, new String[] { STR_NAME, STR_
    NAME });
            System.out.println("Contains Credential; " + bContainsCred);
        CompositeData cd = (CompositeData) wlsMBeanConn.invoke(credName,
"getPortableCredential", new Object[] { ALIAS, KEY }, new String[] { STR_NAME,
STR_NAME });
        cred = PortableCredential.from(cd);

        PortablePasswordCredential pc = (PortablePasswordCredential) cred;

        System.out.println("User name should be " +  USER + " Retrieved - " +
pc.getName());
        System.out.println("Password should be " + PASSWORD + "retrieved - " +
new String(pc.getPassword()));

        //delete entire map
        wlsMBeanConn.invoke(credName, "deleteCredentialMap", new Object[] {ALIAS},
new String[] {STR_NAME} );

        } catch (InstanceNotFoundException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (MBeanException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (ReflectionException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // auto-generated catch block
            e.printStackTrace();
        }

    }

private static void invokeApplicationPolicyMBeanMethods() {
        //add grants to approles

        //first create application policy
        String TESTGET_APP_ROLES_MEMBERS = "testgetAppRolesMembers";
        try {
            wlsMBeanConn.invoke(appPolName, "deleteApplicationPolicy", new
Object[] { TESTGET_APP_ROLES_MEMBERS }, new String[] { STR_NAME });
        } catch (Exception e ) {
            System.out.println("IGNORE: App " + TESTGET_APP_ROLES_MEMBERS + "
might not exist");
        }
        try {
            wlsMBeanConn.invoke(appPolName, "createApplicationPolicy", new
Object[] { TESTGET_APP_ROLES_MEMBERS }, new String[] { STR_NAME });
            // add remove members to applicaiton roles
            // Create App Role here
            String APP_ROLE_NAME = "ravenclaw_house";
            wlsMBeanConn.invoke(appPolName, "createApplicationRole", new Object[]
{ TESTGET_APP_ROLES_MEMBERS, APP_ROLE_NAME, null, null, null }, new String[] {
STR_NAME, STR_NAME, STR_NAME, STR_NAME, STR_NAME });

            CompositeData cd = (CompositeData) wlsMBeanConn.invoke(appPolName,
"getApplicationRole", new Object[] { TESTGET_APP_ROLES_MEMBERS, APP_ROLE_NAME },
new String[] { STR_NAME, STR_NAME });
            PortableApplicationRole appRole = PortableApplicationRole.from(cd);

            //Add custom principal here
            PortableRoleMember prm_custom = new
PortableRoleMember("My.Custom.Principal","CustomPrincipal",null,null,null);

            CompositeData[] arrCompData = { prm_custom.toCompositeData(null) };
            cd = (CompositeData) wlsMBeanConn.invoke(appPolName,
"addMembersToApplicationRole", new Object[] { TESTGET_APP_ROLES_MEMBERS,
appRole.toCompositeData(null), arrCompData }, new String[] { STR_NAME,
CompositeData.class.getName(), CompositeData[].class.getName() });

            // Chk if member got added
            CompositeData[] arrCD = (CompositeData[])
wlsMBeanConn.invoke(appPolName, "getMembersForApplicationRole", new Object[] {
TESTGET_APP_ROLES_MEMBERS, appRole.toCompositeData(null) }, new String[] { STR_
NAME, CompositeData.class.getName() });
            PortableRoleMember[] actRM = getRMArrayFromCDArray(arrCD);
            PortableRoleMember[] expRM = { prm_custom};
            chkRoleMemberArrays(actRM, expRM);

            cd = (CompositeData) wlsMBeanConn.invoke(appPolName,
"removeMembersFromApplicationRole", new Object[] { TESTGET_APP_ROLES_MEMBERS,
appRole.toCompositeData(null), arrCompData }, new String[] { STR_NAME,
CompositeData.class.getName(), CompositeData[].class.getName() });

            // Chk if member got removed
            arrCD = (CompositeData[]) wlsMBeanConn.invoke(appPolName,
"getMembersForApplicationRole", new Object[] { TESTGET_APP_ROLES_MEMBERS,
appRole.toCompositeData(null) }, new String[] { STR_NAME,
CompositeData.class.getName() });
            System.out.println("length should be zero :" + arrCD.length);

            // Remove the App Role
            wlsMBeanConn.invoke(appPolName, "removeApplicationRole", new Object[]
{ TESTGET_APP_ROLES_MEMBERS, APP_ROLE_NAME }, new String[] { STR_NAME, STR_NAME
});
            wlsMBeanConn.invoke(appPolName, "deleteApplicationPolicy", new
Object[] { TESTGET_APP_ROLES_MEMBERS }, new String[] { STR_NAME });

        } catch (InstanceNotFoundException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (MBeanException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (ReflectionException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // auto-generated catch block
            e.printStackTrace();
        }
    }

    private static PortableRoleMember[] getRMArrayFromCDArray(CompositeData[]
arrCD) {
        PortableRoleMember[] actRM = new PortableRoleMember[arrCD.length];
        int idx = 0;
        for (CompositeData cdRM : arrCD) {
            actRM[idx++] = PortableRoleMember.from(cdRM);
        }
        return actRM;
    }

    private static void chkRoleMemberArrays(PortableRoleMember[] arrExpectedRM,
PortableRoleMember[] arrActRM) {

        List < PortableRoleMember > lstExpRM = new ArrayList < PortableRoleMember
>(Arrays.asList(arrExpectedRM));
        List < PortableRoleMember > lstActRM = new ArrayList < PortableRoleMember
>(Arrays.asList(arrActRM));

        for (PortableRoleMember actRM : lstActRM) {
            for (int idx = 0; idx < lstExpRM.size(); idx++) {
                PortableRoleMember expRM = (PortableRoleMember) lstExpRM.get(idx);
                if (expRM.equals(actRM)) {
                    lstExpRM.remove(idx);
                    break;
                }
            }
        }
        System.out.println("List should be empty - " + lstExpRM.size());
    }

    private static void  invokeAdminPolicyMBeanMethhods() {
        //Connection is established as weblogic user, who by OOTB gets all
permissions
        Boolean bool;
        try {
            bool = (Boolean) wlsMBeanConn.invoke(adminPolName,"checkRole",new
Object[]{"Admin"}, new String[]{STR_NAME});
            System.out.println("Werblogic has Admin role: " + bool);
            bool = (Boolean) wlsMBeanConn.invoke(adminPolName,"checkRole",new
Object[] {"Configurator"}, new String[]{STR_NAME});
            System.out.println("Werblogic has Configurator role: " + bool);
            bool = (Boolean) wlsMBeanConn.invoke(adminPolName,"checkRole", new
Object[]{new String[] {"Operator", "Admin", "Configurator"}},
                    new String[]{String[].class.getName()});
            System.out.println("Werblogic has Admin,Operator,Configurator role: "
+ bool);
        } catch (InstanceNotFoundException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (MBeanException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (ReflectionException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // auto-generated catch block
            e.printStackTrace();
        }
    }

    private static void invokeGlobalPolicyMBeanMethods() {
        // lets create a grant in system policy
        PortablePrincipal CUSTOM_JDOE = new
PortablePrincipal("oracle.security.jps.internal.core.principals.CustomXmlUserImpl"
, "jdoe", PortablePrincipal.PrincipalType.CUSTOM);
        PortablePrincipal CUSTOM_APP_ADMINS = new
PortablePrincipal("oracle.security.jps.internal.core.principals.CustomXmlEnterpris
eRoleImpl", "oc4j-app-administrators", PortablePrincipal.PrincipalType.CUSTOM);
        PortablePrincipal[] arrPrincs = {CUSTOM_JDOE, CUSTOM_APP_ADMINS};
        //code source URL
        String URL = "http://www.oracle.com/as/jps-api.jar";
        PortableCodeSource pcs = new PortableCodeSource(URL);
        PortableGrantee pge = new PortableGrantee(arrPrincs, pcs);
        PortablePermission CSF_PERM = new
PortablePermission("oracle.security.jps.service.credstore.CredentialAccessPermissi
on", "context=SYSTEM,mapName=MY_MAP,keyName=MY_KEY", "read");
        PortablePermission[] arrPerms = {CSF_PERM};
        PortableGrant grnt = new PortableGrant(pge, arrPerms);
        CompositeData[] arrCompData = { grnt.toCompositeData(null) };
        try {
            System.out.println("Creating System Policy grant");
            wlsMBeanConn.invoke(gloPolName, "grantToSystemPolicy", new Object[] {
arrCompData }, new String[] { CompositeData[].class.getName() });
            System.out.println("Deleting the created grant");
            wlsMBeanConn.invoke(gloPolName, "revokeFromSystemPolicy", new Object[]
{ arrCompData }, new String[] { CompositeData[].class.getName() });

        } catch (InstanceNotFoundException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (MBeanException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (ReflectionException e) {
            // auto-generated catch block
            e.printStackTrace();
        } catch (IOException e) {
            // auto-generated catch block
            e.printStackTrace();
        }
    }

    private static boolean isRegistered(ObjectName name) {
        try {
            return wlsMBeanConn.isRegistered(name);
        } catch (IOException e) {
            // auto-generated catch block
            e.printStackTrace();
        }
        return false;
    }

    private static void init() {
        String protocol = "t3";
        String jndi_root = "/jndi/";
        String wlserver = "myWLServer";
        String host =  "myHost.com";
        int port =  7001;
        String adminUsername = "myAdminName";
        String adminPassword = "myAdminPassw";
        JMXServiceURL url;
        try {
            url = new JMXServiceURL(protocol,host,port,jndi_root+wlserver);
            HashMap<String, Object> env = new HashMap<String, Object>();
            env.put(Context.SECURITY_PRINCIPAL, adminUsername);
            env.put(Context.SECURITY_CREDENTIALS, adminPassword);
            env.put(JMXConnectorFactory.PROTOCOL_PROVIDER_PACKAGES,
                    "weblogic.management.remote");
            connector = JMXConnectorFactory.connect(url, env);
            wlsMBeanConn = connector.getMBeanServerConnection();

            //create object names
// the next string is set to com.oracle.jps:type=JpsConfig
            configName = new
                 ObjectName(JpsJmxConstants.MBEAN_JPS_CONFIG_FUNCTIONAL);
// the next string is set to com.oracle.jps:type=JpsApplicationPolicyStore
            appPolName = new
                 ObjectName(JpsJmxConstants.MBEAN_JPS_APPLICATION_POLICY_STORE);
// the next string is set to com.oracle.jps:type=JpsGlobalPolicyStore
            gloPolName = new
                 ObjectName(JpsJmxConstants.MBEAN_JPS_GLOBAL_POLICY_STORE);
// the next string is set to com.oracle.jps:type=JpsAdminPolicyStore
            adminPolName = new
                 ObjectName(JpsJmxConstants.MBEAN_JPS_ADMIN_POLICY_STORE);
// the next string is set to com.oracle.jps:type=JpsCredentialStore
            credName = new ObjectName(JpsJmxConstants.MBEAN_JPS_CREDENTIAL_STORE);
        } catch (MalformedURLException e) {
            // take proper action
            e.printStackTrace();
        } catch (IOException e) {
            // take proper action
            e.printStackTrace();
        } catch (MalformedObjectNameException e) {
            // auto-generated catch block
            e.printStackTrace();
        }
    }
}
```

For further details about programmatic configuration of services, see part Part IV, "Developing with Oracle Platform Security Services APIs"

E.3 Restricting Access

The information in this section is not restricted to OPPS MBeans but applies, more generally, to Oracle Fusion Middleware MBeans.

The security access to MBeans is based on logical roles rather than on security permissions. MBeans are annotated using role-based constraints that are enforced at run time by the JMX Framework.

This section illustrates the use of some annotations, describes what they mean, lists the particular access restrictions, and explains the mapping of logical roles to Oracle WebLogic Server enterprise groups.

E.3.1 Annotation Examples

The following code snippet illustrates the use of some enterprise group annotations (in bold text) in an MBean interface:

```
@Description(resourceKey = "demo.ScreenCustomizerRuntimeMBean.description",
             resourceBundleBaseName = "demo.runtime.Messages")
@ImmutableInfo("true")
@Since("1.1")
public interface ScreenCustomizerRuntimeMXBean {
  @Description(resourceKey = "demo.ScreenCustomizerRuntimeMBean.Active",
               resourceBundleBaseName = "demo.runtime.Messages")
  @AttrributeGetterRequiredGlobalSecurityRole(GlobalSecurityRole.Operator)
    public boolean isActive();
  @AttrributeSetterRequiredGlobalSecurityRole(GlobalSecurityRole.Admin)
    public void setActive(boolean val);

  @Description(resourceKey =
                     "demo.ScreenCustomizerRuntimeMBean.ActiveVirtualScreenId",
               resourceBundleBaseName = "demo.runtime.Messages")
  @DefaultValue("0")
  @LegalValues( {"0", "2", "4", "6", "8" })
  @RequireRestart(ConfigUptakePolicy.ApplicationRestart)
  @OperationRequiredGlobalSecurityRole(GlobalSecurityRole.Admin)
     public void setActiveVirtualScreenId(int id) throws IllegalArgumentException;
  …
}
```

In the above code sample, the annotation:

- `@AtrributeGetterRequiredGlobalSecurityRole` specifies that a user must belong to the role Operator to access the get method `isActive`.
- `@AtrributeSetterRequiredGlobalSecurityRole` specifies that a user must belong to the role Admin to access the set method `setActive`.
- `@OperationRequiredGlobalSecurityRole` specifies that a user must belong to the role Admin to access the MBean method `setActiveVirtualScreenId`.

Note that all three annotations above apply just to a specific item in the interface.

The following code snippet illustrates the use of another annotation (in bold text) with a different scope:

```
@Description(resourceKey = "demo.ScreenCustomizerRuntimeMBean.description",
             resourceBundleBaseName = "demo.runtime.Messages")
@ImmutableInfo("true")
@Since("1.1")
@MBeanRequiredGlobalSecurityRole(GlobalSecurityRole.Admin)
public interface ScreenCustomizerRuntimeMXBean { … }
```

In the above code sample, the annotation `@MbeanRequiredGlobalSecurityRole` specifies that a user must belong to the role Admin to access *any* operation or attribute of the MBean, that is, its scope is the entire MBean. Annotations with method or attribute scope override annotations that apply to the entire MBean.

The enumeration `GlobalSecurityRole` defines the set of global, logical roles that are mapped to actual roles in the environment before performing security checks. This enumeration includes the value `NONE` to indicate that any user has read and write access to the annotated operation or attribute.

For details, see the oracle.jmx.framework Javadoc documentation.

E.3.2 Mapping of Logical Roles to WebLogic Roles

Table E–2 shows the mapping of logical roles to enterprise groups.

Table E–2 Mapping of Logical Roles to WebLogic Groups

Logical Role	Default Privileges	WebLogic Group
Admin	Read and write access to all MBeans	Admin
Configurator	Read and write access to configuration MBeans	Admin

Table E–2 (Cont.) Mapping of Logical Roles to WebLogic Groups

Logical Role	Default Privileges	WebLogic Group
Operator	Read access to configuration MBeans; read and write access to all run time MBeans	Operator
Monitor	Read access to all MBeans	Monitor
ApplicationAdmin	Read and write access to all application MBeans	Admin
ApplicationConfigurator	Read and write access to all application MBeans	Admin
ApplicationOperator	Read access to application configuration MBeans; read and write access to application runtime MBeans	Operator
ApplicationMonitor	Read access to all application runtime and configuration MBeans	Monitor

For details about WebLogic roles, see sections Users, Groups, and Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*.

E.3.3 Particular Access Restrictions

By default, all write and update operations require that the user be a member of the Admin or Configurator roles. In addition, operations annotated with the tag `@Impact(value=1)` require the user to be a member of the Admin role, and operations annotated with the tag `@Impact(value=0)` require the user to be a member of the Admin or Operator roles.

Table E–3 describes the roles required to access attributes and operations of Fusion Middleware Control MBeans:

Table E–3 Roles Required per Operation

Operations with impact value	MBean type	Require any of the roles
INFO or attribute getter	System configuration MBean	Monitor, Operator, Configurator, Admin
INFO or attribute getter	Application configuration MBean	Monitor, Operator, Configurator, Admin, ApplicationMonitor, ApplicationOperator, ApplicationConfigurator, ApplicationAdmin
ACTION, ACTION_INFO, UNKNOWN, or attribute setter	System configuration MBean	Admin, Configurator
ACTION, ACTION_INFO, UNKNOWN, or attribute setter	Application configuration MBean	Admin, Configurator, ApplicationAdmin, ApplicationConfigurator
INFO or attribute getter	System runtime MBean	Monitor, Operator, Configurator, Admin
INFO or attribute getter	Application runtime MBean	Monitor, Operator, Configurator, Admin, ApplicationMonitor, ApplicationOperator, ApplicationAdmin

Table E–3 (Cont.) Roles Required per Operation

Operations with impact value	MBean type	Require any of the roles
ACTION, ACTION_ INFO, UNKNOWN, or attribute setter	System runtime MBean	Admin, Operator
ACTION, ACTION_ INFO, UNKNOWN, or attribute setter	Application runtime MBean	Admin, Operator, ApplicationAdmin, ApplicationOperator

F OPSS System and Configuration Properties

This appendix documents OPSS system properties (set through the switch `-D` at server start) and configuration properties (set with elements `<property>` and `<extendedProperty>` in the configuration file `jps-config.xml`).

It includes the following sections:

- OPSS System Properties
- OPSS Configuration Properties

To manage server properties programmatically, use OPSS MBeans. For details and example, see Section E.2.3, "Programming with OPSS MBeans."

Note: All OPSS *configuration* changes (manual or through JpsConfiguration MBean) require server restart to take effect.

OPSS *data* domain changes do not require server restart to take effect. Data changes include modifying an application policy and creating, deleting, or updating a credential.

F.1 OPSS System Properties

A system property that has been introduced or modified is not in effect until the server is restarted. In order to set a system property, the administrator must edit the `setDomainEnv.sh` shell script and add the property to the environment variable `EXTRA_JAVA_PROPERTIES` in that script.

Table F–1 lists the system properties available with OPSS.

Table F–1 OPSS System Properties

Name	Description
`common.components.home`	Specifies the location of the common components home. Required for both Java EE and Ja va SE applications. No default value.
`java.security.debug`	Notifies about a permission failure when the method JpsAuth.checkPermission is called inside a Subject.doAs block and the permission check fails. Note that setting jps.auth.debug or jps.auth.debug.verbose is not enough to get a failure notification in this case. Optional.
`java.security.policy`	Specifies the location of the Java security policy file.

Table F–1 (Cont.) OPSS System Properties

Name	Description
`jps.app.permissioncollectionmap.size`	Controls the number of permissioncollectionmap entries kept in memory. Each entry corresponds with a set of permissions. For this setting to take effect, the property `jps.policystore.ref.useSoftHardMapForSelectedMaps` must be set to `true`. Optional. Valid values: a non-negative integer. Default value: `512`.
`jps.authz`	Enables or disables the delegation of calls to JDK API AccessController.checkPermission, which reduces runtime and debugging overhead. Optional. Valid values: `NULL`, `SM`, `ACC`, and `DEBUG_NULL`. No default value.
`jps.auth.debug`	Controls server logging output. Default value: FALSE. For details, see Section J.1.2.1, "jps.auth.debug." Optional.
`jps.auth.debug.verbose`	Controls server logging output. Default value: FALSE. For details, see Section J.1.2.2, "jps.auth.debug.verbose." Optional.
`jps.combiner.optimize`	Enables or disables the caching of a subject's protection domain. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.
`jps.combiner.optimize.lazyeval`	Enables or disables the evaluation of a subject's protection domain when a check permission is triggered. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.
`jps.combinermap.size`	Controls the number of combinermap entries kept in memory. Each entry corresponds with a set of principals. For this setting to take effect, the property `jps.policystore.ref.useSoftHardMapForSelectedMaps` must be set to `true`. Optional. Valid values: a non-negative integer. Default value: `128`.
`jps.deployment.handler.disabled`	Enables or disables the migration of policies and credentials for applications deployed on a WebLogic Server. Valid only for the WebLogic Server. Set to TRUE to disable the migration of application policies and credentials for all applications deployed on the server regardless of the particular application settings in the application file weblogic-application.xml. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.

Table F–1 (Cont.) OPSS System Properties

Name	Description
`jps.policystore.hybrid.mode`	Enables or disables the hybrid mode. When hybrid mode is enabled, the OPSS policy provider reads from java.policy, weblogic.policy, and the policy store configured in jps-config.xml. Optional. Valid values: `TRUE`, `FALSE`. Default value: `TRUE`.
`jps.policystore.ref.useSoftHardMapForSelectedMaps`	Controls the use of the map type. The map type is used to hold some structures in a special cache so that they are not garbage-collected by the Java Virtual Machine. If set to `false`, then the `SoftKeyHashMap` type is used. If set to `true`, then the `SoftHardKeyHashMap` type is used; this setting allows retaining some of the maps in memory; note that every get/put operation will then have a lock operation overhead. See related properties `jps.subjectmap.size`, `jps.combinermap.size`, and `jps.app.permissioncollectionmap.size`. Optional. Valid values: `true`, `false`. Default value: `false`.
`jps.subject.cache.ttl`	Specifies the number of milliseconds after which group membership changes are in effect. This value must be kept in sych with the value of the WebLogic authenticator `Group Hierarchy Cache`. If this last parameter value is changed, then `jps.subject.cache.ttl` must be reset to match the new `Group Hierarchy Cache` value. Optional. Valid values: any positive integer. Default value: `60000`
`jps.subjectmap.size`	Controls the number of subjectmap entries kept in memory. Each entry corresponds with TTL information about a subject. For this setting to take effect, the property `jps.policystore.ref.useSoftHardMapForSelectedMaps` must be set to `true`. Optional. Valid values: a non-negative integer. Default value: `128`.
`oracle.security.jps.config`	Specifies the path to the domain configuration files `jps-config.xml` or `jps-config-jse.xml`. Paths specifications in those files can be absolute or relative to the location of the configuration file. Required. No default value.
`oracle.deployed.app.dir`	Specifies the path to the directory of a code source URL. Optional. No default value. For an example of use, see <url>.

Table F–1 (Cont.) OPSS System Properties

Name	**Description**
`oracle.deployed.app.ext`	Specifies the extension of code source URL. Optional. No default value. For an example of use, see <url>.
`oracle.security.jps.log.for.approle.substring`	Logs the name of an application role that contains a specified substring; if the substring to match is unspecified, it logs all application role names. Optional. No default value. For an example of use and further details, see Section J.1.2.3, "Debugging the Authorization Process."
`oracle.security.jps.log.for.permeffect`	Logs a grant that was granted or denied according to a specified value; if the value is unspecified, it logs all grants (regardless whether they were granted or denied). Optional. No default value. For an example of use and further details, see Section J.1.2.3, "Debugging the Authorization Process."
`oracle.security.jps.log.for.permclassname`	Logs the name of the permission class that matches exactly a specified name; if the name to match is unspecified, it logs all permission class names. Optional. No default value. For an example of use and further details, see Section J.1.2.3, "Debugging the Authorization Process."
`oracle.security.jps.log.for.permtarget.substring`	Logs the name of a permission target that contains a specified substring; if the substring to match is unspecified, it logs all permission targets. Optional. No default value. For an example of use and further details, see Section J.1.2.3, "Debugging the Authorization Process."
`oracle.security.jps.log.for.enterprise.principalname`	Logs the name of the principal (enterprise user or enterprise role) that matches exactly a specified name; if the name to match is unspecified, it logs all principal names. Optional. No default value. For an example of use and further details, see Section J.1.2.3, "Debugging the Authorization Process."

Table F–1 (Cont.) OPSS System Properties

Name	Description
`opss.audit.logDirectory`	Specifies the location of the Opss audit log files for SE applications if it is not set in jps-config-jse.xml.. Optional. No default value. Valid values: any writeable directory.
`wlst.offline.log`	Specifies the location of the log file when running offline WLST commands. Optional. No default value. Valid values: <filename>, stdout, strerr, disable.
`wlst.offline.log.priority`	Specifies the level of the notification. Optional. No default value. Valid values: OFF, SEVERE, WARNING, INFO , CONFIG, FINE, FINER, FINEST, ALL, debug, info, warn, error, fatal.

F.2 OPSS Configuration Properties

This section describes the properties of various instances in the following sections:

- Common Properties
- Policy Store Properties
- Credential Store Properties
- LDAP Identity Store Properties
- Properties Common to All OID-Based Instances
- Anonymous and Authenticated Roles Properties
- Trust Service Properties
- Audit Service Properties
- Keystore Service Properties

F.2.1 Common Properties

Table F–2 describes the OPSS properties common to all services (except trust service).

Table F–2 Common Properties

Name	Description
The following properties are valid in both Java EE and Java SE applications	
`bootstrap.security.principal.key`	The key for the password credentials to access the OID store, stored in the bootstrap wallet. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Required. No default value. The out-of-the-box value is `bootstrap`.

Table F–2 (Cont.) Common Properties

Name	Description
`bootstrap.security.principal.map`	The map for the password credentials to access the OID store, stored in the bootstrap wallet. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Required. Default value: `BOOTSTRAP_JPS`.
`jdbc.url`	The URL of the JBDC. Valid in Java SE and Java EE applications. Applies to only DB stores. Required. No default value. Value example: `jdbc:oracle:thin:@xxx27.com:1345:asi102cn`
`ldap.url`	The URL of the OID security store, with the format `ldap://host:port`. Valid in Java EE and Java SE applications. Applies only to OID stores. Required. No default value.
`oracle.security.jps.farm.name`	The RDN format of the domain node in the OID store. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Required. No default value.
`oracle.security.jps.ldap.root.name`	The RDN format of the root node in the OID store. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Required. No default value.
`oracle.security.jps.pdp.PolicyProvider.PermissionCollectionCache.MaxSize`	The maximum number of permission collections allowed in the cache per protection domain and request permission class. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: 5000
`server.type`	The type of the policy store. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Required. No default value. Values: `OID`, `DB_ORACLE`.
The following properties are valid in Java EE applications only	

Table F–2 (Cont.) Common Properties

Name	Description
`datasource.jndi.name`	The JNDI name of the JDBC data source instance. Valid in Java EE applications only. Applies to only DB stores. Required. No default value.
`oracle.security.jps.failover.retry.times`	The number of retry attempts. Valid in Java EE applications only. Applies to only DB stores. Optional. Default value: 3
`oracle.security.jps.failover.retry.interval`	The number of seconds between retry attempts. Valid in Java EE applications only. Applies to only DB stores. Optional. Default value: 15
`weblogic.dbuser.map` `weblogic.dbuser.key`	Specify the credential's map and key for the Weblogic DB user/password. They apply only when `oracle.security.jps.db.useWeblogicDBUserMapKey` is true; in this case, both or none of them should be configured. Valid in Java EE applications only. Applies to only DB stores. Optional. Default value: none.
`oracle.security.jps.db.useWeblogicDBUserMapKey`	Specifies where to find the map and key for the WebLogic DB user/password. This property is automatically set when reassociating to a DB-based store. Valid in Java EE applications only. Applies to only DB stores. Optional. Valid values: `true` or `false`. Default value: `false`. If `true`, then the properties `weblogic.dbuser.map` and `weblogic.dbuser.key` specify the credential's map and key for the Weblogic DB user/password. Otherwise, if `false` or not specified, then the properties `bootstrap.security.principal.map` and `bootstrap.security.principal.key` specify the credential's map and key for the weblogic DB user/password.
The following properties are valid in Java SE applications only	

Table F–2 (Cont.) Common Properties

Name	Description
`security.principal`	The clear text name of the principal to use instead of the user name specified in the bootstrap. Used in developments environments only. Valid in Java SE applications only. Applies to OID and DB stores. Optional. No default value.
`security.credential`	The clear text password for the security principal to use instead of the password specified in the bootstrap. Not recommended. Valid in Java SE applications only. Applies to OID and DB stores. Optional. No default value.
`jdbc.driver`	The JDBC driver. Valid in Java SE applications only. Applies to only DB stores. Required. No default value. Value example: `oracle.jdbc.driver.OracleDriver`

F.2.2 Policy Store Properties

The policy store properties are described in the following sections:

- Policy Store Configuration
- Runtime Policy Store Configuration

F.2.2.1 Policy Store Configuration

The policy store provider class that can be used with OID-based or DB-based instances is the following:

```
oracle.seurity.jps.internal.policystore.ldap.LdapPolicyStoreProvider
```

Table F–3 describes the properties specific to policy store instances. The properties are listed in three blocks according to the kind of application they can be used in. Additional properties are listed in Common Properties.

Table F–3 Policy Store Properties

Name	Description
`oracle.security.jps.policystore.resourcetypeenforcementmode`	Controls the throwing of exceptions if any of the following checks fail: ▪ Verify that if two resource types share the same permission class, that permission must be either `ResourcePermission` or extend `AbstractTypedPermission`, and this last resource type cannot be created. ▪ Verify that all permissions have resource types defined, and that the resource matcher permission class and the permission being granted match. If set to `Strict`, when any of the above checks fail, the system throws an exception and the operation is aborted. If set to `Lenient`, when any of the above checks fail, the system does not throw any exceptions, the operation continues without disruption, and any discrepancies encountered are logged in the log files. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: `Lenient` Valid values: `Strict`, `Lenient`.

Example 1

The following fragment illustrates the configuration of an OID-based policy store instance for a Java EE application:

```
<propertySet name="props.ldap.1">
 <property name="java.naming.ldap.derefAliases" value="never"/>
 <property name="bootstrap.security.principal.key" value="bootstrap_
6aCNhgRM3zF04ToliwecdF6K3oo="/>
 <property name="oracle.security.jps.farm.name" value="cn=compact1_oid26008"/>
 <property name="server.type" value="OID"/>
 <property name="oracle.security.jps.ldap.root.name" value="cn=jpsTestNode"/>
 <property name="ldap.url" value="ldap://myComp.com:2020"/>
</propertySet>

<serviceProvider type="POLICY_STORE" name="policystore.provider"
class="oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"/>

<serviceInstance name="policystore.ldap" provider="policystore.provider">
 <propertySetRef ref="props.ldap.1"/>
</serviceInstance>
```

Example 2

The following fragment illustrates the configuration of an OID-based policy store instance for a Java SE application:

```
<serviceInstance name="policystore.oid" provider="policy.oid">
   <property value="OID" name="server.type"/>
   <property value="bootstrap" name="bootstrap.security.principal.key"/>
   <property name="ldap.url" value="ldap://myHost.com:1234"/>
   <property name="oracle.security.jps.ldap.root.name" value="cn=jpsNode"/>
   <property name="oracle.security.jps.farm.name" value="cn=domain1"/>
</serviceInstance>
```

For additional configurations samples for Java SE applications, see Section 18.1.2, "Configuring LDAP-Based Policy and Credential Stores."

Example 3

The following fragment illustrates the configuration of DB-based stores (including an instance of a runtime service provider) for a Java EE application:

```
<jpsConfig>
...
  <propertySets>
    <!-- property set props.db.1 common to all DB services -->
    <propertySet name="props.db.1">
      <property name="jdbc.url" value="jdbc:oracle:thin@xxx.com:1521:orcl"/>
      <property name="datasource.jndi.name"  value="opssds"/>
      <property value="cn=farm" name="oracle.security.jps.farm.name"/>
      <property value="cn=jpsroot" name="oracle.security.jps.ldap.root.name"/>
      <property value="dsrc_lookup_key"
                name="bootstrap.security.principal.key"/>
      <property value="credential_map" name="bootstrap.security.principal.map"/>
    </propertySet>
  </propertySets>

  <serviceProviders>
    <serviceProvider
     class="oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider"
     type="POLICY_STORE" name="rdbms.policystore.provider" >
       <description>RDBMS based PolicyStore provider</description>
    </serviceProvider>

    <serviceProvider type="KEY_STORE" name="keystore.provider"
       class="oracle.security.jps.internal.keystore.KeyStoreProvider">
      <description>PKI Based Keystore Provider</description>
      <property name="provider.property.name" value="owsm"/>
    </serviceProvider>

    <serviceProvider name="pdp.service.provider" type="PDP"
      class="oracle.security.jps.az.internal.runtime.provider.PDPServiceProvider">
      <description>OPSS Runtime Service provider</description>
    </serviceProvider>
  </serviceProviders>

  <serviceInstances>
    <serviceInstance name="policystore.rdbms"
                     provider="rdbms.policystore.provider">
      <property value="DB_ORACLE" name="server.type"/>
      <propertySetRef ref = "props.db.1"/>
      <property name="session_expiration_sec" value="60"/>
      <property name="failover.retry.times"  value="5"/>
    </serviceInstance>

    <serviceInstance name="credstore.rdbms" provider="rdbms.credstore.provider">
      <propertySetRef ref = "props.db.1"/>
    </serviceInstance>

    <serviceInstance name="keystore.rdbms" provider="rdbms.keystore.provider">
      <propertySetRef ref = "props.db.1"/>
      <property name="server.type"  value="DB_ORACLE"/>
    </serviceInstance>

    <serviceInstance name="pdp.service" provider="pdp.service.provider">
```

```
      <property name="oracle.security.jps.runtime.pd.client.sm_name"
value="permissionSm"/>
      <property name="oracle.security.jps.pdp.AuthorizationDecisionCacheEnabled"
value="true"/>
      <property
name="oracle.security.jps.pdp.AuthorizationDecisionCacheEvictionCapacity"
value="500"/>
      <property
name="oracle.security.jps.pdp.AuthorizationDecisionCacheEvictionPercentage"
value="10"/>
      <property name="failover.retry.times"  value="5"/>
      <property name="failover.retry.interval" value="20"/>
      <property name="oracle.security.jps.policystore.refresh.purge.timeout",
                value="30000"/>
      <propertySetRef ref = "props.db.1"/>
    </serviceInstance>
  </serviceInstances>

  <jpsContexts default="default">
    <jpsContext name="default">
      <serviceInstanceRef ref="pdp.service"/>
      <serviceInstanceRef ref="policystore.rdbms"/>
      <serviceInstanceRef ref="credstore.rdbms"/>
      <serviceInstanceRef ref="keystore.rdbms"/>
    </jpsContext>
  </jpsContexts>
...
</jpsConfig>
```

Example 4

The following fragment illustrates the configuration of a DB-based policy store instance for a Java SE application:

```
<serviceInstance name="policystore.rdbms" provider="policy.rdbms">
  <property name="server.type" value="DB_ORACLE"/>
  <property name="jdbc.url" value="jdbc:oracle:thin:@xxx.com:1722:orcl"/>
  <property name="jdbc.driver" value="oracle.jdbc.driver.OracleDriver"/>
  <property name="bootstrap.security.principal.key" value="bootstrap_
DWgpEJgXwhDIoLYVZ2OWd4R8wOA=" />
  <property name="oracle.security.jps.ldap.root.name" value="cn=jpsTestNode"/>
  <property name="oracle.security.jps.farm.name" value="cn=view_steph.atz"/>
</serviceInstance>
```

For additional configurations samples for Java SE applications, see Section 18.1.3, "Configuring DB-Based OPSS Security Stores."

F.2.2.2 Runtime Policy Store Configuration

The runtime policy store provider class that can be used with OID- or DB-based instances is the following:

```
oracle.seurity.jps.az.internal.runtime.provider.PDPServiceProvider
```

Table F–4 lists the runtime properties of policy store instances.

Table F–4 Runtime Policy Store Properties

Name	Description
`oracle.security.jps.policystore.rolemember.cache.type`	The type of the role member cache. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: ■ STATIC - Cache objects are statically cached and can be cleaned explicitly only according the applied cache strategy, such as FIFO. The garbage collector does not clean a cache of this type. ■ SOFT - The cleaning of a cache of this type relies on the garbage collector when there is a memory crunch. ■ WEAK - The behavior of a cache of this type is similar to a cache of type SOFT, but the garbage collector cleans it more frequently. Default value: `STATIC`.
`oracle.security.jps.policystore.rolemember.cache.strategy`	The type of strategy used in the role member cache. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: ■ FIFO - The cache implements the first-in-first-out strategy. ■ NONE - All entries in the cache grow until a refresh or reboot occurs; there is no control over the size of the cache; not recommended but typically efficient when the policy footprint is very small. Default value: `FIFO`.
`oracle.security.jps.policystore.rolemember.cache.size`	The number of the roles kept in the member cache. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: 1000.
`oracle.security.jps.policystore.policy.lazy.load.enable`	Enables or disables the policy lazy load. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: `TRUE`, `FALSE`. Default value: `TRUE`.

Table F–4 (Cont.) Runtime Policy Store Properties

Name	Description
`oracle.security.jps.policystore.policy.cache.strategy`	The type of strategy used in the permission cache. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: ▪ PERMISSION_FIFO - The cache implements the first-in-first-out strategy. ▪ NONE - All entries in the cache grow until a refresh or reboot occurs; there is no control over the size of the cache; not recommended but typically efficient when the policy footprint is very small. Default value: `PERMISSION_FIFO`.
`oracle.security.jps.policystore.policy.cache.size`	The number of grants kept in the permission cache. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: 1000.
`oracle.security.jps.policystore.refresh.enable`	Enables or disables the policy store refresh. If this property is set, then `oracle.security.jps.ldap.cache.enable` cannot be set. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: `TRUE`, `FALSE`. Default value: `TRUE`.
`oracle.security.jps.ldap.cache.enable`	Enables or disables the refresh of the cache. If this property is set, then `oracle.security.jps.policystore.refresh.enable` cannot be set. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: `TRUE`, `FALSE`. Default value: `TRUE`.
`oracle.security.jps.policystore.refresh.purge.timeout`	The time, in milliseconds, after which the policy store cache is purged. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: 43200000 (12 hours).

Table F–4 (Cont.) Runtime Policy Store Properties

Name	Description
`oracle.security.jps.ldap.policystore.refresh.interval`	The interval, in milliseconds, at which the policy store is polled for changes. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: 600000 (10 minutes).
`oracle.security.jps.policystore.rolemember.cache.warmup.enable`	Controls the way the ApplicationRole membership cache is created. If set to TRUE, the cache is created at server startup; otherwise, it is created on demand (lazy loading). Set to TRUE when the number of users and groups is significantly higher than the number of application roles; set to FALSE otherwise, that is, when the number of application roles is very high. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.
`security.jps.runtime.pd.client.localpolicy.work_folder`	The folder for temporary storage. Valid in Java EE and Java SE applications. Applies to XML, OID, and DB stores. Optional. Default value: the system temporary folder.
`oracle.security.jps.pdp.AuthorizationDecisionCacheEnabled`	Specifies whether the authorization cache should be enabled. Valid in Java EE and Java SE applications. Applies to XML, OID, and DB stores. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.
`oracle.security.jps.pdp.AuthorizationDecisionCacheEvictionPercentage`	The percentage of sessions to drop when the eviction capacity is reached. Valid in Java EE and Java SE applications. Applies to XML, OID, and DB stores. Optional. Default value: 10
`oracle.security.jps.pdp.AuthorizationDecisionCacheEvictionCapacity`	The maximum number of authorization and role mapping sessions to maintain. When the maximum is reached, old sessions are dropped and reestablished when needed. Valid in Java EE and Java SE applications. Applies to XML, OID, and DB stores. Optional. Default value: 500

Table F–4 (Cont.) Runtime Policy Store Properties

Name	Description
`oracle.security.jps.pdp.AuthorizationDecisionCacheTTL`	The number of seconds during which session data is cached. Valid in Java EE and Java SE applications. Applies to XML, OID, and DB stores. Optional. Default value: 60
`oracle.security.jps.policystore.resourcetypeenforcementmode`	Controls the throwing of exceptions if any of the following checks fail: ▪ Verify that if two resource types share the same permission class, that permission must be either `ResourcePermission` or extend `AbstractTypedPermission`, and this last resource type cannot be created. ▪ Verify that all permissions have resource types defined, and that the resource matcher permission class and the permission being granted match. If set to `Strict`, when any of the above checks fail, the system throws an exception and the operation is aborted. If set to `Lenient`, when any of the above checks fail, the system does not throw any exceptions, the operation continues without disruption, and any discrepancies encountered are logged in the log files. Valid in Java EE and Java SE applications. Applies to OID and DB stores. Optional. Default value: `Lenient` Valid values: `Strict`, `Lenient`.

F.2.3 Credential Store Properties

Table F–5 lists the properties specific to credential store instances. Additional properties are listed in Common Properties.

Table F–5 Credential Store Properties

Name	Description
`encrypt`	Specifies whether to encrypt credentials. Valid in Java EE and Java SE applications. Applies only to file and OID stores. Valid values: `true`, `false`. Optional. Default value: `false`.

The following fragment illustrates the configuration of a credential store in a Java EE application:

```
<propertySet name="props.ldap.1">
  <property name="java.naming.ldap.derefAliases" value="never"/>
  <property name="bootstrap.security.principal.key" value="bootstrap_
6aCNhgRM3zF04ToliwecdF6K3oo="/>
```

```
  <property name="oracle.security.jps.farm.name" value="cn=compact1_oid26008"/>
  <property name="server.type" value="OID"/>
  <property name="oracle.security.jps.ldap.root.name" value="cn=jpsTestNode"/>
  <property name="ldap.url" value="ldap://myComp.com:2020"/>
</propertySet>

<serviceProvider type="CREDENTIAL_STORE" name="ldap.credentialstore.provider"
class="oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider"/>
<serviceInstance name="credstore.ldap" provider="ldap.credentialstore.provider">
  <propertySetRef ref="props.ldap.1"/>
</serviceInstance>
```

F.2.4 LDAP Identity Store Properties

Table F–6 lists the properties of LDAP-based identity store instances. Extended properties are explicitly stated. User and Role API properties corresponding to a property are also stated.

See Also: Chapter 8, "Configuring the Identity Store Service".

Table F–6 LDAP-Based Identity Store Properties

Name	Description

`idstore.type`	The type of the identity store.
	Valid in Java SE and Java EE applications.
	Required
	Valid values:
	`OID` - Oracle Internet Directory
	`OVD` - Oracle Virtual Directory
	`ACTIVE_DIRECTORY` - Microsoft Active Directory
	`IPLANET` - Oracle Directory Server Enterprise Edition
	`EDIRECTORY` - Novell eDirectory
	`OPEN_LDAP` - OpenLdap
	`LIBOVD` - Oracle Library OVD
	`CUSTOM` - Any other type
	If using a custom authenticator, the service instance configuration must include one of the following properties:
	`<property name="idstore.type" value="<your-idstore-type>"`
	`<property name="ADF_IM_FACTORY_CLASS"`
	`value="<your-IDM-FACTOY_CLASS_NAME>"`
	Corresponding User and Role API property: ADF_IM_FACTORY_CLASS

`ldap.url`	The LDAP URL value. Valid in Java SE and Java EE applications. Required. No default value. Value example: `ldap://myServerName.com:1389`. Corresponding User and Role API property: ADF_IM_PROVIDER_URL

Table F–6 (Cont.) LDAP-Based Identity Store Properties

Name	Description
`user.search.bases`	The user search base for the LDAP server in DN format. Extended property. Valid in Java SE and Java EE applications. Required. No default value. Value example: `cn=users,dc=us,dc=abc,dc=com` Corresponding User and Role API property: USER_SEARCH_BASES
`group.search.bases`	The group or enterprise search base for the LDAP server in DN format. Extended property. Valid in Java SE and Java EE applications. Required No default value. Value example: `cn=groups,dc=us,dc=abc,dc=com` Corresponding User and Role API property: ROLE_SEARCH_BASES
`idstore.config.provider`	The idstore provider class. Valid only in Java EE applications. Required The only supported value is: `oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfig Provider`

`group.create.bases`	The base DNs used to create groups or enterprise roles. Extended property.
	Valid in Java EE and Java SE applications.
	Required to allow writing operations with the User and Role API. Otherwise, optional.
	Value example of a single DN:
	`<extendedProperty>`
	` <name>group.create.bases</name>`
	` <values>`
	`  <value>cn=groups,dc=us,dc=oracle,dc=com</value>`
	` </values>`
	`</extendedProperty>`
	Corresponding User and Role API property: ROLE_CREATE_BASES

`user.create.bases`	The base DNs used to create users. Extended property.
	Valid in Java EE and Java SE applications.
	Required to allow writing operations with the User and Role API. Otherwise, optional.
	Value example of a single DN:
	`<extendedProperty>`
	` <name>user.create.bases</name>`
	` <values>`
	`  <value>cn=users,dc=us,dc=oracle,dc=com</value>`
	` </values>`
	`</extendedProperty>`
	Corresponding User and Role API property: USER_CREATE_BASES

Table F–6 (Cont.) LDAP-Based Identity Store Properties

Name	Description
`group.filter.object.classes`	The fully qualified names of object classes used to search enterprise roles and groups. Extended property. Valid in Java EE and Java SE applications. Optional. Value example: `groupOfUniqueNames`. Corresponding User and Role API property: ROLE_FILTER_OBJECT_CLASSES

`group.mandatory.attrs`	The attributes that must be specified when creating enterprise roles or groups. Extended property.
	Valid in Java EE and Java SE applications.
	Optional.
	Value example:
	`<extendedProperty>`
	 `<name>group.mandatory.attrs</name>`
	 `<values>`
	  `<value>cn</value>`
	  `<value>objectClass</value>`
	 `</values>`
	`</extendedProperty>`
	Corresponding User and Role API property: ROLE_MANDATORY_ATTRS

`group.member.attrs`	The attribute of a static role that specifies the distinguished names (DNs) of the members of an enterprise role or group. Extended property.
	Valid in Java EE and Java SE applications.
	Optional.
	Value example:
	`<extendedProperty>`
	 `<name>group.member.attrs</name>`
	 `<values>`
	  `<value>uniqueMember</value>`
	 `</values>`
	`</extendedProperty>`
	Corresponding User and Role API property: ROLE_MEMBER_ATTRS

`group.object.classes`	The fully qualified names of one or more schema object classes used to represent enterprise roles or groups. Extended property.
	Valid in Java EE and Java SE applications.
	Optional.
	Value example:
	`<extendedProperty>`
	 `<name>group.object.classes</name>`
	 `<values>`
	  `<value>top</value>`
	  `<value>groupOfUniqueNames</value>`
	 `</values>`
	`</extendedProperty>`
	Corresponding User and Role API property: ROLE_OBJECT_CLASSES

Table F–6 (Cont.) LDAP-Based Identity Store Properties

Name	Description
`group.selected.create.base`	The base DNs for creating enterprise roles or groups. Valid in Java EE and Java SE applications. Optional. Value example: `cn=users,dc=us,dc=abc,dc=com` (single DN) Corresponding User and Role API property: ROLE_SELECTED_CREATEBASE
`groupname.attr`	The attribute that uniquely identifies the name of the enterprise role or group. Valid in Java EE and Java SE applications. Optional. Value example: `cn` Corresponding User and Role API property: ROLE_NAME_ATTR
`group.selected.search.base`	The base DNs for searching enterprise roles or groups. Valid in Java EE and Java SE applications. Optional. Value example: `cn=users,dc=us,dc=abc,dc=com` (single DN)
`max.search.filter.length`	The maximum number of characters of the search filter. Valid in Java EE and Java SE applications. Optional. Value: a positive integer. Corresponding User and Role API property: MAX_SEARCHFILTER_LENGTH
`search.type`	The type of search to employ when the repository is queried. Valid in Java EE and Java SE applications. Optional. Valid values: `SIMPLE`, `PAGED`, or `VIRTUAL_LIST_VIEW`. Corresponding User and Role API property: IDENTITY_SEARCH_TYPE
`user.filter.object.classes`	The fully qualified names of object classes used to search users. Extended property. Valid in Java EE and Java SE applications. Optional. Value example: `inetOrgPerson` Corresponding User and Role API property: USER_FILTER_OBJECT_CLASSES
`user.login.attr`	The login identity of the user. Valid in Java EE and Java SE applications. Optional. Value example: `<property name="user.login.attr" value="mail"/>` Corresponding User and Role API property: USER_LOGIN_ATTR

Table F–6 (Cont.) LDAP-Based Identity Store Properties

Name	Description

`user.mandatory.attrs`	The attributes that must be specified when creating a user. Extended property.
	Valid in Java EE and Java SE applications.
	Optional.
	Value example:
	`<extendedProperty>`
	` <name>user.mandatory.attrs</name>`
	` <values>`
	`  <value>cn</value>`
	`  <value>objectClass</value>`
	`  <value>sn</value>`
	` </values>`
	`</extendedProperty>`
	Corresponding User and Role API property: USER_MANDATORY_ATTRS

`user.object.classes`	The fully qualified names of the schema classes used to represent users. Extended property. Valid in Java EE and Java SE applications. Optional. Corresponding User and Role API property: USER_OBJECT_CLASSES
`username.attr`	The LDAP attribute that uniquely identifies the name of the user. Valid in Java EE and Java SE applications. Optional. Corresponding User and Role API property: USER_NAME_ATTR
`ldap.host`	The name of the system hosting the identity store. Valid in Java EE and Java SE applications. Optional.
`subscriber.name`	The default realm for the identity store. Valid in Java EE and Java SE applications. Optional. Value example: `dc=us,dc=oracle,dc=com`. Corresponding User and Role API property: ADF_IM_SUBSCRIBER_NAME
`virtualize`	Controls the authenticators where search and modifications are allowed; if set to TRUE, searching and modifying is available in all configured authenticators; otherwise, if set to FALSE, searching and modifying is available in only the first authenticator in the configured stack. Set to TRUE if you intend to use the User and Role API to search or write information in all authenticators. Valid in Java EE and Java SE applications. Optional. Valid values: `TRUE` or `FALSE`. Default value: `FALSE`. Value example: `<property name="virtualize" value="true"/>`

Note: If the authenticator attribute `username` is changed (because, for example, of post-provisioning or migrating from a test to a production environment), then the identity store service parameter `username.attr` in the identity store service must also be changed accordingly. Those two values should be kept equal.

The following fragment illustrates the configuration of an OID-based identity store for a Java SE application:

```
<serviceInstance name="idstore.ldap" provider="idstore.ldap.provider">
    <property name="idstore.type" value="OID"/>
    <property name="ldap.url" value="ldap://myHost.com:1234"/>
    <extendedProperty>
       <name>user.search.bases</name>
          <values>
             <value>cn=users,dc=us,dc=oracle,dc=com</value>
          </values>
    </extendedProperty>
    <extendedProperty>
       <name>group.search.bases</name>
          <values>
             <value>cn=groups,dc=us,dc=oracle,dc=com</value>
          </values>
    </extendedProperty>
</serviceInstance>
```

F.2.5 Properties Common to All OID-Based Instances

Table F–7 lists properties common to all OID-based stores that can be specified in any service instance.

In the case of an OID-based identity store service instance, to ensure that the User and Role API picks up the connection pool properties when it is using the JNDI connection factory, the identity store service instance *must* include the following property:

```
<property
name="INITIAL_CONTEXT_FACTORY" value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

Table F–7 Generic OID Properties

Name	Description
`connection.pool.authentication`	Specifies the type of OID connection that the JNDI connection pool uses. Valid in Java EE and Java SE applications. Optional. Values: `none`, `simple`, and `DIGEST-MD5`. Default value: `simple`.
`connection.pool.max.size`	Specifies the maximum number of connections in the OID connection pool. Valid in Java EE and Java SE applications. Optional. Value example: 30
`connection.pool.min.size`	Specifies the minimum number of connections in the OID connection pool. Valid in Java EE and Java SE applications. Optional. Value example: 5
`connection.pool.protocol`	Specifies the protocol to use for the OID connection. Valid in Java EE and Java SE applications. Optional. Values: `plain`, `ssl`. Default value: `plain`.
`connection.pool.provider.type`	Specifies the connection pool to use. Valid in Java EE and Java SE applications. Optional. Values: `JNDI`, `IDM`. Default value: `JNDI`.
`connection.pool.timeout`	Specifies the number of milliseconds that an idle connection can remain in the pool; after timeout, the connection is closed and removed from the pool. Valid in Java EE and Java SE applications. Optional. Default value: 300000 (5 minutes)
`oracle.security.jps.ldap.max.retry`	Specifies the maximum number of retry attempts if there are problems with the OID connection. Valid in Java EE and Java SE applications. Optional. Value example: 5

The following fragment illustrates a configuration of several properties:

```
<jpsConfig ... >
   ...
   <!-- common properties used by all LDAPs -->
   <property name="oracle.security.jps.farm.name" value="cn=OracleFarmContainer"/>
   <property name="oracle.security.jps.ldap.root.name"
```

```
                value="cn=OracleJpsContainer"/>
      <property name="oracle.security.jps.ldap.max.retry" value="5"/>
      ...
   </jpsConfig>
```

F.2.6 Anonymous and Authenticated Roles Properties

Table F–8 lists the properties that can be used to configure file-, OID-, or DB-based anonymous users, anonymous roles, and authenticated roles.

Table F–8 Anonymous and Authenticated Roles Properties

Name	Description
`anonymous.role.description`	Specifies a description of the anonymous role. Valid in Java EE and Java SE applications. Optional. No default value.
`anonymous.role.name`	Specifies the name of the principal in the anonymous role. Valid in Java EE and Java SE applications. Optional. Default value: `anonymous-role`
`anonymous.role.uniquename`	Specifies the name of the anonymous role. Valid in Java EE and Java SE applications. Optional. Default value: `anonymous-role`
`anonymous.user.name`	Specifies the name of the principal in the anonymous user. Valid in Java EE and Java SE applications. Optional. Default value: `anonymous`
`authenticated.role.description`	Specifies a description of the authenticated role. Valid in Java EE and Java SE applications. Optional. No default value.
`authenticated.role.name`	Specifies the name of the principal in authenticated user roles. Valid in Java EE and Java SE applications. Optional. Default value: `authenticated-role`
`authenticated.role.uniquename`	Specifies the name of the authenticated role. Valid in Java EE and Java SE applications. Optional. Default value: `authenticated-role`

Table F–8 (Cont.) Anonymous and Authenticated Roles Properties

Name	Description
`remove.anonymous.role`	Specifies whether the anonymous role should be removed from the subject after a user is authenticated. Valid in Java EE and Java SE applications. Optional. Valid values: `TRUE`, `FALSE`. Default value: `FALSE`.

F.2.7 Trust Service Properties

Table F–9 lists the properties specific to the trust service.

Table F–9 Trust Service Properties

Name	Description
`trust.keystoreType`	Specifies the type of the trust service store: JKS or KSS. Valid in Java EE and Java SE applications. Optional. Valid values: `JKS` of `KSS`. Default: none. When unspecified, if a KSS store is already provisioned, then the value is set to `KSS`; otherwise it is set to `JKS`.
`trust.keyStoreName`	Applies only when `trust.keystoreType` is set to `KSS`, and it specifies the store name with the format: `kss://<stripeName>/<keyStoreName>` Valid in Java EE and Java SE applications. Optional. Default: `kss://opss/trustservice_ks`.
`trust.trustStoreName`	Applies only when `trust.keystoreType` is set to `KSS`, and it specifies the store URL with the format: `kss://<stripeName>/<keyStoreName>` Valid in Java EE and Java SE applications. Optional. Default: `kss://opss/trustservice_ts`.
`trust.aliasName`	Specifies the alias to use to get an X.509 certificate and private key from the keystore. Valid in Java EE and Java SE applications. Optional. Default: the name of the WLS domain.
`trust.issuerName`	Specifies the name to be included in the token. It is used by the destination trust service to pick up and validate the token. Valid in Java EE and Java SE applications. Optional. Default: the name of the WLS domain.

Table F–9 (Cont.) Trust Service Properties

Name	Description
`trust.provider.className`	Specifies the fully-qualified name of the trust provider class. Valid in Java EE and Java SE applications. Required. Value: `the only supported value is oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl`.
`trust.clockSkew`	Specifies, in seconds, the time-gap allowed when verifying time conditions. Valid in Java EE and Java SE applications. Optional. Default: 0.
`trust.token.validityPeriod`	Specifies, in seconds, the time that a token remains valid after being issued. Valid in Java EE and Java SE applications. Required. Default: none.
`trust.csf.map`	Specifies the map of the credential to access the keystore. Valid in Java EE and Java SE applications. Optional. Default: the value of the keystore instance property `keystore.csf.map`.
`trust.csf.keystorePass`	Applies only when `trust.keystoreType` is set to `JKS`, and it specifies the key of the credential to access the private key (the map is set by `trust.csf.map`). Valid in Java EE and Java SE applications. Optional. Default: the value of the keystore instance property `keystore.pass.csf.key`.
`trust.csf.keyPass`	Applies only when `trust.keystoreType` is set to `JKS`, and it specifies the key of the credential to acces the keystore (the map is set by `trust.csf.map`). Valid in Java EE and Java SE applications. Optional. Default: the value of the keystore instance property `keystore.sig.csf.key`.
`trust.token.includeCertificate`	Specifies whether the SAML token includes a certificate. Valid in Java EE and Java SE applications. Required. Valid values: `true` or `false`. Default: `false`.

The following sample illustrates the configuration of a trust service:

```
<propertySet name="trust.provider.embedded">
```

```
  <property name="trust.provider.className"
value="oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl"/
>
  <property name="trust.clockSkew" value="60"/>
  <property name="trust.token.validityPeriod" value="1800"/>
  <property name="trust.aliasName" value="orakey"/>
  <property name="trust.issuerName" value="orakey"/>
  <property name="trust.csf.map " value="my-csf-map"/>
  <property name="trust.csf.keystorePass" value="my-keystore-csf-key"/>
  <property name="trust.csf.keypass" value="my-signing-csf-key"/>
</propertySet>
```

F.2.8 Audit Service Properties

Table F–10 lists the properties specific to the audit service. Additional properties are listed in Common Properties.

Table F–10 Audit Service Properties

Property	Description	Required?	Values	Default Value
audit.filterPreset	The level of auditing.	no	None, Low, Medium, or High	None
audit.customEvents	For Custom, a list of audit events that should be audited. The events must be qualified using the component type. Commas separate events and a semicolon separates component types. Example: `JPS:CheckAuthorization, CreateCredential; OIF:UserLogin`	no		
audit.specialUsers	list of one or more users whose activity is always audited, even if filterPreset is None. Usernames that contain commas must be escaped properly. For example, when using Fusion Middleware Control, specify three users like this - "admin, fmwadmin, cn=test\,cn=user\,ou:ST\,L=RS\,c=is\, " In `WLST`, the backslash "\" should also be escaped. For example: `setAuditPolicy(addSpecialUsers="cn =orcladmin\\\,cn=com")` For more information, see Appendix C, "Oracle Fusion Middleware Audit Framework Reference.".	no		
audit.maxFileSize	Controls the size of a bus stop file where audit events are written. Integer is in Bytes	no		104857600
audit.loader.interval	Controls the frequency with which audit loader uploads to database. Integer is in Seconds.	no		15 seconds
audit.loader .repositoryType	Store type for the audit events. If type is Database (DB), also define audit.loader.jndi or JDBC property.	yes	File, DB	File

Table F–10 (Cont.) Audit Service Properties

Property	Description	Required?	Values	Default Value
audit.loader.jndi	JNDI name of the data source in application servers for uploading audit events into database.	no		jdbc/AuditAppendDataSource
audit.db.principal.map / audit.db.principal.key	The map and key for the JDBC user name and password credential in bootstrap credential store,when running in JavaSE, and repositoryType is DB.	no		
audit.loader.jdbc.string	The JDBC string for JDBC connection when running in JavaSE, and repositoryType is DB.	no		
audit.logDirectory	The base directory for bus-stop files.	required for JavaSE		jse
audit.timezone	Determines whether events are recorded in the Oracle WebLogic Server timestamp or in UTC. For details, see Section 14.4.	no	utc, local	utc
audit.change.scanning.interval	In a distributed environment with OID- or DB-based audit store, the audit service monitors each OPSS running instance for changes in audit runtime policies, and dynamically re-loads any cached policies. This property determines how frequently, in milliseconds, the service checks for any changes.	no	whole number greater than zero	60000 (60 seconds)

The following is an example of audit service configuration:

```
<serviceInstance name="audit" provider="audit.provider"
location="./audit-store.xml">
   <property name="audit.filterPreset" value="Medium"/>
   <property name="audit.loader.jndi" value="jdbc/AuditAppendDataSource"/>
   <property name="audit.loader.repositoryType" value="DB" />
   <property name="server.type" value="DB_ORACLE"/>
   <property name="audit.timezone" value="local" />
 </serviceInstance>
```

F.2.9 Keystore Service Properties

Table F–11 lists the properties specific to the keystore service. Additional properties are listed in Common Properties.

Table F–11 Keystore Service Properties

Property	Description	Required?	Values	Default
keystore.file.path	Location of the file keystores.xml when file provider is configured	Yes, if a file-based keystore provider is configured.	-	./
ca.key.alias	Key alias within "system/castore" of the third party CA used for Keystore service instance	No	-	-
location	Location of the keystore; can be absolute or relative path.	Yes, if keystore.type is JKS.No, if keystore.type is PKCS11 or HSM (LunaSA)	Path to keystore	./default-keystore.jks
keystore.type	Type of keystore.	No	KSS, JKS, PKCS11, Luna	JKS
keystore.csf.map	Credential store map name used by Oracle Web Services Manager. Used only by OWSM.	No	Credential store map name	oracle.wsm.security
keystore.pass.csf.key	Credential store key that points to Keystore password. Used only by OWSM.	No	Credential store csf key name	keystore-csf-key
keystore.sig.csf.key	Credential store key name that points to alias and password of signing key in keystore.For HSM, it is the direct key alias name rather than the credential store key name. Used only by OWSM.	No	Credential store csf key name or, for HSM, the direct alias	sign-csf-key
keystore.enc.csf.key	Credential store key name that points to alias and password of encryption key in keystore.For HSM, it is the direct key alias name rather than the credential store key name. Used only by OWSM.	No	Credential store csf key name or, for HSM, the direct alias	enc-csf-key

The following is an example of Keystore Service configuration:

```
<propertySet name="props.ldap.1">
  <property name="java.naming.ldap.derefAliases" value="never"/>
  <property name="bootstrap.security.principal.key" value="bootstrap_
6aCNhgRM3zF04ToliwecdF6K3oo="/>
  <property name="oracle.security.jps.farm.name" value="cn=compact1_oid26008"/>
  <property name="server.type" value="OID"/>
  <property name="oracle.security.jps.ldap.root.name" value="cn=jpsTestNode"/>
  <property name="ldap.url" value="ldap://myComp.com:2020"/>
</propertySet>

<serviceProvider type="KEY_STORE" name="keystore.provider"
class="oracle.security.jps.internal.keystore.KeyStoreProvider">
  </serviceProvider>
<serviceInstance name="keystore.ldap" provider="keystore.provider">
  <propertySetRef ref="props.ldap.1"/>
</serviceInstance>
```

The following is an example of Keystore Service configuration for an LDAP-based provider:

```
<serviceInstance name="keystore" provider="keystore.provider"
      location="./default-keystore.jks">
   <description>Default JPS Keystore Service</description>
   <property name="server.type" value="OID"/>
   <property name="keystore.type" value="JKS"/>
   <property name="keystore.csf.map" value="oracle.wsm.security"/>
   <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
   <property name="keystore.sig.csf.key" value="sign-csf-key"/>
   <property name="keystore.enc.csf.key" value="enc-csf-key"/>
<property value="bootstrap" name="bootstrap.security.principal.key"/>
<property value="cn=wls-jrfServer" name="oracle.security.jps.farm.name"/>
<property value="cn=jpsTestNode" name="oracle.security.jps.ldap.root.name"/>
<property value="ldap://myHost.com:1234" name="ldap.url"/>
</serviceInstance>
```

The following is an example of Keystore Service configuration for an DB-based provider:

```
<propertySet name="props.db.1">
   <property name="jdbc.url" value="jdbc:oracle:thin:@host:port:sid"/>
   <property name="oracle.security.jps.farm.name" value="cn=farm"/>
   <property name="server.type" value="DB_ORACLE"/>
   <property name="oracle.security.jps.ldap.root.name" value="cn=jpsroot"/>
   <property name="jdbc.driver" value="oracle.jdbc.OracleDriver"/>
   <property name="bootstrap.security.principal.map" value="credendial_map"/>
   <property name="bootstrap.security.principal.key" value="credential_key"/>
</propertySet>

…
…
<serviceInstance name="keystore.rdbms" provider="keystore.provider"
           location="./default-keystore.jks">
   <propertySetRef ref = "props.db.1"/>
   <property name="server.type"  value="DB_ORACLE"/>
   <property name="keystore.type" value="JKS"/>
   <property name="keystore.csf.map" value="oracle.wsm.security"/>
   <property name="keystore.pass.csf.key" value="keystore-csf-key"/>
   <property name="keystore.sig.csf.key" value="sign-csf-key"/>
   <property name="keystore.enc.csf.key" value="enc-csf-key"/>
</serviceInstance>
```

G OPSS API References

This appendix contains references to OPSS APIs.

The list of available OPSS APIs is presented in the following section:

- Section G.1, "OPSS API References"

G.1 OPSS API References

The following Javadoc documents describe the various APIs that OPSS exposes:

OPSS APIs

Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services

OPSS MBean APIs

Oracle Fusion Middleware Java API Reference for Oracle Platform Security Services Mbeans

OPSS User and Role APIs

Oracle Fusion Middleware User and Role Java API Reference for Oracle Platform Security Services

IDS APIs

Oracle Fusion Middleware Java API Reference for Identity Governance Framework Identity Directory

H

Using an OpenLDAP Identity Store

This appendix describes the special set up required in case the identity store uses OpenLDAP 2.2.

It includes the following section:

H.1 Using an OpenLDAP Identity Store

To use OpenLDAP 2.2 as an identity store, proceed as follows:

1. Use the WebLogic Server administration console to create a new authenticator provider. For this new provider:
   - Select OpenLDAPAuthenticator from the list of authenticators.
   - Set the control flag of the OpenLDAPAuthenticator to SUFFICIENT.
   - Set the control flag of the DefaultAuthenticator to SUFFICIENT.
   - Change the order of authenticators to make the OpenLDAPAuthenticator the first in the list.
   - In the Provider Specific page for the OpenLDAPAuthenticator, enter User Base DN and Group Base DN, and set the value of the objectclass in the Group From Name Filter to something other than groupofnames.
2. From the Home directory of the OpenLDAP installation:
   - Open the file `slapd.conf` for edit.
   - In that file, insert the following line in the "include" section at the top:

     ```
     include ./schema/inetorgperson.schema
     ```

   - Save the file, and restart the OpenLDAP.

The above settings make possible adding the object class `inetorgperson` to every new external role you create in the OpenLDAP; this object class is required to map the external role to an application role.

I

Adapter Configuration for Identity Virtualization

The identity virtualization feature, described in Section 8.3, "Configuring the Identity Store Service", requires some additional configuration to support a split profile. This appendix describes how to create and manage the adapters used for split profiles:

This appendix includes the following sections:

I.1 About Split Profiles

The Identity Virtualization feature enables you to query multiple LDAP directories through OPSS. For example, you can fetch data from both Oracle Internet Directory and Microsoft Active Directory in a single query.

The feature supports a "split profile" file, where an application makes use of attributes for a single identity that are stored on two different sources; for example, where the username, password, and employeeID for a single person are stored on Microsoft Active Directory, and that person's employeeID and business role are stored in Oracle Internet Directory.

For example, when a WebCenter application needs to obtain attributes for a single identity from more than one source directory, it uses the split profile to leverage the join functionality of Identity Virtualization. These joins use a standard join adapter. For details, see:

- Understanding Oracle Virtual Directory Adapters in the *Oracle Fusion Middleware Administrator's Guide for Oracle Virtual Directory*
- Understanding the Join View Adapter in the *Oracle Fusion Middleware Administrator's Guide for Oracle Virtual Directory*

The adapter configuration is stored in `adapters.os_xml`, but connection details such as host, port and credentials of a back-end directory come from OPSS.

I.2 Configuring a Split Profile

The same user occurs in both identity stores with some attributes in one store and other attributes in the other store. A query on the user record requires data from both stores. The configuration tasks are:

1. Configure the identity store service with the `virtualize` property to enable queries against multiple LDAP stores.

   For details, see Section 8.3, "Configuring the Identity Store Service."

2. Connect to the Weblogic Admin server to run commands to configure the join adapter for the identity stores. For information about the available WLST commands, see "Library Oracle Virtual Directory (LibOVD) Commands" in *WLST Command Reference for WebLogic Server*.

   For details about how to bring up the WLST prompt, see "Getting Started Using Command-Line Tools" in the *Administering Oracle Fusion Middleware*.

   For information about the

3. Create the join adapter in the primary identity store:

   ```
   createJoinAdapter(adapterName="Join Adapter Name",  root="Namespace",
   primaryAdapter="Primary adapter Name")
   ```

4. Add the join rule to the secondary store(s):

   ```
   addJoinRule(adapterName="Join Adapter Name", secondary="Secondary Adapter
   Name", condition="Join Condition")
   ```

   Note: If there is more than one secondary identity store, run the `addJoinRule` command for each secondary store.

5. Run the `modifyLDAPAdapter` command:

   ```
   modifyLDAPAdapter(adapterName="AuthenticatorName", attribute="Visible",
   value="Internal")
   ```

   Note: If there is more than one secondary identity store, run the `modifyLDAPAdapter` command for each secondary ID store.

Example

In this example the same user occurs in two stores; the first store is Microsoft Active Directory and the second store is Oracle Internet Directory. In the example, we assume that Microsoft Active Directory is the primary store and Oracle Internet Directory is the secondary store.

Note: When configuring the LDAP connection parameters, the `user.create.bases` and `group.create.bases` must correspond to the primary adapter's namespace. For details about the parameters, see Section 8.3.1, "What is Configured?."

Authenticator 1
Authenticator Name: Microsoft Active Directory (AD)

User Base: `cn=users,dc=acme,dc=com`

Authenticator 2
Authenticator Name: Oracle Internet Directory (OID)
User Base: `cn=users,dc=oid,dc=com`

The steps to implement the split profile are as follows:

1. Create the join adapter:

   ```
   createJoinAdapter(adapterName="JoinAdapter1", root="dc=acme,dc=com",
   primaryAdapter="AD")
   ```
   The adapter name shown here is an example; use an appropriate name in actual usage.

2. Specify the join rule:

   ```
   addJoinRule(adapterName="JoinAdapter1", secondary="OID", condition="uid=cn")
   ```

   "`uid=cn`" is the join condition in the above example which indicates that if `uid` value of a user in Oracle Internet Directory (secondary) matches with `cn` value of the Microsoft Active Directory user (primary), then the attributes are combined.

   The attribute on the left side of the condition is the attribute in the secondary adapter and the attribute on the right side is the attribute in the primary adapter.

3. Modify the adapters:

   ```
   modifyLDAPAdapter(adapterName="OID", attribute="Visible", value="Internal")

   modifyLDAPAdapter(adapterName="AD", attribute="Visible", value="Internal")
   ```

   The adapter names used here are the actual name of the authenticators. The adapter names in all the primary and secondary parameters also refer to the authenticator name. The join adapter name can be any name you choose.

4. Restart Weblogic Admin and Managed servers.

I.3 Deleting a Join Rule

You use the `removeJoinRule` command to remove a join rule from a join adapter.

Syntax

removeJoinRule
adapterName ="*adapterName*"
secondary="*Secondary Adapter associated with the JoinRule*"

Example

```
removeJoinRule(adapterName="JoinAdapter1", secondary="OID")
```

I.4 Deleting a Join Adapter

You use the `deleteAdapter` command to delete a join adapter.

Syntax

deleteAdapter(adapterName="*name*")

Example

```
deleteAdapter(adapterName="JoinAdapter1")
```

I.5 Changing Adapter Visibility

You use the `modifyLDAPAdapter` command to change the visibility of the adapters. For example:

```
modifyLDAPAdapter(adapterName="AuthenticatorName", attribute="Visible",
value="Yes")
```

I.6 Enabling Access Logging for Identity Virtualization Library

Enabling access logging for Identity Virtualization Library allows you to capture all requests and responses flowing through Identity Virtualization Library, which can be very useful if you are trying to triage performance issues.

To enable access logging for Identity Virtualization Library:

1. Remove any Identity Virtualization Library loggers that were previously configured in Debug mode. You must remove these loggers to see real performance numbers.
2. Create a WebLogic logger named `oracle.ods.virtualization.accesslog` in WebLogic Server with NOTIFICATION level.
3. Create a WebLogic loghandler, specifying a file name similar to ovd-access.log and associate that log handler to the logger you created in step 2.

   This loghandler logs all Oracle Virtual Directory access log messages into a separate file.
4. Create a backup of the *DOMAIN_HOME*/config/fmwconfig/ovd/default/provider.os_xml file, and then add the following XML fragment (if it is not already present):

   ```
   <providers ..>
      ...
      <auditLogPublisher>
         <provider name="FMWAuditLogPublisher">
           ...
         </provider>
         <provider name="AccessLogPublisher">

   <configClass>oracle.ods.virtualization.config.AccessLogPublisherConfig</configC
   lass>
            <properties>
               <property name="enabled" value="true"/>
            </properties>
         </provider>
      </auditLogPublisher>
      ...
   </providers>
   ```
5. Restart the WebLogic Server Admin and Managed servers.

Oracle Virtual Library can now generate the access log in the ovd-access.log file.

J Troubleshooting OPSS

This appendix describes common problems that you may encounter when configuring and using OPSS, and explains how to solve them.

It contains the following sections:

- Diagnosing Security Errors
- The OPSS Diagnostic Framework
- Troubleshooting Reassociation and Migration
- Troubleshooting Server Starting
- Troubleshooting Permissions
- Troubleshooting Connections and Access
- Troubleshooting Oracle Business Intelligence Reporting
- Troubleshooting Searching
- Troubleshooting Versioning
- Troubleshooting Other Errors
- Need Further Help?

J.1 Diagnosing Security Errors

This section describes the tools available to diagnose and solve a variety of security errors. It contains the following sections:

- About Log Files and OPSS Loggers
- System Properties
- Solving Security Errors

The logging support with Fusion Middleware Control is explicitly stated when the tool can help managing, isolating, or interpreting faults.

J.1.1 About Log Files and OPSS Loggers

This section describes the various log files and OPSS loggers supported by Oracle WebLogic Server; it also explains how to configure, set logger levels, and locate and view log files with Fusion Middleware Control and WLST commands, in the following sections:

- About Diagnostic Log Files

- About Generic Log Files
- About Authorization Loggers
- Offline WLST Commands Loggers
- Other OPSS Loggers
- About User and Role API Logger
- About Audit Loggers
- Managing Loggers with Fusion Middleware Control
- Managing Loggers with a Script

J.1.1.1 About Diagnostic Log Files

Each server instance in a domain writes all OPSS-based exceptions raised by its subsystems and applications to a server log file in the file system of the local host computer.

By default, this log file is located in the `logs` directory below the server instance root directory. The names of these log files have the following format: *ServerName*`-diagnostic.logxxxxx`, where xxxxx denotes an integer between 1 and 99999.

Here are some examples of diagnostic file full names: *DomainName*`/servers/AdminServer/logs/AdminServer-diagnostic.log00001` (administration server log), *DomainName*`/servers/soa/logs/soa-diagnostic.log00013` (managed server log).

All server instances output security-related errors to diagnostic files. Server-related security errors, such as exceptions raised by issues with a subject or principal, and errors that may occur while migrating or reassociating domain security data, get written in the administration server diagnostic log. Application-related security errors, such as exceptions raised by application-specific policies or credentials, get written in the corresponding managed server diagnostic log.

J.1.1.2 About Generic Log Files

In addition to diagnostic log files, Oracle WebLogic Server supports other log files for each server in a domain and for each domain in a topology.

By default and similar to diagnostic log files, server log files are located in the `logs` directory below the server instance root directory. Domain log files are located in the `logs` directory below the administration server root directory. The names of these log files have the format *ServerName*`.logxxxxx` and `domain.logxxxxx`, where xxxxx denotes an integer between 1 and 99999.

Here are some examples of server and domain log files full names: *DomainName*`/servers/AdminServer/logs/AdminServer.log00001`, *DomainName*`/servers/AdminServer/logs/domain1.log00033`.

Server and domain logs are files where one should look for generic errors, such as exception raised by authenticators or other domain service providers.

The domain logs duplicate some messages written to server logs (for servers in the domain), and they help determine the server on which a fault has occurred in a domain that contains a large number of servers.

Note: The generation of a new log file is determined by its rotation policy; typically, the rotation is determined by file size, so when a log file exceeds a specified size, the system generates a new one with a name whose integer suffix is increased by 1.

For details about particular loggers, see About Authorization Loggers and About Audit Loggers.

Related Documentation

For information about server log files and domain log files, see section Server Log Files and Domain Log Files in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

For information about the Oracle WebLogic Framework, see *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

For additional information about logging services, see *Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server*.

For complete details about logging in Oracle Fusion Middleware, see chapter Managing Log Files and Diagnostic Data in *Administering Oracle Fusion Middleware*.

J.1.1.3 About Authorization Loggers

OPSS provides several loggers that help troubleshooting runtime authorization failures:

- oracle.security.jps.util.JpsAuth
- oracle.security.jps.trace.logger
- oracle.jps.authorization
- oracle.jps.common
- oracle.security.jps.az.internal.runtime.policy.AbstractPolicyImpl
- oracle.security.jps.internal.policystore.JavaPolicyProvider

These loggers can be enabled and disabled dynamically, that is, without having to stop and restart the Oracle WebLogic Application Server; for details about setting the properties of a logger, see Managing Loggers with Fusion Middleware Control. Typically, the level of loggers is set to `TRACE:32`.

For information about additional loggers, see Other OPSS Loggers.

J.1.1.3.1 oracle.security.jps.util.JpsAuth The logger `oracle.security.jps.util.JpsAuth` logs the start and return of the method `checkPermission`; the following snippets of a log file illustrate the entry and exit demarcations to this method in the log file:

```
[SRC_CLASS: oracle.security.jps.util.JpsAuth] [APP: JeeScenarioApp]
[SRC_METHOD: Entering checkPermission] ENTRY
(oracle.security.jps.ResourcePermission
resourceType=TaskFlowResourceType,resourceName=ResourceNameX read)

[SRC_CLASS: oracle.security.jps.util.JpsAuth] [APP: JeeScenarioApp]
[SRC_METHOD: Exiting checkPermission] RETURN
java.security.AccessControlException: access denied
(oracle.security.jps.ResourcePermission
resourceType=TaskFlowResourceType,resourceName=ResourceNameX read)
```

The following snippet illustrates a successful authorization log:

```
[JpsAuth] Check Permission
PolicyContext: [JeeScenarioApp]
Resource/Target: [getSubjectFromDomainCombiner]
Action:[null]
Permission Class: [javax.security.auth.AuthPermission]
       Result:              [SUCCEEDED]
       Subject:             [null]
       Evaluator:           [SM]
```

The following snippet illustrates an unsuccessful authorization log:

```
[JpsAuth] Check Permission
PolicyContext: [JeeScenarioApp]
Resource/Target: [resourceType=TaskFlowResourceType,resourceName=ResourceNameX]
Action:[read]
Permission Class: [oracle.security.jps.ResourcePermission]
       Result:              [FAILED]
       Evaluator:           [ACC]
       Failed
```

J.1.1.3.2 oracle.security.jps.trace.logger The logger `oracle.security.jps.trace.logger` logs information about application roles, permissions, targets, principals, and granted and denied policies. Since enabling this logger can lead to a large output, it is recommended that it be used to debug a single use case only. Specifically, this logger records:

- The following information about an authorization request: the application roles granted to an enterprise role, all deny's and grant's, the permission class names, the permission targets, and the principal names.
- Member cache updates, such as when a principal is added to a role; keywords: "Principal:", "Inserted Roles:".
- Role managing information; keywords: "In App:", "Query Store for Principal:", "Number of direct app roles:".
- Calls to the method `getPermissions`.

J.1.1.4 Offline WLST Commands Loggers

When using an offline WLST commands, such as `migrateSecurityStore`, OPSS loggers can be enabled by starting the JVM with the following system properties:

```
-Dwlst.offline.log
-Dwlst.offline.log.priority
```

Set these properties as follows:

- `wlst.offiline.log` can have one of the following values: `<filename>`, `stdout`, `sterr`, or `disable`. If this property is not set explicitly, the log files get created, by default, in the directory `$MW_HOME/logs`.
- `wlst.offline.log.priority` can have one of the following values: `OFF`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`, `ALL`, `debug`, `info`, `warn`, `error`, `fatal`.

J.1.1.5 Other OPSS Loggers

In addition to authorization loggers, OPSS provides the following loggers:

`oracle.jps.deployment` enables diagnosing issues with OPSS artifacts packed with the application, when the application is deployed; keyword:"migration".

`oracle.jps.openaz` enables diagnosing issues with PEP API calls. Setting `oracle.jps.openaz.level` to `FINEST`, logs information about submitted requests - identity, resource, action, context - and authorization results.

J.1.1.6 About User and Role API Logger

To trace OPSS User and Role API calls, set the logger `oracle.idm.userroleapi` to `TRACE:32`.

J.1.1.7 About Audit Loggers

There are several run-time components in the Fusion Middleware Audit Framework. This section helps you navigate the diagnostic log files for these components and explains how to interpret diagnostic messages.

The log files are located at:

```
DomainName/servers/$SERVER_NAME/logs/$SERVER_NAME-diagnostic.log
```

Table J–1 lists the various diagnostic log files.

Table J–1 Log Files for Audit Diagnostics

Component	Log Location	Configuring Loggers
Java EE Components using Audit APIs	DomainName/servers/$SERVER_NAME/logs/$SERVER_NAME-diagnostic.log	oracle.security.audit.logger (See instructions below)
OPMN Components Using Audit APIs	See Section J.1.1.7.3.	See Section J.1.1.7.3.
Startup Class Audit Loader	DomainName/servers/$SERVER_NAME/logs/$SERVER_NAME-diagnostic.log	oracle.security.audit.logger (See instructions following this table)
OPMN Audit Loader	$ORACLE_INSTANCE/diagnostics/logs/OPMN/opmn/rmd.out	java.util.logging.config.file system property can be set to the file that contains the log level for OPMN Audit Loader
Config/Proxy Mbeans	DomainName/servers/$SERVER_NAME/logs/$SERVER_NAME-diagnostic.log	oracle.security.audit.logger (See instructions below)
Audit Schema Support	RCU log location (Default is $ORACLE_HOME/rcu/log/)RCU_LOG_LOCATION can be set to change this location	RCU log level (Default is ERROR) RCU_LOG_LEVEL - [SEVERE; ERROR; NOTIFICATION; TRACE

J.1.1.7.1 Configuring the Audit Loggers

You can configure oracle.security.audit.logger using Fusion Middleware Control.

oracle.security.audit.logger can take any log level from ERROR to TRACE allowing control over the amount of information that gets logged.

You can also view these diagnostic files with Fusion Middleware Control.

See Also: For more information about the following topics, see chapter 10, Managing Log Files and Diagnostic Data, in *Administering Oracle Fusion Middleware*:

- instructions for configuring the loggers
- details on viewing logs from domain, server, and each application

J.1.1.7.2 Interpreting Audit Diagnostics

The Audit diagnostic messages can be categorized into two types - errors and trace messages.

All error messages are numbered IAU-XXX. These messages are found in the Error Message Guide with a proper cause and an action that can be taken to rectify the error.

The trace messages, however, are meant to provide more information about the running components. Depending on its nature, a message may require some action on your part.

J.1.1.7.3 Configuring Audit Logging for OPMN Components

You can configure audit logging for system components managed with OPMN.

Using the `auditconfig.xml` configuration file, you can specify the log directory location to which the component's audit logs should be written. Use the `LogsDirectory` element to specify the audit log location. In the following example, the log is located at `/tmp/audit/auditlogs`:

```
<LogsDirectory>
   <MaxFileSize></MaxFileSize>
   <Location>/tmp/audit/auditlogs</Location>
</LogsDirectory>
```

J.1.1.8 Managing Loggers with Fusion Middleware Control

Fusion Middleware Control provides several pages to manage log information. Using this tool you can:

- Configure several attributes of a log file, including the log level and rotation.
- Search the contents of all log files in a domain and group the results of a query by message ID or type.
- Correlate a given error with others by context or time span.
- Download a portion of a log file or the results of a query in one of several formats.

This section explains briefly how to configure a log file. The other functions above are explained, also briefly, in section Section J.1.3, "Solving Security Errors."

For full details about these topics, see section Managing Log Files and Diagnostic Data, in the *Administering Oracle Fusion Middleware*.

To configure a log file with Fusion Middleware Control, proceed as follows:

1. Navigate to *Server* > **Logs** > **Log Configuration**, to display the **Log Configuration** page for the selected server. This page allows you to configure the log level for both persistent loggers and active run-time loggers.
2. Click the Log File entry for the desired logger, to display the page showing the current parameter settings for that file.

3. In this page, select a row and then click the button **Edit Configuration**, to display the **Edit Log File** dialog, where you can set various parameters, including the log level and the rotation policy; typically, the logger level is set to TRACE:32.

To specify a logger not visible in Fusion Middleware Control, proceed as follows:

1. Navigate to *Server* > **Logs** > **Log Configuration**, to display the **Log Configuration** page for the selected server.
2. In the **Log Levels** tab and from the pull-down **View**, select **Loggers With Persistent Log Level State**.
3. In the area **Specify Loggers** at the bottom of the page, enter the logger name and select a level for it from the pull-down **Oracle Diagnostic Logging Level (Java Level)**.

J.1.1.9 Managing Loggers with a Script

You can manage loggers using the script `setLoggerLevel` as illustrated in the following invocation:

```
setLogLevel(logger="oracle.idm.userroleapi", level="TRACE:32", addLogger=1,
persist=1)
```

J.1.2 System Properties

To increase the debug output, set the following system properties in the script that starts your Oracle WebLogic Server and restart the server:

- jps.auth.debug
- jps.auth.debug.verbose

To get debug output during the authorization process, set any of the system properties described in section Debugging the Authorization Process.

Two other system properties that can be passed at server start and that can help debugging security issues are the following:

- `-DDebugOPSSPolicyLoading`, a flag that monitors the progress and setting of the OPSS policy provider.
- `-Djava.security.debug=policy`, the standard Java security debug flag that produces print information about policy files as they are parsed, including their location in the file system, the permissions they grant, and the certificates they use for signed code.

Note: A consequence of setting a high logging output is that many threads may be reported in a stuck state, specially when file loading takes place. To avoid this situation, change the time out value that Oracle WebLogic Server uses to mark a thread as stuck to a higher value.

A system property cannot be set without restarting the server. In order to set a system property the administrator must edit the `setDomainEnv.sh` shell script and add the property to the environment variable `EXTRA_JAVA_PROPERTIES` in that script.

J.1.2.1 jps.auth.debug

Assume that just this system property is set to true:

```
-Djps.auth.debug=true
```

Then, a permission check that fails generates an output with details illustrated in the following sample:

```
[JpsAuth] Check Permission
          PolicyContext:        [jps-wls-Demo]
          Resource/Target:      [app.monitor]
          Action:               [read,write]
          Permission Class:     [java.util.PropertyPermission]
          Evaluator:            [ACC]
          Result:               [FAILED]
Failed ProtectionDomain:ClassLoader=weblogic.servlet.jsp.JspClassLoader@fb111c
finder: weblogic.utils.classloaders.CodeGenClassFinder@106bb21 annotation:
CodeSource=file:/C:/MyOracle/domains/base_domain/servers/AdminServer/tmp/_WL_
user/jps-wls-Demo/kebqfo/jsp_servlet/test.class
Principals=total 5 of principals(
 1. weblogic.security.principal.WLSUserImpl "duane"
 2. weblogic.security.principal.WLSGroupImpl "employee"
 3. JpsPrincipal: oracle.security.jps.principals.JpsAuthenticatedRoleImpl
"authenticated-role"
 4. JpsPrincipal: oracle.security.jps.principals.JpsAnonymousRoleImpl
"anonymous-role"
 5. JpsPrincipal: oracle.security.jps.service.policystore.ApplicationRole
"appRoleEmployee")
 Permissions=(
 (java.util.PropertyPermission line.separator read)
 ...
 (oracle.security.jps.service.credstore.CredentialAccessPermission
context=SYSTEM,mapName=default,keyName=* read,write))
```

A permission check that succeeds generates no output. To disable permission check messages, set this property to false; by default, it is set to true. Disabling persmission check messages is not recommended in production environments.

J.1.2.2 jps.auth.debug.verbose

Assume that `jps.auth.debug` and `jps.auth.debug.verbose` are *both* set to true:

```
-Djps.auth.debug=true
-Djps.auth.debug.verbose=true
```

Then, a permission check that succeeds generates an output with details illustrated in the following sample:

```
[JpsAuth] Check Permission
          PolicyContext:        [jps-wls-Demo]
          Resource/Target:      [app.monitor]
          Action:               [read,write]
          Permission Class:     [java.util.PropertyPermission]
          Result:               [SUCCEEDED]
          Subject:              [total 5 of principals(
 1. weblogic.security.principal.WLSGroupImpl "manager"
 2. weblogic.security.principal.WLSUserImpl "shawn"
 3. JpsPrincipal:
oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl
"authenticated-role" GUID=null DN=null
 4. JpsPrincipal:
oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl "anonymous-role"
GUID=null DN=null
 5. JpsPrincipal: oracle.security.jps.service.policystore.ApplicationRole
"appRoleManager" GUID=null DN=null)]
          Evaluator:            [ACC]
```

A permission check that fails generates an output with details illustrated in the following sample:

```
JpsAuth] Check Permission
          PolicyContext:        [jps-wls-Demo]
          Resource/Target:      [app.monitor]
          Action:               [read,write]
          Permission Class:     [java.util.PropertyPermission]
          Evaluator:            [ACC]
          Result:               [FAILED]
          Failed
ProtectionDomain:ClassLoader=weblogic.servlet.jsp.JspClassLoader@1b7682d finder:
weblogic.utils.classloaders.CodeGenClassFinder@7d32cf annotation:
CodeSource=file:/C:/Mydom/domains/domain/servers/AdminServer/jspservlet/test.class
 Principals=total 5 of principals(
 1. weblogic.security.principal.WLSUserImpl "duane"
 2. weblogic.security.principal.WLSGroupImpl "employee"
 3. JpsPrincipal: oracle.security.principals.JpsAuthenticatedRoleImpl
"authenticated-role" GUID=null DN=null
 4. JpsPrincipal: oracle.security.principals.JpsAnonymousRoleImpl "anonymous-role"
GUID=null DN=null
 5. JpsPrincipal: oracle.security.jps.service.policystore.ApplicationRole
"appRoleEmployee" GUID=null DN=null)
 Permissions=(
 (java.util.PropertyPermission line.separator read)
 ...
 (java.lang.RuntimePermission stopThread))
 Call Stack: java.security.AccessControlException: access denied
(java.util.PropertyPermission app.monitor read,write)
 java.security.AccessControlContext.checkPermission(AccessControlContext.java:323)
 ...
 weblogic.work.ExecuteThread.run(ExecuteThread.java:173)
          ProtectionDomains for class stack:
 Class[0]: class oracle.security.jps.util.JpsAuth$Diagnostic$SMSupport
 ProtectionDomain: ClassLoader=sun.misc.Launcher$AppClassLoader@360be0
 CodeSource=file:/C:/MyOracle/jdeveloper/modules/oracle.jps_12.1.3/jps-api.jar
 Principals=total 0 of principals<no principals>
 Permissions=(
 (java.io.FilePermission \C:\MyOracle\jdeveloper\modules\jps-api.jar read)
 ...
 )
 Class[1]: class oracle.security.jps.util.JpsAuth$Diagnostic$SMSupport
```

To disable permission check messages, set both `jps.auth.debug` and `jps.auth.debug.verbose` to false; by default, `jps.auth.debug.vebose` is set to false.

J.1.2.3 Debugging the Authorization Process

This section describes the use of several other system properties that help debugging the authorization process based on several criteria. Specifically, the following system properties:

```
oracle.security.jps.log.for.approle.substring
oracle.security.jps.log.for.permeffect
oracle.security.jps.log.for.permclassname
oracle.security.jps.log.for.permtarget.substring
oracle.security.jps.log.for.enterprise.principalname
```

generate logging messages during the following authorization phases:

- Phase 1 - The application roles that were granted to an enterprise user or to an enterprise role during the OPSS Subject computation.
- Phase 2 - The permission instances that were granted to a grantee.
- Phase 3 - The outcome of a permission check, that is, whether the grant was granted or denied.

Each of the above properties and the phases they apply are described next.

`oracle.security.jps.log.for.approle.substring` - During phases 1, 2, and 3, it logs the name of an application role that contains a specified substring; if the substring to match is unspecified, it logs all application role names.

`oracle.security.jps.log.for.permeffect` - During phase 3 and according to a specified value, it logs a grant that was granted or denied; if the value is unspecified, it logs all grants (regardless whether they were granted or denied).

`oracle.security.jps.log.for.permclassname` - During phases 2 and 3, it logs the name of the permission class that matches exactly a specified name; if the name to match is unspecified, it logs all permission class names.

`oracle.security.jps.log.for.permtarget.substring` - During phases 2 and 3, it logs the name of a permission target that contains a specified substring; if the substring to match is unspecified, it logs all permission targets.

`oracle.security.jps.log.for.enterprise.principalname` - During phases 1, 2, and 3, it logs the name of the principal (enterprise user or enterprise role) that matches exactly a specified name; if the name to match is unspecified, it logs all principal names.

The following characteristics apply to all of the above system properties:

- They are optional.
- They can be set at most once.
- The matchings (where they apply) are case insensitive

To enable the logging of any of the above system properties, proceed as follows:

1. Set the desired system properties.
2. Stop the JVM.
3. Restart the JVM.
4. Set the logger `oracle.security.jps.dbg.logger` to `TRACE:32`. For details on how to set a logger, see Managing Loggers with Fusion Middleware Control.
5. Run the scenario to be debugged.
6. Examine the log output; to locate the messages output by the settings of any of the above properties, search the log file for the key word *[oracle.security.jps.dbg.logger]*.

J.1.2.3.1 Examples of Use The following examples illustrate typical settings of the above system properties.

- To log all application role names that contain the substring `myAppRole`, include the following setting:

```
-Doracle.security.jps.log.for.approle.substring=myAppRole
```

- To log all denied permission checks, include the following setting:

  ```
  -Doracle.security.jps.log.for.permeffect=deny
  ```

- To log all granted permission checks, include the following setting:

  ```
  -Doracle.security.jps.log.for.permeffect=grant
  ```

- To log all granted or denied permission checks, do not set `oracle.security.jps.log.for.permeffect`.
- To log all permission checks that match exactly the class name `java.util.PropertyPermission`, include the following setting:

  ```
  -Doracle.security.jps.log.for.permclassname=java.util.PropertyPermission
  ```

- To log all target names that contain the substring `p.mon`, include the following setting:

  ```
  -Doracle.security.jps.log.for.permtarget.substring=p.mon
  ```

- To log all authorizations involving the principal name `manager`, include the following setting:

  ```
  -Doracle.security.jps.log.for.enterprise.principalname=manager
  ```

- To log application role names that match a substring or principal names that match a string, set both `oracle.security.jps.log.for.approle.substring` and `oracle.security.jps.log.for.enterprise.principalname` as indicated above.
- To log *all* application roles names and *all* principal names, set neither `oracle.security.jps.log.for.approle.substring` nor `oracle.security.jps.log.for.enterprise.principalname`.

J.1.3 Solving Security Errors

There is no generic way to resolve errors when they occur. One must search for hints and frequently follow multiple hypotheses until, hopefully, the source of the error is isolated and understood. To this end, this section describes how to search and interpret log information to resolve most common security errors. These topics are addressed in the following sections:

- Understanding Sample Log Entries
- Searching Logs with Fusion Middleware Control
- Identifying a Message Context with Fusion Middleware Control
- Generating Error Listing Files with Fusion Middleware Control

J.1.3.1 Understanding Sample Log Entries

Understanding log error output is crucial to isolate and solve an error. Let's take a closer look at a diagnostic log file to describe the information you find for an error logged in such a file. This description is best illustrated with a real-life example.

The following is an excerpt of an error in the file `AdminServer-diagnostic.log`:

```
[2009-01-07T09:15:02.393-08:00] [AdminServer] [ERROR] [JPS-00004]
[oracle.jps.admin]
[tid: [ACTIVE].ExecuteThread: '3' for queue: 'weblogic.kernel.Default
```

```
(self-tuning)'] [userId: weblogic] [ecid: 0000Hum5kxw7MAn54nU4Ui19PD8S000005,0]
Unable to add principal to the application role. Reason: Principal
"abc.xxx@myComp.com" is already a member of the application role
"BPMWorkflowAdmin"[[
java.security.PrivilegedActionException:
oracle.security.jps.service.policystore.PolicyObjectAlreadyExistsException:
Unable to add principal to the application role. Reason: Principal
"abc.xxx@myComp.com" is already a member of the application role
"BPMWorkflowAdmin"
        at java.security.AccessController.doPrivileged(Native Method)
        at oracle.security.jps.mas.mgmt.jmx.policy.JpsApplicationPolicyStoreImpl.
addRemoveMembersToRole(JpsApplicationPolicyStoreImpl.java:408)
        at oracle.security.jps.mas.mgmt.jmx.policy.JpsApplicationPolicyStoreImpl.
addMembersToApplicationRole(JpsApplicationPolicyStoreImpl.java:385)
        at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
…
```

The meaning of the fields in the preceding message is as follows:

- [2009-01-07T09:15:02.393-08:00]

  Identifies the date and time when the error was logged.

- [AdminServer]

  Identifies the name of the server where the error occurred.

- [JPS-00004]

  Identifies the error code and hints to the kind of error that occurred. For a complete list of JPS error codes, see chapter 41 in *Error Messages*.

- [oracle.jps.admin]

  Identifies the logger category. The subcategories of `oracle.jps` (such as `admin` above) indicate the kind of error that occurred. They include the following:

  - common - generic errors.
  - upgrade - upgrade errors.
  - config - configuration errors.
  - deployment - deployment errors.
  - authentication - login module errors in Java SE applications only.
  - idmgmt - identity store errors.
  - credstore - credential store errors.
  - authorization - policy store errors at run time.
  - policymgmt - policy store management errors.
  - admin - JMX and WLST errors.

- [tid: [ACTIVE].ExecuteThread: '3' for queue: 'weblogic.kernel.Default (self-tuning)']

  Identifies the thread where the error occurred.

- [userId: weblogic]

  Identifies the user that performed the operation that generated the error.

- [ecid: 0000Hum5kxw7MAn54nU4Ui19PD8S000005,0]

Identifies the execution context id. Typically used to correlate and trace sequence of events. Ecids provide information about the flow across processes, such as, from a request, to the WebLogic server, to an Oracle Internet Directory server.

- Unable to add principal to the application role. Reason: Principal abc.xxx@myComp.com is already a member of the application role BPMWorkflowAdmin

  Identifies the reason why the error was logged.

- java.security.PrivilegedActionException: oracle.security.jps.service.policystore.PolicyObjectAlreadyExistsException: Unable to add principal to the application role. Reason: Principal abc.xxx@myComp.com is already a member of the application role BPMWorkflowAdmin

  Identifies the exception that was raised and the reason for it.

J.1.3.2 Searching Logs with Fusion Middleware Control

To initiate a search in the contents of all log files in a domain, select *Domain* **> Logs > View Log Messages**, to display the **Log Messages** page.

In this page you have several parameters that you can choose from to specify your search query; specifically, you can:

- Choose a time interval in which a message was issue, by selecting the appropriate **Date Range**.
- Display messages with a given severity error, by checking any of the **Message Types** boxes.
- Display messages satisfying further constrains, by choosing an item from the menu **Message** and entering a string in the box next to it. For example, you could query for just messages that contain the string **exception** in it.
- Add extra query fields, by clicking the button **Add Fields** and checking any of the available choices. For example, you could add the field Host, and then enter the appropriate query, such as **starts with** a particular string.

Once these parameters are set, click **Search** and the result of the query is displayed in the page. The result of a query can be further redisplayed by message type, message ID, or simple list of messages, by selecting an item from the menu **Show**. Moreover, the result can be automatically refreshed by choosing an item from the menu at the top right of the page (by default set to Manual Refresh).

To broaden a search to log files beyond a domain, use the button **Broaden Target Scope** at the top right of the page.

J.1.3.3 Identifying a Message Context with Fusion Middleware Control

In some situations, it is necessary to know the context in which a message has occurred. For example, it may be useful to know messages that have preceded or followed a given error message by, say, 2 minutes.

The tab **View Related Messages** provides this functionality, and you can use it as follows:

1. Display the results of a query with **Show** set to **Messages**.
2. Select a message within the result table. Note that the tabs **View Related Messages** and **Export Messages to File** become then available. Let's assume, for example, that the selected message has the time stamp Jan 21, 2009 4:05:00 PM PST.

3. Select **Time Interval** from the **Date Range** menu, and enter a **Start Date** and an **End Date.** For example, you could enter Jan 21, 2009 4:02:00 PM, as a start date, and Jan 21, 2009 4:07:00 PM.
4. Select **by Time** from the menu **View Related Messages**, to display the page with all the messages related to the selected one in the specified time span.
5. In the **Related Messages by Time** page, you can modify the time window around the time of the selected message by choosing an item from the menu **Scope**, at the right of the page.

J.1.3.4 Generating Error Listing Files with Fusion Middleware Control

In some situations, you may want to download the list of errors displayed into a separate file to forward it, for example, to a support center, or just to keep it for your records.

Whenever available, the tab **Export Messages** allows you to generate a file containing just the displayed results by choosing an item from the menu. The format of the generated file can be plain text, XML, or CSV.

The following sample, showing only the first of 29 messages, is an excerpt of a text file generated this way:

```
#
#Search Criteria
#Start Time: 2009-01-21T16:34:41.381-08:00
#End Time:  2009-01-21T16:39:41.381-08:00
#Message Types: ERROR, WARNING

#Selected Targets List
#/Farm_base_domain/base_domain/AdminServer:Oracle WebLogic Server
#/Farm_base_domain/base_domain/AdminServer/DMS Application:Application Deployment
#/Farm_base_domain/base_domain/AdminServer/em:Application Deployment
#/Farm_base_domain/base_domain/AdminServer/wsil-wls:Application Deployment
#/Farm_base_domain/base_domain/AdminServer/wsm-pm:Application Deployment
#
[2009-01-21T16:34:54.045-08:00] [AdminServer] [WARNING] []
[org.apache.myfaces.trinidad.bean.PropertyKey] [host: stacz39] [nwaddr:
140.87.5.40] [tid: 13] [userId: <anonymous>] [ecid:
0000HvvkgjVE^MT6uBj8EH19TvXj000008,0] [APP: em] [Target: /Farm_base_domain/base_
domain/AdminServer/em] [Target Type: Application Deployment] Unserializable
value:oracle.sysman.core.view.tgtctls.common.DefaultTreeModel@1fcadd2 for
key:UINodePropertyKey[value,17]
…
#
#Number of messages exported: 29
#
```

J.2 The OPSS Diagnostic Framework

The OPSS diagnostics is a framework that helps support personnel reduce the resolution time of reported problems. Using this framework, one can extract internal states of a domain into a dump that may illuminate the cause of a particular problem.

The OPPS diagnostic framework provides a number of tests that can be invoked (using WLST) to generate a dump; these dumps record the characteristics of a domain servers and the settings of a particular OPSS service in that domain, and help locating what the issue may be.

The OPSS diagnostic framework provides the following tests:

- Configuration test
  - oracle.security.jps.diag.test.JpsConfigTest
- Connectivity tests
  - oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTest
  - oracle.security.jps.diag.test.JpsIdentityStoreUserRoleApiLdapConnectivityTest
- Credential store tests
  - oracle.security.jps.diag.test.JpsCredentialStoreConfigTest
  - oracle.security.jps.diag.test.JpsCredentialStoreTest
- Identity store tests
  - oracle.security.jps.diag.test.JpsIdentityStoreIDSSearchTest
  - oracle.security.jps.diag.test.JpsIdentityStoreLibOvdConfigTest
  - oracle.security.jps.diag.test.JpsIdentityStoreUserRoleApiSearchTest
- Key store tests
  - oracle.security.jps.diag.test.JpsKeyStoreTest
  - oracle.security.jps.diag.test.JpsKeyStoreConfigTest

Running a Test

The following line describes the syntax of the command `executeDump`, which is used to invoke any of the above tests:

```
executeDump(name='opss.diagTest', outputFile='dumpLocation',
args={'testtname':'testName'})
```

Note the following points:

- The first argument, `name='opss.diagTest'`, is fixed.
- `outputFile` specifies the location where the dump is generated, relative to where the command is run.
- `testName` stands for one of the available tests; the other strings in the third argument are fixed and must be specified as described the syntax above. If the wildcard character '*' is passed as a `testName`, then all the tests are run in the following order:

  ```
  oracle.security.jps.diag.test.JpsConfigTest
  oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTest
  oracle.security.jps.diag.test.JpsCredentialStoreConfigTest
  oracle.security.jps.diag.test.JpsCredentialStoreTest
  oracle.security.jps.diag.test.JpsIdentityStoreIDSSearchTest
  oracle.security.jps.diag.test.JpsIdentityStoreLibOvdConfigTest
  oracle.security.jps.diag.test.JpsIdentityStoreUserRoleApiLdapConnectivityTest
  oracle.security.jps.diag.test.JpsIdentityStoreUserRoleApiSearchTest
  oracle.security.jps.diag.test.JpsKeyStoreConfigTest
  oracle.security.jps.diag.test.JpsKeyStoreTest
  ```

The following sequence illustrates the use of the diagnose framework that is typically conducted by a support person:

1. A problem is received from a customer.
2. From the logged failure, it is determined that the problem has to do with, for example, a connection.

3. One or more relevant tests in the domain in question are run; for example, the following test:

   - oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTest

   To run this test, proceeds as follows:

   1. Invoke the off line WLST:

      ```
      $ wlst.sh
      ```

   2. Connect to the WebLogic server, as illustrated in the following line:

      ```
      wls:/offline> connect('adminuser', 'adminpass', 't3://localhost:7001')
      ```

   3. Invoke the test, as illustrated in the following line:

      ```
      >executeDump(name='opss.diagTest', outputFile='myDumpTest',
      args={'testname':
      'oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTe
      st'})
      ```

4. The test generates a dump at the specified location; a snippet of the dump generated by the above test follows:

   ```
   ...
   LDAP authenticator : OID
   host : myHost.com
   port : 7066
   principal : cn=orcladmin
   password : ********
   Is SSL? false
   Testing LDAP connection to ldap://myHost.com:7066 with principal cn=orcladmin.
   JNDI settings:
   java.naming.provider.url = ldap://myHost.com:7066
   java.naming.factory.initial = com.sun.jndi.ldap.LdapCtxFactory
   com.sun.jndi.ldap.connect.timeout = 5000
   java.naming.security.principal = cn=orcladmin
   java.naming.security.authentication = simple
   java.naming.security.credentials = ********
   Failed to establish LDAP connection to ldap://myHost.com:7066.
   The stack trace:
   javax.naming.CommunicationException: myHost.com:7066 [Root exception is
   java.net.ConnectException: Connection refused]
   at com.sun.jndi.ldap.Connection.<init>(Connection.java:214) …
   ```

5. Inspection of the dump possibly helps to identify the cause of the issue. In the sample dump above, for example, the text in bold indicates that the connection to the LDAP authenticator could not be established.

J.3 Troubleshooting Reassociation and Migration

This section describes the following issues:

- Reassociation Failure
- Migration Failure
- Unsupported Schema
- Missing Policies in Reassociated Policy Store
- Migration Failure

J.3.1 Reassociation Failure

Policy and credential reassociation from a file-based store to an OID store may fail for several reasons. This section explains three reasons why this operation may fail.

Symptom 1- Error Code 32

Reassociation fails and an error like the following is logged in the administration server diagnostic file *serverName*.`diagnostic.log`:

```
[LDAP: error code 32 - No Such Object]
Authentication to LDAP server ldap://myServer.com:3060 is unsuccessful.
```

Diagnosis 1

The error above identifies a problem with the target node in the OID server, namely, that the node specified does not exist.

It is required that the root node specified in the text box **JPS Root DN** (of the page **Set Security Provider**) be present in the OID directory *before* invoking the reassociation.

Solution 1

Verify that the data you enter in the box **JPS Root DN** matches the name of a node in the target OID directory, and then rerun the reassociation.

Symptom 2- Error Code 68

Reassociation fails and an error like the following is logged in the administration server diagnostic file *serverName*.`diagnostic.log`:

```
Authentication to LDAP server ldap://myServer.com:3060 is successful.
Starting to migrate policy store...
Set up security provider reassociation successfully.
Checked and seeded security store schema successfully.
null
[LDAP: error code 68 - Object already
exists]:cn=SystemPolicy,cn=domain1,cn=JPSContext,cn=nb_policy
Error occurred while migrating LDAP based policy store.
```

Diagnosis 2

The error above indicates that the name specified in the box **WebLogic Domain Name** is a descendant (more precisely, a grandchild) of the **JPS Root DN** node in the target OID directory.

It is required that the domain specified do *not* be a descendant of the root node.

Solution 2

Verify that the name you enter in the box **WebLogic Domain Name** does not match the name of a grandchild of the specified **JPS Root DN** node, and rerun the reassociation.

Symptom 3

Reassociation, carried out with Fusion Middleware Control, fails and an error like the following is logged in the administration server diagnostic file *serverName*.`diagnostic.log`:

```
[2009-01-21T10:09:24.326-08:00] [AdminServer] [ERROR] [] [oracle.jps.admin] [tid
: [ACTIVE].ExecuteThread: '15' for queue: 'weblogic.kernel.Default (self-tuning)
'] [userId: weblogic] [ecid: 0000HvuOTpe7q2T6uBADUH19Tpyb000006,0] Unable to rem
ove the principal from the application role. Reason: Principal "Managers" is not
```

```
a member of the application role "test-role"[[
java.security.PrivilegedActionException: oracle.security.jps.service.policystore
.PolicyObjectNotFoundException: Unable to remove the principal from the applicat
ion role. Reason: Principal "Managers" is not a member of the application role "
test-role"
       at oracle.security.jps.mas.mgmt.jmx.policy.JpsApplicationPolicyStoreImpl
.addRemoveMembersToRole(JpsApplicationPolicyStoreImpl.java:408)...
```

Diagnosis 3

The error above points to some problem with the application role `test-role`, which is, in this case, the root of the problem.

Ensure that when entering data to perform reassociation with Fusion Middleware Control, you use the button **Test LDAP Authentication** immediately after you have completed entering all required values to connect to the target OID server. This test catches any problems with those values before reassociation begins.

Solution 3

In our example, a quick inspection of the file system-jazn-data.xml reveals that the application test-role is used by an application policy, but it was not defined. Here is an excerpt of that file illustrating where the required data is missing:

```
<application>
    <name>myApp</name>
        <app-roles>
 <--! test-role should have been defined here -->
        </app-roles>
        <jazn-policy>
            <grant>
                <grantee>
                    <principals>
                       <principal>
                         <class>
oracle.security.jps.service.policystore.ApplicationRole</class>
                         <name>test-role</name>
                         <guid>66368900E7E511DD9F62F9ADA4233FE2</guid>
                       </principal>
                     </principals>...
```

To solve this particular error, (a) fix `system-jazn-data.xml` by inserting the definition of the application test-role; (b) revert to file-based domain stores with the fixed file; and (c) rerun the reassociation.

Symptom 4 - Audit Store is Not Reassociated

In release 11.1.1.6.0, if you reassociated security stores through the Fusion Middleware Control Enterprise Manager (EM) console, most stores (policy store, credential store and so on) moved except for the audit store. This is because the audit store did not support reassociation through the console, only through the WLST command `reassociateSecurityStore`.

Diagnosis 4

In a situation where the original migration from release 11.1.1.6 to release 12.1.3 was done through Enterprise Manager, this leaves the audit repository as file-based. You can use the following resolution to move all security store data to OID/DB in order to enable audit.

Solution 4

In the original environment, run WLST command `reassociateSecurityStore` with a different `jpsroot` node. This effects an OID-to-OID directory reassociation and any existing data also gets migrated to the new node. After you take this action, audit data will no longer be file based and `jps-config` will have the new node.

J.3.2 Unsupported Schema

This section explains a reason why reassociation to an OID server may fail.

Symptom

Reassociating the security store to an OID repository fails and the AdminServer log reports an error like the following:

```
[2011-02-09T07:01:13.884-05:00] [AdminServer] [ERROR] [] [oracle.jps.admin] [tid:
[ACTIVE].ExecuteThread: '6' for queue: 'weblogic.kernel.Default (self-tuning)']
[userId: weblogic] [ecid:
41050d66ef2ec40b:-4c1fb689:12e06cc7b6c:-8000-00000000000001e1,0] Schema seeding
failed, check the server type of the given ldap url.[[
oracle.security.jps.JpsException: Error Modifying JPS Schema,  Record: dn:
cn=schema
changetype: modify
delete: objectclasses
objectclasses: ( 2.16.840.1.113894.7.2.2 NAME 'orclContainer' SUP ( top ) MUS
 T ( cn ) MAY ( orclVersion $ orclServiceType ) )
-
: [LDAP: error code 32 - No Such Object]:cn=schema
```

Diagnosis

The error `LDAP: error code 32` indicates that the schema of the reassociation target OID repository is not supported, that is, the version of the target LDAP repository is not one of the OPSS supported stores.

Solution

Update the target OID repository to one supported and then try reassociating again. The version of an OID store must be 10.1.4.3 or later. For a list of supported versions, see Section 9.2, "Using an LDAP-Based OPSS Security Store."

J.3.3 Missing Policies in Reassociated Policy Store

Symptom

When a file-based policy store is reassociated to use an Oracle Internet Directory policy store, the reassociation reports that it completed successfully.

At runtime, however, the system does not behave as expected. Codebase policies, that are supposed to be present in the system policy after migration, are missing.

Diagnosis

At runtime, the server reports a stack trace that resembles the following:

```
<BEA-000000> <JspServlet: initialization complete>
####<May 4, 2009 8:32:50 AM PDT> <Error> <HTTP> <ap626atg> <WLS_Spaces>
<[ACTIVE] ExecuteThread: '3' for queue: 'weblogic.kernel.Default
(self-tuning)'> <<WLS Kernel>> <> <> <1241451170341> <BEA-101020>
<[ServletContext@20193148[app:webcenter module:/webcenter path:/webcenter
spec-version:2.5]] Servlet failed with Exception
```

```
java.security.AccessControlException: access denied
(oracle.security.jps.service.policystore.PolicyStoreAccessPermission
context=APPLICATION,name=webcenter getApplicationPolicy)
        at
java.security.AccessControlContext.checkPermission(AccessControlContext.java:323)
        at
java.security.AccessController.checkPermission(AccessController.java:546)
        at
oracle.security.jps.util.JpsAuth$AuthorizationMechanism$3.checkPermission(JpsAuth.
java:348)
        at
oracle.security.jps.util.JpsAuth$Diagnostic.checkPermission(JpsAuth.java:268)
        at
oracle.security.jps.util.JpsAuth$AuthorizationMechanism$6.checkPermission(JpsAuth.
java:372)
        at oracle.security.jps.util.JpsAuth.checkPermission(JpsAuth.java:408)
        at oracle.security.jps.util.JpsAuth.checkPermission(JpsAuth.java:431)
        at
oracle.security.jps.internal.policystore.AbstractPolicyStore.checkPolicyStoreAcces
sPermission(AbstractPolicyStore.java:246)
        at
oracle.security.jps.internal.policystore.ldap.LdapPolicyStore.getApplicationPolicy
(LdapPolicyStore.java:281)
        at
oracle.security.jps.internal.policystore.PolicyUtil.getGrantedAppRoles(PolicyUtil.
java:898)
        at
oracle.security.jps.internal.policystore.PolicyUtil.getJpsAppRoles(PolicyUtil.java
:1354)
        at
oracle.security.jps.wls.JpsWlsSubjectResolver$1.run(JpsWlsSubjectResolver.java:273
)
        at
oracle.security.jps.wls.JpsWlsSubjectResolver$1.run(JpsWlsSubjectResolver.java:270
)
        at java.security.AccessController.doPrivileged(Native Method)
```

Here the permission:

```
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
context=APPLICATION,name=webcenter getApplicationPolicy
```

is granted to a code base, and the authorization is not allowed since it evaluates to `false`.

Solution

Check the AdminServer diagnostic logs for messages like the following:

```
AdminServer-diagnostic.log:[2009-05-28T02:27:52.249-07:00] [AdminServer]
[NOTIFICATION] [JPS-00072] [oracle.jps.config] [tid: Thread-39] [ecid:
0000I66Z0KH0fplp4sm3Ui1A7_Rl00002s,1:5001] [arg: 11.1.1.1.0] [arg: 11.1.1.0.0]
Policy schema upgrade not required. Store Schema version 11.1.1.1.0 is compatible
to the seed schema version 11.1.1.0.0
AdminServer-diagnostic.log:[2009-05-28T02:28:58.012-07:00] [AdminServer]
[NOTIFICATION] [JPS-00078] [oracle.jps.config] [tid: Thread-39] [ecid:
0000I66Z0KH0fplp4sm3Ui1A7_Rl00002s,1:5001] [arg: 11.1.1.1.0] [arg: 11.1.1.0.0]
Credential store schema upgrade not required. Store Schema version 11.1.1.1.0 is
compatible to the seed schema version 11.1.1.0.0
```

A message of this type suggests that the schema was never seeded during the re-association. If the correct schema is not seeded in the Oracle Internet Directory server, the system will not work as expected.

To ensure that the schema is seeded during re-association, proceed as follows:

1. Remove the `cn=OPSS` container under the `cn=OracleSchemaVersion` container in the Oracle Internet Directory server.
2. Start with a clean working instance of an OPSS policy store using the file-based store.
3. Re-associate this file-based store to the Oracle Internet Directory server.

Check the AdminServer diagnostic logs to confirm that the OPSS schema was seeded in the OID server by looking for this message:

```
AdminServer-diagnostic.log:[2009-05-29T07:18:18.002-07:00] [AdminServer]
[NOTIFICATION] [JPS-00078] [oracle.jps.config] [tid: Thread-12] [ecid:
0000I61Z0MH0fplp4sm3Ui1A7_Ll00002s,1:5001] [arg: 11.1.1.0.0]  Policy schema
version set to 11.1.1.0.0
```

If re-associating to a Release 11*g* Oracle Internet Directory server, the schema version should read: 11.1.1.1.0

If re-associating to a Release 10.1.4.3 Oracle Internet Directory server, the schema version should read: 11.1.1.0.0

The Policy Store schema version is set in the Oracle Internet Directory server under this container:

```
cn=PolicyStore,cn=OPSS,cn=OracleSchemaVersion
```

Similarly, the Credential Store schema version is set in the Oracle Internet Directory server under this container:

```
cn=CredentialStore,cn=OPSS,cn=OracleSchemaVersion
```

J.3.4 Migration Failure

This section describes a reason why the automatic migration of policies at application deployment may fail. Note that the deployment of an application may succeed even though the migration of policies failed.

> **Note:** The reason why the automatic migration can fail, as explained in this section, can also lead to similar failures when reassociating domain stores.

For a failure also related to migration, see Incompatible Versions of Policy Stores.

Symptom

The application is configured to migrate policies automatically at deployment. The application deployment succeeds, but the diagnostic file corresponding to the server where it has been deployed outputs a message like the following:

```
[2009-01-21T13:34:48.144-08:00] [server_soa] [NOTIFICATION] []
[oracle.jps.deployment] [tid: [ACTIVE].ExecuteThread: '2' for queue:
'weblogic.kernel.Default (self-tuning)'] [userId: weblogic]
[ecid: 0000Hvv7U_H7q2T6uBADUH19Tq0B00002I,0] [APP: JpsJdev#V2.0]
Application [JpsJdev#V2.0] is being deployed, start policy migration.
```

```
[2009-01-21T13:34:48.770-08:00] [server_soa] [WARNING] []
[oracle.jps.deployment] [tid: [ACTIVE].ExecuteThread: '2' for queue:
'weblogic.kernel.Default (self-tuning)'] [userId: weblogic]
[ecid: 0000Hvv7U_H7q2T6uBADUH19Tq0B00002I,0] [APP: JpsJdev#V2.0]
Exception in application policy migration.[[
oracle.security.jps.JpsException: appplication Role:
test_role not found for the application in the destination policy store
at oracle.utility.destination.apibased.JpsDstPolicy.convertAppPolicyPrincipal
(JpsDstPolicy.java:815)
at oracle.utility.destination.apibased.JpsDstPolicy.clone
(JpsDstPolicy.java:691)...
```

The above excerpt was extracted from the file `server_soa-diagnostic.log`, and the application `JpsJdev` was deployed to the managed server `server_soa`. Note that the key phrase to look for to locate such error is highlighted in the sample above. In addition, the error describes the artifact that raised the exception, the application role `test_role`.

Diagnosis

Something is wrong with the definition of this role in the application file `jazn-data.xml`. In fact, a quick look at this file reveals that the role `test_role` is referenced in a grantee, as illustrated in the following excerpt:

```
<grantee>
   <display-name>myPolicy</display-name>
   <principals>
     <principal>
       <class>oracle.security.jps.service.policystore.ApplicationRole</class>
       <name>test_role</name>
     </principal>
   </principals>
</grantee> ...
```

But the name of what is supposed to be the application role named `test_role`, however, was inadvertently misspelled to `test_rolle`:

```
<application>
   <name>JpsJdev</name>
   <app-roles>
     <app-role>
       <name>test_rolle</name>
       <class>oracle.security.jps.service.policystore.ApplicationRole</class>
        <members> ...
```

Solution

Ensure that all application roles referenced in application policies have been properly defined in the `jazn-data.xml` file. If a referenced role name cannot be matched, as in the samples above, the migration fails.

J.4 Troubleshooting Server Starting

This section explains several reasons why the Oracle WebLogic Server may fail to start in the following sections:

- Missing Required LDAP Authenticator
- Missing Administrator Account

- Missing Permission
- Server Fails to Start
- Other Server Start Issues
- Permission Failure Before Server Starts

J.4.1 Missing Required LDAP Authenticator

This section explains a reason why the Oracle WebLogic Server may fail to start after modifying the list of authenticators in a domain.

Symptom

After modifying the list of authenticator providers in a domain, the Oracle WebLogic Server fails to start, and the error messages output include the following:

```
java.lang.IllegalArgumentException: null KeyStore name
```

Diagnosis

One cause of this problem is that the list of authenticators in your domain does not include an LDAP authenticator.

Important: An LDAP authenticator is *required* in this list for any domain using OPSS.

Solution

Since the server cannot start and an LDAP authenticator is missing, you must add it manually, as follows:

1. Open the file `DOMAIN_NAME/config/config.xml`.
2. Edit `config.xml` and include, within the element `<realm>`, an LDAP authenticator, such as the default authenticator illustrated in the following sample:

```
<realm>
 ...
 <sec:authentication-provider xsi:type="wls:default-authenticatorType">
 </sec:authentication-provider>
 ...
</realm>
```

3. Restart the server.

Once the server is back up and running, you can modify the list of providers to include the provider of your choice using the WebLogic Administration Console, but ensure that at least one of them is an LDAP authenticator provider.

To this end, use the WebLogic Administration Console as follows:

1. Navigate to the page **Create a new Authenticator Provider**.
2. Enter the authenticator name and select an authenticator type, all of which are LDAP-based:
   - ActiveDirectoryAuthenticator
   - DefaultAuthenticator (this is the one inserted manually in the sample above)
   - LDAPAuthenticator

- LDAPX509IdentityAsserter
- OpenLDAPAuthenticator
- OracleInternetDirectoryAuthenticator
- OracleVirtualDirectoryAuthenticator

J.4.2 Missing Administrator Account

This section explains a reason why the Oracle WebLogic Server may fail to start.

Symptom

After removing the out-of-box default authenticator and adding, say an Oracle Internet Directory authenticator, the server fails to start.

Diagnosis

Most likely, you have forgotten to enter an account member of the Administrators group in your added authenticator. The server requires that such an account be present in one domain authenticator. This account is always present in the default authenticator.

Solution

Since the server cannot start, you must add the deleted one LDAP authenticator manually, as follows:

1. Open the file `DOMAIN_NAME/config/config.xml`.
2. Edit `config.xml` and include, within the element `<realm>`, the default authenticator, as illustrated in the following sample:

```
<realm>
 ...
 <sec:authentication-provider xsi:type="wls:default-authenticatorType">
 </sec:authentication-provider>
 ...
</realm>
```

3. Restart the server.

Once the server is back up and running, proceed as follows:

1. Create an account that is member of the Administrators group in the OID authenticator.
2. Set the Oracle Internet Directory authenticator flag to SUFFICIENT.
3. Restart the server, which it should start without problems, since it is using the account in the Administrators group provided in the default authenticator.
4. Reset the Oracle Internet Directory authenticator flag to REQUIRED and remove the default authenticator. The server should now start using the account in the Administrators group that you created in the Oracle Internet Directory authenticator.

J.4.3 Missing Permission

This section explains a reason why the Oracle WebLogic Server may fail to start.

Symptom

The server fails to start when it started with security manager is enabled (with the system property `-Djava.security.manager`).

Diagnosis

One reason why you may run into this issue is the lack of permission grants to PKI APIs in `oraclepki.jar` when the security manager is enabled at server startup.

Solution

Ensure that a grant like the following is present in the file `weblogic.policy`, or add it if it is not:

```
grant codeBase "file:${oracle.home}/modules/oracle.pki_${opss.version}/*" {
 permission java.security.AllPermission;
};
```

The above grant is provided by default. Note that when security manager is enabled, the access to all system resources requires codebase permission grants.

For complete details about using the Java Security Manager to protect WebLogic resources, see *Developing Applications with the WebLogic Security Service*.

> **Note:** Printing Security Manager is a WebLogic server enhancement to the Java Security Manager. Use Printing Security Manager to identify all of the required permissions for a Java application running under Java Security Manager. Unlike the Java Security Manager, which identifies needed permissions one at a time, the Printing Security Manager identifies *all* the needed permissions without intervention.

J.4.4 Server Fails to Start

This section explains two reasons why the Oracle WebLogic Server will fail to start.

Symptom 1

The domain directory `${domain.home}/config/fmwconfig` is on an NFS-mounted partition, and an error message like the following is logged when the server is started:

```
JPS-01050: Opening of wallet based credential store failed. Reason
java.io.IOException: PKI-02002: Unable to open the wallet. Check password.
```

Furthermore, when `orapki` debugging is turned on and the server is started once again, the following message is logged:

```
java.io.IOException: No locks available.
```

> **Note:** To enable `orapki` debugging, start the server with the following property set: `-Doracle.pki.debug=true`.

Diagnosis 1

Since OPSS requires file locking when managing security artifacts in file-based stores, the error message `No locks available` indicates that the file system on which the domain directory is NFS-mounted does not support file locking.

Solution 1

Perform either of the following and restart the server:

- Upgrade from NFS v3 to NFS v4.
- Mount the remote file system with the `nolock` option enabled.
- Move files in `${domain.home}/config/fmwconfig` to a local storage

Symptom 2

The server fails to start because a wallet operation run into an exception. Furthermore, when `orapki` debugging is turned on (see note above) and the server is started once again, a file permission error is logged.

Diagnosis 2

Wallet operations create and make use of temporary files in the folder `/tmp`; all files matching the pattern `/tmp/*pki*` must be owned by the user that started the server. A file permission error message indicates that some files matching that pattern are not owned by the appropriate user.

Solution 2

Remove any files matching the pattern `/tmp/*pki*` that are *not* owned by the user that is starting the server and restart the server.

Symptom 3

This symptom is same as symptom 2 above, but it has a different resolution. The server fails to start because a wallet operation run into an exception. Furthermore, when orapki debugging is turned on (see note above) and the server is started once again, a file permission error is logged.

Diagnosis 3

The server must be started by the same OS user as the one who *installed the domain*. The server will fail to start even if started by any other member of the OS group to which the installer belongs.

Solution 3

Restart the server using the OS user who installed the domain.

J.4.5 Other Server Start Issues

This section explains several reasons why the Oracle WebLogic Server may fail to start.

Symptom

When attempting to load and set the policy provider, the Oracle WebLogic Server fails to start and logs an exception similar to the one illustrated in the following snippet:

```
<Mar 30, 2010 3:15:54 PM EDT> <Error> <Security> <BEA-090892> <The dynamic loading
of the OPSS java security policy provider class
oracle.security.jps.internal.policystore.JavaPolicyProvider failed due to problem
inside OPSS java security policy provider. Exception was thrown when loading or
setting the JPSS policy provider.
...
<Mar 30, 2010 3:15:54 PM EDT> <Critical> <WebLogicServer> <BEA-000386> <Server
subsystem failed. Reason: weblogic.security.SecurityInitializationException: The
dynamic loading of the OPSS java security policy provider class
oracle.security.jps.internal.policystore.JavaPolicyProvider failed due to problem
```

```
inside OPSS java security policy provider. Exception was thrown when loading or
setting the JPSS policy provider.
...
weblogic.security.SecurityInitializationException: The dynamic loading of the OPSS
java security policy provider class
oracle.security.jps.internal.policystore.JavaPolicyProvider failed due to problem
inside OPSS java security policy provider. Exception was thrown when loading or
setting the JPSS policy provider.
...
```

Diagnosis

The server startup includes loading and setting the policy provider as defined in the configuration file `jps-config.xml`; if this task is not completed successfully, the Oracle WebLogic Server fails to start. As illustrated in the sample above, this type of failure is identified in the server's log by the string

```
Exception was thrown when loading or setting the JPSS policy provider.
```

To determine the root cause of a particular failure server startup, check the server's log file and inspect the logged stack trace. For details about identifying errors, see Diagnosing Security Errors.

Here are some reasons why the server fails to start:

1. The path to the configuration file is incorrectly specified.
2. The default context is missing in the configuration file.
3. The XML parser is not available.
4. A code source URL is incorrectly specified in a system policy. This situation is identified by a logged exception that includes the string

   ```
   java.net.MalformedURLException: unknown protocol.
   ```

Solution

A solution for each of the above cases above is explained next.

1. Ensure that the correct path is specified by the system parameter oracle.security.jps.config:

   ```
   -Doracle.security.jps.config=<full-path-to-jps-config.xml>
   ```

   Note that special characters (such as backlashes or white space characters) in the full path specification must be properly escaped. One way to verify correctness is to test using the specified full path in a command line.
2. The configuration file must include a default context. For an example of a default context configuration, see <jpsContext>.
3. Make sure that the XML parser is available in your system and that the XML parser JAR file is included in the classpath.
4. Typical incorrect and corrected code source URLs are illustrated in the following two samples.

   ```
   Sample 1 - Incorrect URL
   <grantee>
     <codesource>
        <url>${my.oracle.home}/jlib/webcacheua.jar</url>
      </codesource>
   </grantee>
   ```

```
Sample 1 - Corrected URL (in bold)
<grantee>
  <codesource>
     <url>file:/${my.oracle.home}/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

Sample 2 - Incorrect URL
<grantee>
  <codesource>
     <url>c:/myfolder/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

Sample 2 - The corrected URL (in bold) is either one of the following three:
<grantee>
  <codesource>
     <url>file:///c:/myfolder/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

<grantee>
  <codesource>
     <url>file:c:/myfolder/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

<grantee>
  <codesource>
     <url>file:/c:/myfolder/jlib/webcacheua.jar</url>
   </codesource>
</grantee>
```

For details about the syntax of URL specifications in a code source (including the use of system variables), see <url>.

J.4.6 Permission Failure Before Server Starts

This section describes a reason why a permission check may fail before the server has completed its starting phase.

Symptom

An authorization check fails before the server has started. The server has completed its starpup when it outputs the a line like the following:

```
<WebLogicServer> <BEA-000365> <Server state changed to STARTING>
```

Diagnosis

A permission check error before the server has changed status to `STARTING` usually indicates that the authorization service required to check that permission was not fully initialized at the time of the request.

Solution

To workaround this issue, proceed as follows:

1. Edit the file `weblogic.policy` to add the appropriate grant(s).
2. Start the Oracle WebLogic Server with the following two system properties set:

- `java.security.policy` set to the location of the `weblogic.policy` file.
- `jps.policystore.hybrid.mode` set to true.

J.5 Troubleshooting Permissions

This section describes the following issues:

- Troubleshooting System Policy Failures
- Failure to Grant or Revoke Permissions - Case Mismatch
- Authorization Check Failure
- User Gets Unexpected Permissions
- Granting Permissions in Java SE Applications
- Application Policies Not Seen in 12c HA Environment

J.5.1 Troubleshooting System Policy Failures

A system policy is a policy that specifies a set of permissions that a principal or a codesource is allowed to perform, and it holds for an entire domain. Codesource grants are required, for example, when an application code needs to perform a management operation on a policy, a credential, a key, or audit.

Typically, a run-time authorization failure on any of the above operations manifests as a `java.security.AccessControlException`. This section describes the process that one would follow to diagnose and fix such exception.

Symptom

A runtime authorization failure occurs, as illustrated by the following error:

```
java.security.AccessControlException: access denied
(oracle.security.jps.service.credstore.CredentialAccessPermission
context=SYSTEM,mapName=oracle.patching,keyName=FUSION_APPS_PATCH_WLS_ADMIN-KEY
read)
```

A first step is to get detailed information about the failure by first enabling the following loggers and then reproducing the failure:

```
setLogLevel(target="serverName", logger="oracle.security.jps.util.JpsAuth",
level="TRACE:32", persist=1);

setLogLevel(target="serverName", logger="oracle.security.jps.trace.logger",
level="TRACE:32", persist=1);

setLogLevel(target="serverName", logger="oracle.security.jps.dbg.logger",
level="TRACE:32", persist=1);

setLogLevel(target="serverName",
logger="oracle.security.jps.internal.policystore.JavaPolicyProvider",
level="TRACE:32", persist=1);

setLogLevel(target="serverName", logger="oracle.jps.common", level="TRACE:32",
persist=1);
```

In the `JpsAuth` logger's output, look for the string "Failed ProtectionDomain"; the following sample output of the that logger illustrates the relevant lines around that string:

```
Failed ProtectionDomain:ClassLoader=sun.misc.Launcher$AppClassLoader@1823ab20
CodeSource=file:/scratch/idmprov/idmtop/products/iam/patch_wls1036/patch_
jars/BUG14331527_1036.jar
Principals=total 0 of principals<no principals>
Permissions=((java.io.FilePermission /scratch/idmprov/idmtop/products/iam/patch_
wls1036/patch_jars/BUG14331527_1036.jar read)
…
Call Stack:    java.security.AccessControlException: access denied
(oracle.security.jps.service.credstore.CredentialAccessPermission context=SYSTEM,
mapName=OAM_STORE, keyName=jks write)
e]
java.security.AccessControlContext.checkPermission(AccessControlContext.java:374)
e] java.security.AccessController.checkPermission(AccessController.java:546)
e] oracle.security.jps.util.JpsAuth$AuthorizationMechanism$3.checkPermission
(JpsAuth.java:463)
```

Diagnosis

The above failure indicates a mismatch between the provisioned codesource grant and the run-time expanded evaluation of the grant.

Solution

To resolve this issue, one must update the provisioned codesource grant so that it matches the permission, target, and action(s) that the run-time evaluates to. To this end, proceed as follows:

1. Inspect the provisioned codesource grant using either Fusion Middleware Control as explained in Section 10.3, "Managing Policies with Fusion Middleware Control," or the WSLT online command `listCodeSourcePermissions`. For details about this command, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*.
2. If the provisioned codesource grant does not match the data in the runtime error, modify the grant using Fusion Middleware Control or the WSLT online command `grantPermission`. For details about this command, see *Oracle Fusion Middleware Infrastructure Security WLST Command Reference*..
3. Ensure that domain and other variables expand to correctly specified the grant. Note that, in loggers, a codesource is output with an absolute path to the JAR file, but the URL is specified relative to environment variables.
4. If the grant was seeded with the application (that is, specified in the file `system-jazn-data.xml`) then review the grant in that file so that the fix will take effect when the application is deployed in a new environment.
5. If your code is required to run in a privileged block, make sure that it is doing so.

J.5.2 Failure to Grant or Revoke Permissions - Case Mismatch

This section explains the likely reasons why an enterprise user or role (group) may fail to be granted or revoked permissions.

Symptom

An enterprise user or group, properly entered in a domain authenticator, is not granted or revoked the permissions defined by a grant.

Diagnosis

This problem is likely to occur when there is a case mismatch between the stored name (in a domain authenticator) and the supplied name (either actively entered by a user or obtained programmatically). For example, this mismatch would occur when the stored user name is *JdOE* and the supplied user name is *jdoe*.

Solution

There are two ways to resolve this issue.

The first solution involves setting the appropriate property in the authenticator being used in your domain. As long as both strings (the supplied and the stored) contain identical sequence of characters (irrespective of case), this setting guarantees that the user name populated in the Subject matches the user name present in a domain authenticator, even when the corresponding characters differ in case. Thus, when this setting is in place, the user names *JdOE* and *jdoe* match.

To set your domain authenticator property, proceed as follows:

1. Use the Administration Console to navigate to the page where your authenticator is configured. For example, if you are using the default authenticator, navigate to the DefaultAuthenticator page by choosing **Realms > myrealm > Providers > DefaultAuthenticator.**
2. Choose the tab **Provider Specific**.
3. Set the property **userRetrievedUserNameAsPrincipal** to true.
4. Restart the server.

The second solution considers the case where the supplied name is obtained programmatically, that is, where one must produce a principal from a user name.

To obtain the correct user or group name, either pass the name *exactly* as it is stored in the authenticator or use the sequence of calls illustrated in the following code snippet:

```
import weblogic.security.principal.WLSGroupImpl;
import weblogic.security.principal.WLSUserImpl;

// Set the context
JpsContextFactory ctxFact = JpsContextFactory.getContextFactory();
ServerContextFactory scf = (ServerContextFactory) ctxFact;
JpsContext ctx = scf.getContext(ServerContextFactory.Scope.SYSTEM);
ctx = ctxFact.getContext();

// Set the identity store
IdentityStore identityStore =
ctx.getServiceInstance(IdentityStoreService.class).getIdmStore();

// In case of a user name, search the user that matches the supplied name
User user = idStore.searchUser(IdentityStore.SEARCH_BY_NAME, suppliedUserName);

// Use the obtained object (user) to obtain the stored user name and create
// the Principal
String storedUserName = user.getName();
Principal userPrincipal = new WLSUserImpl(storedUserName);

// Similarily, in case of a role name, search the role that matches
// the supplied role name
Role role = identityStore.searchRole(IdentityStore.SEARCH_BY_NAME,
suppliedRoleName);

// Use the obtained object (role) to obtain the stored role name and create
// the Principal
String storedRoleName = role.getName();
Principal rolePrincipal = new WLSGroupImpl(storedRoleName);
```

Important: When creating a user or role principal, you must use the calls:

```
Principal userPrincipal = new
WLSUserImpl(user.getUserProfile()getName());
Principal rolePrincipal = new
WLSGroupImpl(role.getRoleProfile().getName());
```

Instead of the calls:

```
Principal userPrincipal = new WLSUserImpl(user.getName());
Principal rolePrincipal = new WLSGroupImpl(role.getName());
```

J.5.3 Authorization Check Failure

This section explains a reason why an authorization check has failed.

Symptom

An attempt to authorize a user by your application fails, and the system logs an error containing a line like the following:

```
Servlet failed with Exception
oracle.adf.controller.security.AuthorizationException:ADFC-0619:
Authorization check failed: '/StartHere.jspx' 'VIEW'.
```

Diagnosis

One reason that can lead to such an authorization failure is a mismatch between the run-time policy context and the policy store stripe that you application is using.

On the one hand, the application stripe (or subset of policies in the policy store) that an application uses is specified in the file `web.xml` with the parameter `application.name` within the filter configuring the `JpsFilter` (for a servlet) or the interceptor configuring the `JpsInterceptor` (for an EJB). For details and samples, see Application Name (Stripe). If the application stripe is not specified (or left blank), then the system picks up an application stripe based on the application name.

On the other hand, the run-time policies that your application uses are specified in the file `system-jazn-data.xml` with the element `<application.name>`.

If those two names do not match or if you have not explicitly specified the stripe to use, then, most likely, your application is accessing the wrong policy stripe and, therefore, not able to authorized your application users as expected.

Solution

Ensure that you specify explicitly your application stripe, and that stripe is the one that your application is supposed to use. In most cases, the two names specified in those two different files (as explained above) match; however, in cases where several applications share the same policy stripe, they may differ.

J.5.4 User Gets Unexpected Permissions

This section explains the likely reasons why a user gets permissions other than those anticipated.

Symptom

A new user or a modified user gets unexpected permissions.

Diagnosis

This issue is likely to come up in cases where a user is added with the name of previously removed user, or an old user gets its name or uid changed. The common reason why the user may get more or less permissions than expected is that the policy store has not been properly updated before a user is removed or a user data is changed.

Solution

Before deleting a user, revoke all permissions, application roles, and enterprise groups that had been granted to the user. If you fail to remove all security artifacts referencing a user to be deleted, they are left dangling and, potentially, inherited if another user with the same name or uid is created at a later time.

Similar considerations apply to when a user name or uid is changed: all policies (grants, permissions, roles) referring to the old data must be updated so that they work as expected with the new data.

J.5.5 Granting Permissions in Java SE Applications

This section describes the correct way to code a grant in Java SE applications. Even though the problem described is not an issue in Java EE applications, for maximum portability, it is recommended that this solution be used in Java EE applications too.

Symptom

The application code includes a fragment like the following, by an application creates a grant:

```
Permission p = new FilePermission(resourceName, "write");
PrincipalEntry myPrincipal2 =
InfoFactory.newPrincipalEntry("weblogic.security.principal.WLSGroupImpl",
enterpriseRoleName2);
ap.grant(new Principal[]{myPrincipal2.getPrincipal()}, null, new Permission[]{p});
```

At runtime, however, the grant is not taking effect as expected.

Diagnosis

A bit of inspection indicates that the policy store repository includes the following attribute:

```
orcljaznjavaclass=oracle.security.jps.internal.policystore.UnresolvedPrincipal+cn=
enterpriseRoleName2
```

Solution

The lines of code above should be replaced by the following:

```
Permission p = new FilePermission (resourceName, "write");
PermissionEntry permEntry  = InfoFactory.newPermissionEntry(p.getClassName(),
p.getName(),  p.getActions());
ap.grant (new PrincipalEntry[] {myPrincipal2}, null, new PermissionEntry[]
{permEntry});
```

The solution uses the array `PrincipalEntry` instead of the array `Principal` and the array `PermissionEntry` instead of the array `Permission`.

Note: This same issue applies to the method `revoke`, which also has overloaded variants that accept `Principal[]` or `PrincipalEntry[]`

J.5.6 Application Policies Not Seen in 12c HA Environment

This section describes a sequence that results in application policies not taking effect in a 12c HA (High Availability) domain, and how to work around it.

Symptom

The following sequence throws an exception:

1. Deploy a custom application (packed with application policies) in a 12c HA environment, either to the domain administration server or to a domain managed server (but not to both).
2. Undeploy the application.
3. Redeploy the application to a server *different* from the one deployed to in step 1.

Step 3 above results in an exception and the application policies do not take effect. This issue is observed in 12c HA domains only.

Diagnosis

This issue is seen because the domain servers do not have their caches synchronized; note that the various servers run in distinct JVM's (Java Virtual Machine).

Suppose, for example, that the HA domain has three servers: server A (the administration server), server B (a managed server), and server C (a managed server). Suppose, further, that the application is first deployed to the domain administration server (server A), and then all three servers are restarted; the caches in servers A, B, and C will then have been initialized (with policies read from the security store) and are synchronized.

If the application is then undeployed (from server A), then the server A's cache will be cleared, but the caches in servers B and C will not. Further, if the application is redeployed, say, to server B, then the caches become out of synch, and an exception is thrown (because policy objects already exist).

Solution

The workaround to this issue is as follows:

After step 2 and before step 3 above, restart all servers in the HA domain. With this additional step, the modified procedure leads to synchronized caches in all servers in the HA domain, and then the application policies are seen as expected.

J.6 Troubleshooting Connections and Access

This section describes the following issues:

- Failure to Connect to the Embedded LDAP Authenticator
- Failure to Connect to an OID Server
- Failure to Access Data in the Credential Store
- Security Access Control Exception
- Failure to Establish an Anonymous SSL Connection

J.6.1 Failure to Connect to the Embedded LDAP Authenticator

This section explains the likely reasons why a connection to the embedded LDAP authenticator can fail.

Symptom

The connections that client applications use to request queries to the embedded LDAP authenticator, via the User and Role API, are stored and maintained in a connection pool. By default, and out-of-the-box, this pool is the JNDI pool, as specified in the file `jps-config.xml`.

If the number of current connections in the pool exceeds the maximum allowed by the LDAP service, client applications will not be able to connect to the service or, even when they are already connected, receive a “socket closed” exception. The server log would indicate, in this case, that the number of concurrent connections allowed has been exceeded.

Diagnosis

To avoid going over the limit, one needs to adjust the maximum number of concurrent connections allowed by the LDAP service as appropriate to the application's needs. This threshold needs to be finely tuned up: a too small maximum may not be sufficient (and cause the exception mentioned above); a too large maximum may risk a denial of service (DOS) attack. The correct maximum depends on your application and the particular LDAP service the application uses.

Solution

There are two alternative ways that resolve this issue:

- Increase the maximum number of concurrent connections allowed by the authenticator:
  - If the authenticator your application is using is the WebLogic Embedded LDAP authenticator, then edit the file *DomainName*/servers/*MyServerName*/data/ldap/conf/vde.prop, and increase the value of the property `vde.quota.max.conpersubject` from the default 100 to, for example, 200, or any other value.
  - Otherwise, if your application is using any other authenticator, consult the authenticator's documentation to learn how to modify the maximum.
- Edit the file *DomainName*/config/fmwconfig/jps-config.xml and remove the property `CONNECTION_POOL_CLASS` from the authenticator server instance (by default, this property has the value `oracle.security.idm.providers.stdldap.JNDIPool`.

Note that (a) these settings do not exclude each other, that is, you can carry out both of them; and (b) in any case, you must restart the server for the changes to take effect.

J.6.2 Failure to Connect to an OID Server

This section explains the likely reasons why a connection to an Oracle Internet Directory server can fail. This failure can also happen during reassociation.

Symptom

The migration of data from a source repository to a target OID server repository fails.

Diagnosis

Typically, this kind of problem is due to an incorrect set up of parameters in the target OID server.

For further probing into Oracle WebLogic Server log files, search any of the log files in the directories `DomainName/servers/AdminServer` or `DomainName/servers/ManagedServers` for the following strings: <Error>, <Critical>, and <Warning>.

For more information about identifying and solving errors, see Section J.1, "Diagnosing Security Errors."

Solution

Verify that all the target server data provided for the migration is valid. You may require the assistance of your OID server administrator to perform this validation.

Note: If you are using Fusion Middleware Control to reassociate to an OID server, ensure that you use the button **Test LDAP Authentication** before initiating the operation. Typically, this test catches incorrect supplied parameters.

J.6.3 Failure to Access Data in the Credential Store

This section explains a likely reason why an application fails to access data in the domain's credential store.

Symptom

An application fails to retrieve credential data from the domain's credential store, and an error message (containing lines like the one illustrated below) is logged (text in between brackets should describe information specific to the particular failure):

```
07/07/26 18:22:22 [JpsAuth] For permisson ( CredentialAccessPermission [target]
[actions]), domain that failed: ProtectionDomain
 cs(file:somePath/aWarFile.war/WEB-INF/classes/), []
```

Diagnosis

If an application is to access the credential store to perform an operation (such as retrieving a user password, for example), then its code must be granted the appropriate permission to perform the secured operation; otherwise, the application runs into an error like the one described above.

Solution

To grant the permission that an application requires to access the credential store, include the appropriate `CredentialAccessPermission` in the application's `jazn-data.xml`; this grant takes effect when the application is deployed or redeployed.

To include a permission using Fusion Middleware Control, see Section 10.3, "Managing Policies with Fusion Middleware Control."

To include a permission using a WLST command, see Section 10.4, "Managing Application Policies with WLST commands."

The following fragment of the file `jazn-data.xml` illustrates how to grant all code in the application `myApp` permission to read all credentials in the folder `myAlias`:

```
<jazn-data>
```

```
    <!--  codebase policy -->
    <jazn-policy>
        <grant>
            <grantee>
                <codesource>
 <!-- This grants applies to all code in the following directory -->
                    <url>${domain.home}/tmp/_WL_user/myApp/-</url>
                </codesource>
            </grantee>
            <permissions>
                <permission>

<class>oracle.security.jps.service.credstore.CredentialAccessPermission</class>
<!-- Allow read permission to all credentials under folder MY_MAP -->
                    <name>context=SYSTEM,mapName=MY_MAP,keyName=*</name>
                    <actions>read</actions>
                </permission>
            </permissions>
        </grant>
    </jazn-policy>
</jazn-data>
```

J.6.4 Security Access Control Exception

This section explains a reason why your code may run into a security access control exception.

Symptom

At run time, your application outputs an error like the following one (only the first few lines are shown):

```
<Jan 20, 2009 5:45:33 PM PST> <Error> <HTTP> <BEA-101020>
<[weblogic.servlet.internal.WebAppServletContext@140cf52
- appName: 'Application2',
name: 'Application2.war',
context-path: '/Application2',
spec-version: '2.5']
Servlet failed with
Exceptionjava.lang.RuntimeException:java.security.AccessControlException:access
denied
...
```

Diagnosis

The above error means that a call in your code does not have sufficient permissions to execute a secured operation.

Solution

Your code must be granted the appropriate permissions to execute the secured operation. Depending on the scope of the permission you would like to set, you have two alternatives.

The first one is to grant permission to all application code in the application's EAR or WAR files; in this case, the call to the secured operation can be inserted anywhere in the application code.

The second one is to grant permission to just a JAR file; in this case, the call to the secured operation must be inside a privileged block.

Each of these solutions is next illustrated by an application attempting to access the credential store.

The following fragment of an application`jazn-data.xml` illustrates how to set permission to read any key within the map MY_MAP in the credential store to *any* code within the directory `BasicAuth`:

```
<jazn-policy>
   <grant>
       <grantee>
           <codesource>
              <url>file:${domain.home}/servers/_WL_user/BasicAuth/-</url>
           </codesource>
       </grantee>
       <permissions>
           <permission>
             <class>
                 oracle.security.jps.service.credstore.CredentialAccessPermission
             </class>
             <name>context=SYSTEM,mapName=MY_MAP,keyName=*</name>
             <actions>read</actions>
          </permission>
      </permissions>
   </grant>
</jazn-policy>
```

If the permission is to be granted to the code in a particular EAR or WAR file, the `url` specification above would have to be changed to one like the following:

```
<url>file:${domain.home}/servers/_WL_user/jpsBasicAuth/.../BasicAuth.ear</url>
```

In both above cases, the call to read the credential store can be placed anywhere in the application code.

If, however, the permission is to be granted to just the code in a particular JAR file, the `url` specification above would have to be changed to one like the following:

```
<url>file:${domain.home}/servers/_WL_user/jpsBasicAuth/myJars/Foo.jar</url>
```

In this last case, the code in the file `Foo.jar` that calls a read operation on the credential store must be placed in an AccessController.doPrivileged block, as illustrated in the following code snippet:

```
import oracle.security.jps.*;
import oracle.security.jps.service.credstore.*;

JpsContextFactory factory = JpsContextFactory.getContextFactory();
JpsContext jpsContext = factory.getContext();
final CredentialStore store =
jpsContext.getServiceInstance(CredentialStore.class);
Credential cred = AccessController.doPrivileged(new
PrivilegedExceptionAction<PasswordCredential>() {
    public PasswordCredential run() throws JpsException {
        return store.getCredential("MY_MAP", "anyKey");
    }
});

PasswordCredential pwdCred = (PasswordCredential)cred;
...
```

Note that since our sample grant above allows only read permission, none of the set or reset operations work, even inside a privileged block.

J.6.5 Failure to Establish an Anonymous SSL Connection

This section explains the likely reasons why you are not able to establish an anonymous SSL connection while reassociating policies and credentials.

Symptom

A step in the reassociation of file-based policies and credentials to an Oracle Internet Directory server with Fusion Middleware Control involves testing the anonymous SSL connection to the OID server (specifically with the button Test LDAP). This test fails.

Diagnosis

Your target OID server must be trusted by the Oracle WebLogic Server and the port number you are using to the OID server must be an SSL port.

Solution

Establishing a connection to an OID server requires some previous configuration on the OID server. For details, see Section 9.2.1, "Prerequisites to Using an LDAP-Based Security Store."

In addition, to use an anonymous SSL connection, you must enter a port that has been set for receiving secure data. If your OID server has not been configured with such a port, the connection fails.

Ensure that the supplied OID server port is an SSL port configured to listen in anonymous SSL mode, and that the supplied server name reachable. Typically, the setting of this port involves an OID server administrator.

J.7 Troubleshooting Oracle Business Intelligence Reporting

This section describes common problems and solutions for Oracle Business Intelligence when used as a reporting tool for Oracle Fusion Middleware security. It contains the following topics:

- Audit Templates for Oracle Business Intelligence Publisher
- Oracle Business Intelligence Publisher Time Zone

J.7.1 Audit Templates for Oracle Business Intelligence Publisher

To view Oracle Fusion Middleware Audit Framework reports in a reporting application, you must use the audit schema.

For details, see Section C.2.

J.7.2 Oracle Business Intelligence Publisher Time Zone

You may see problems with Oracle Fusion Middleware Audit Framework reports if Oracle Business Intelligence Publisher and the database are installed in sites with different time zones.

To avoid this issue, ensure that Oracle Business Intelligence Publisher and the database are installed in the same time zone.

J.8 Troubleshooting Searching

This section describes the following issues:

- Search Failure when Matching Attribute in Policy Store

- Search Failure with an Unknown Host Exception

J.8.1 Search Failure when Matching Attribute in Policy Store

This section describes a reason why cataloging of an attribute is needed.

Symptom

While searching the policy store, an exception similar to the following is encountered:

```
oracle.security.jps.service.policystore.PolicyStoreOperationNotAllowedException
javax.naming.OperationNotSupportedException:
[LDAP: error code 53 - Function Not Implemented, search filter attribute
orcljpsresourcetypename is not indexed/cataloged];
remaining name 'cn=Permissions,cn=JAASPolicy,cn=IDCCS, cn=sprint6_policy_
domain,cn=JPSContext,cn=FusionAppsPolicies'
```

Diagnosis

The error above indicates that the attribute `orcljpsresourcetypename` must be cataloged before it is used in a filter to search the policy store.

Solution

An Oracle Internet Directory attribute used in a search filter must be indexed and cataloged. Indexing and cataloging are optional operations, in general, but required for OPSS-related attributes. Attribute indexing and cataloging is automatically performed by the WLST command `reassociateSecurityStore`.

For details about managing attribute catalogs and identifying whether an attribute is indexed, see the following sections in *Oracle Fusion Middleware Administrator's Guide for Oracle Internet Directory*:

To catalog attributes manually use the command `ldapmodify`, as illustrated bellow:

```
>ldapmodify -h <host> -p <port> -D <bind DN> -w <bind password> -v -f <catalogue
modify ldif file name>
```

To catalog, for example, the attributes `createtimestamp` and `modifytimestamp` use an LDIF file like the following:

```
dn: cn=catalogs
changetype: modify
add: orclindexedattribute
orclindexedattribute: modifytimestamp
orclindexedattribute: createtimestamp
```

The list of Oracle Internet Directory attributes that must be indexed follows:

```
OrclJpsAllResourcKeyword
OrclJpsAllResourceActionKeyword
OrclJpsEncodedAttributes
OrclJpsExtensionType
OrclJpsResourceConverter
OrclJpsResourceMatchingAlg
OrclJpsResourceMatchingAlgorithm
OrclJpsResourceNameExpression
OrclJpsRoleType
orcOesAppAttributes
orclASInstanceName
orclFarmName
orclJPSObjGUID
orclJavaApplicationEntityRef
```

```
orclJpsPolicyDomainName
orclJpsResourceActionsetMembers
orclJpsResourceExpression
orclJpsResourceLocalityRef
orclJpsResourceMatcherJavaclass
orclJpsResourceName
orclJpsResourceTypeActionAttrs
orclJpsResourceTypeActionNames
orclJpsResourceTypeName
orclJpsResourceTypeProviderName
orclJpsResourceTypeResourceAttrs
orclJpsRoleCategory
orclJpsRoleMemberExpression
orclJpsSuperResourceType
orclOESActCollectionName
orclOESActCollectionRfs
orclOESActionAttributes
orclOESActionConstraint
orclOESAlgorithmJavaClass
orclOESAllResourceType
orclOESAllowAdviceRef
orclOESAllowObligationRef
orclOESAttributeCategory
orclOESAttributeCollectionHandlerFunctionName
orclOESAttributeCollectionHandlerPackageName
orclOESAttributeCollectionHandlerSchemaName
orclOESAttributeCollectionName
orclOESAttributeDataType
orclOESAttributeIssuer
orclOESAttributeNamespace
orclOESAttributeType
orclOESCombinerParameter
orclOESConditionExpression
orclOESDSColumnAttrs
orclOESDSPrimKey
orclOESDataSourceCtrnt
orclOESDataSourceName
orclOESDataSourceType
orclOESDefaultPolSetRef
orclOESDenyAdviceRef
orclOESDenyObligationRef
orclOESDistributionEndTime
orclOESDistributionID
orclOESDistributionIssuer
orclOESDistributionMessage
orclOESDistributionPercentComplete
orclOESDistributionStartTime
orclOESEffect
orclOESEnvAttributes
orclOESEnvConstraint
orclOESExecutionFrequency
orclOESExpression
orclOESFunctionCategory
orclOESFunctionClass
orclOESFunctionParameters
orclOESFunctionReturnType
orclOESIsSensitive
orclOESIsSingleValued
orclOESMatchInfo
orclOESMaxDelegationDepth
```

```
orclOESObligationFulfillOn
orclOESPDPAddress
orclOESPDPConfigurationID
orclOESPDPInstanceName
orclOESPDPStatusSuccess
orclOESPIPType
orclOESPolicyCategory
orclOESPolicyCombinerParameter
orclOESPolicyCombiningAlgorithmRef
orclOESPolicyDefaults
orclOESPolicyExtension
orclOESPolicyIssuer
orclOESPolicyRef
orclOESPolicyRuleOrder
orclOESPolicyRuleRef
orclOESPolicySetCategory
orclOESPolicySetDefaults
orclOESPolicySetRef
orclOESPolicySetType
orclOESPolicyType
orclOESPolicyVersion
orclOESPresenceRequired
orclOESPrincConstraint
orclOESPrincipalAttributes
orclOESResConstraint
orclOESResTypeCategory
orclOESResourceAttributes
orclOESResourceHirchyType
orclOESResourceNameDelim
orclOESResourceParentName
orclOESRoleMapping
orclOESRuleCombinerParameter
orclOESRuleCombiningAlgorithmRef
orclOESSQLExpression
orclOESSetCombinerParameter
orclOESSetMemberOrder
orclOESTargetExpression
orclOESXMLExpression
orclassignedpermissions
orclassignedroles
orcldistributionversion
orcljazncodebase
orcljaznjavaclass
orcljaznpermissionactions
orcljaznpermissionresourceref
orcljaznpermissionsigner
orcljaznpermissiontarget
orcljaznpermissiontargetexpr
orcljaznprincipal
orcljaznsigner
orcljpsRuleCombiningAlgorithmRef
orcljpsactionsdelim
orcljpsassignee
orclrolescope
```

J.8.2 Search Failure with an Unknown Host Exception

When searching for information in an Active Directory environment that is configured for OID referrals, the referrals fail if the host being referred to is in a different domain than the Active Directory server.

Symptom

When a user requests a resource, at times verification of the user's identity can fail due to an inability to validate the user's identity in the directory. This error can occur in an Active Directory environment when the user's browser runs on a non-Windows computer, or if the user's browser runs on a Windows computer that is not in the Active Directory server domain.

Diagnosis

This problem can arise due to OID referral chasing. An OID referral occurs when a domain controller does not have the section of the directory tree where a requested object resides. The domain controller refers the client to another destination so that the client can conduct a DNS search for another domain controller. If the client is configured to chase referrals, the search can continue.

For the scenario where the user has a Windows-based computer, an issue can occur with OID referrals if the client's domain controller does not have a trust relationship with the Active Directory domain controller.

Solution

If you encounter this issue, add the entry for the Active Directory host's address in the following list:

```
WINDOWS_HOME_DIRECTORY\system32\drivers\etc\hosts
```

On Windows XP, the list is located here:

```
C:\WINDOWS\system32\drivers\etc\host
```

On a Unix-based system, add this entry to the `/etc/hosts` file, using the format:

```
IP_address_of_AD_host AD_host_name
```

where *`AD_host_name`* is the host name specified in the referral, for example:

```
123.123.123.123 my2003ad.com
```

J.9 Troubleshooting Versioning

This section describes the following issues:

- Incompatible Versions of Binaries and Policy Store
- Incompatible Versions of Policy Stores

J.9.1 Incompatible Versions of Binaries and Policy Store

This section describes the reason why the server would throw the exception `PolicyStoreIncompatibleVersionException`. See also Incompatible Versions of Policy Stores.

Symptom

An error similar to the following is logged or issued by the server:

```
Oracle.security.jps.service.policystore. PolicyStoreIncompatibleVersionException
JPS-06100: Policy Store version 11.1.1.5.0 and Oracle Platform Security Services
version 11.1.1.4.0 are not compatible.
```

Diagnosis

The above exception indicates that the domain OPSS binaries version (11.1.1.4.0) and the policy store version (11.1.1.5.0) used by that domain have incompatible versions. The version of the policy store is established during reassociation and that version is used until the policy store is upgraded to a newer version.

OPSS domain binary versions are *backward* compatible with policy store versions used by that domain, but they are not forward compatible. Thus, the error above indicates that the policy store has version newer that the version of the OPSS binaries. 11.1.1.4.0 OPSS binaries cannot use a newer version of the policy store.

Here are three scenarios where OPSS binaries ends up running into this incompatibility.

- Scenario 1
  - Domain1 and Domain2 point to the same policy store; Domain1, Domain2, and that policy store are all version 11.1.1.3.0.
  - The binaries in Domain1 are upgraded to 11.1.1.4.0.
  - The policy store is upgraded to 11.1.1.4.0 (using the command `upgradeOPSS`).
  - When the Domain2 is brought up again, its version and the policy store version are incompatible.
- Scenario 2
  - Domain1 points to a policy store and both are version 11.1.1.3.0.
  - An attempt to migrate the policy store to a 11.1.1.4.0 policy store fails because the migration would render a scenario with incompatible versions.

    Migration is supported only when the versions of the OPSS binaries and the policy store are same.
- Scenario 3
  - A 11.1.1.3.0 Domain1 attempts to join a 11.1.1.4.0 policy store (in some other domain), using the command `reassociateSecurityStore` with the join argument.
  - The operation fails because the sharing would render a scenario with incompatible versions.

    Reassociation is supported only when the versions of the OPSS binaries and the policy store are same.

Solution

The solution, common to all three scenarios above, is either one of the following:

- Update the domain OPSS binaries to match the version of the policy store the domain is pointing to.
- Reassociate the domain policy store to a policy store that has version not newer than the version of the domain OPSS binaries.

J.9.2 Incompatible Versions of Policy Stores

This section describes the reason why, while migrating the OPSS security store, the exception `PolicyStoreIncompatibleVersionException` is encountered. See also Incompatible Versions of Binaries and Policy Store.

The above exception indicates that the version of the source store is *higher* than the version of the target store, an invalid combination of versions. Migration proceeds only if the version of the source is not higher than the version of the target.

The workaround is to upgrade the target store to a version compatible with the version of the source store.

J.10 Troubleshooting Other Errors

This section describes the following issues:

- Runtime Permission Check Failure
- Tablespace Needs Resizing
- Oracle Internet Directory Exception
- User and Role API Failure
- Characters in Policies
- Invalid Key Size

J.10.1 Runtime Permission Check Failure

This section explains a reason why a permission may fail to pass a permission check.

Symptom

At run time, your application outputs an error like the following one (only the first few lines are shown):

```
[JpsAuth] Check Permission
          PolicyContext:        [null]
          Resource/Target:      [test]
          Action:               [null]
          Permission Class:     [com.oracle.permission.SimplePermission]
          Evaluator:            [ACC]
          Result:               [FAILED]
          Failed
ProtectionDomain:ClassLoader=weblogic.utils.classloaders.ChangeAwareClassLoader@14
061a8
finder: weblogic.utils.classloaders.CodeGenClassFinder@2dce7a8
annotation: Application2@Application2.war
CodeSource=file:/scratch/servers/AdminServer/tmp/permission/TestServlet$1.class
Principals=total 0 of principals<no principals>
Permissions=(
(oracle.security.jps.service.credstore.CredentialAccessPermission
context=SYSTEM,mapName=default,keyName=* read,write)
(java.net.SocketPermission localhost:1024- listen,resolve)
(oracle.security.jps.service.policystore.PolicyStoreAccessPermission
context=APPLICATION,name=* getApplicationPolicy)
(oracle.security.jps.service.policystore.PolicyStoreAccessPermission
context=SYSTEM getConfiguredApplications)
(com.oracle.permission.SimplePermission *)
...
java.security.AccessControlException: access denied
(com.oracle.permission.SimplePermission test)...
```

Diagnosis

When two or more applications share a permission class, that permission class must be set in the system class path so the class is loaded just once. Otherwise, only the first application loading the class passes the permission check; other ones loading the same class thereafter may fail the permission check and output an error like the one illustrated above.

Note that even though the permission class is in the permission collection (see bold text in sample output above), the check fails and the access is denied. This is because, at that point, the environment contains *several* instances of a permission class with the same name.

Solution

Ensure that if two or more applications to be run in the same domain share a permission class, then include that class in the system class path.

J.10.2 Tablespace Needs Resizing

This section describes a reason why policy SQL operations may fail in case of DB-based OPSS security store.

Symptom

While performing policy operations, the Oracle database issues a warning like the following:

```
"ORA-1652: unable to extend temp segment by 128 in tablespace"
```

Diagnosis

The reason for this error is that the temporary table used by the operations is full, that is, it has no more free blocks available.

Solution

To solve this issue, proceed as follows:

1. Extend the size of the temporary table to 4096 M, as illustrated in the following command:

   ```
   ALTER TABLESPACE "<TEMP_TABLESPACE_NAME>" ADD TEMPFILE '<data_file_name>' size
   4096M
   ```

2. Restart the administration server.

J.10.3 Oracle Internet Directory Exception

A domain with a security store using Oracle Internet Directory 11.1.1.6.0 runs into an OID exception like the following:

```
[LDAP: error code 53 - OID-5018: Cataloging for attr orcljpsextensiontype
is already in progress.]:cn=catalogs
```

The attribute reported (`orcljpsextensiontype` in the above sample) may differ, but the solution is the same, namely, that you need to apply a patch to fix bug 13782459 in Oracle Internet Directory 11.1.1.6.0. For a list of Oracle Internet Directory patches, see Section 9.2, "Using an LDAP-Based OPSS Security Store."

J.10.4 User and Role API Failure

This section explains some reasons why you may fail to access data in a domain authenticator with the User and Role API.

Symptom

The User and Role API fails to access data in a configured authenticator.

Diagnosis 1

The OPSS User and Role API can access data *only* in the first LDAP authenticator configured in a domain. At least one such authenticator must be present in a domain. The API access to that first LDAP authenticator fails if the target user is not present in that authenticator, even though that user is present in some other domain authenticator.

Solution 1

Enter the missing user in the first LDAP authenticator, or reorder the list of LDAP authenticators in your domain.

Diagnosis 2

Let's assume that the target user on which the API that fails *is* present in the first LDAP authenticator configured in your domain.

By default, the User and Role API uses the attribute `uid` to perform user search in an LDAP authenticator. If for some reason, a user entered in the LDAP is lacking this attribute, then the User and Role API fails.

Solution 2

Ensure that all users in the first LDAP authenticator have the attribute `uid` set.

Note: If you are developing a Java SE application (and only in this case) and want the User and Role API to employ an attribute other than the default one (`uid`) to search users, say `mail` for example, then the properties `username.attr` and `user.login.attr` must be configured in the LDAP provider instance of the identity store (in the file `jps-config-jse.xml`) as illustrated in the following code snippet:

```
<serviceInstance provider="idstore.ldap.provider"
name="idstore.ldap">
   ...
   <property name="username.attr" value="mail"/>
   <property name="user.login.attr" value="mail"/>
   ...
</serviceInstance>
```

To add properties to a provider instance with a prescribed script, see Section E.1, "Configuring OPSS Service Provider Instances with a Script."

J.10.5 Characters in Policies

This section explains several issues related to characters used in policies, in the following sections:

- Use of Special Characters in Oracle Internet Directory 10.1.4.3
- XML Policy Store that Contains Certain Characters
- Characters in Application Role Names
- Missing Newline Characters in XML Policy Store

J.10.5.1 Use of Special Characters in Oracle Internet Directory 10.1.4.3

When the policy store is an Oracle Internet Directory 10.1.4.3 repository, then using the characters '*', '(', ')', or '\' in the RFC 2252/2253 filter results in error 53 (DSA unwilling to perform). To resolve this error, apply the patch for bug number 7711351 to Oracle Internet Directory 10.1.4.3.

J.10.5.2 XML Policy Store that Contains Certain Characters

The issue explained in this section is relevant to XML Policy Stores only, that is, it does not apply to OID policy stores.

The following characters:

```
 < "  &  $ ? * , / \ ` :    ( ) ^ ' % +  { }
```

are not recommended as part of an Application Role name when using a file-based policy store.

If it becomes necessary to use one of those characters to create a role, for example, then ensure that such characters are escaped in the input to API Policy Store methods like `ApplicationPolicy.searchAppRoles()`, so they return correct results.

For example, if you have an application role named "appRole^$" it will need to be input as `ApplicationPolicy.searchAppRoles("appRole\\^\\$")` to find the match in the policy store.

Alternatively, you could use a wild card in the search expression without including these escaped special characters, and it will also match that application role:

```
ApplicationPolicy.searchAppRoles("appRole*")
```

J.10.5.3 Characters in Application Role Names

An application role name is a string of printable characters other than white space, that is, it can contain alpha-numeric characters (ASCII or Unicode) and other printable characters (such as underscore or square brackets) *except* for white space. This rule applies to all three kinds of supported storage: XML, OID, and DB.

J.10.5.4 Missing Newline Characters in XML Policy Store

In a file-based policy store, a new-line character is required between the closing of a `<permission>` or `<principal>` tag and the opening of the following one.

Following are examples of strings illustrating incorrect and correct formats.

Incorrect example fragment of policy store:

```
<permission>
    <class>java.lang.RuntimePermission</class>
    <name>getClassLoader</name>
 </permission> <permission>
      <class>java.io.FilePermission</class>
      <name>/foo</name>
      <actions>read,write</actions>
 </permission>
```

Correct example fragment of policy store:

```
<permission>
    <class>java.lang.RuntimePermission</class>
    <name>getClassLoader</name>
</permission>
<permission>
      <class>java.io.FilePermission</class>
      <name>/foo</name>
      <actions>read,write</actions>
</permission>
```

J.10.6 Invalid Key Size

This section explains some reasons why you may get an invalid key size exception.

Symptom

An exception with the following message is logged:
"java.security.InvalidKeyException: Illegal key size"

```
java.security.InvalidKeyException: Illegal key size
```

Diagnosis 1

During domain creation, OPSS uses the keystore service to create an AES symmetric key that is used for encrypting security store data. OPSS tries to generate a 256-bit key first. If it is not supported, then it tries to generate a192-bit key, and if that is not supported either, it creates a 128-bit key. If 128-bit keys are not supported, the above exception is thrown.

Diagnosis 2

During OPSS startup (for both Java EE and SE applications), if the OPSS encryption key cannot be read because the runtime JDK does not support the AES algorithm, or if the domain provisioned key size differs from the size that the JDK supports, the above exception is thrown.

Solution

A common solution for the above is the following: install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files in the JDK, and then retry the domain creation or OPSS startup.

J.11 Need Further Help?

You can find more solutions on My Oracle Support (formerly MetaLink) at `http://myoraclesupport.oracle.com`. If you do not find a solution to your problem, log a service request.