Oracle® Fusion Middleware

Securing Applications with Oracle Platform Security Services

12c (12.2.1.4.0)
F29738-04
April 2024

ORACLE

Oracle Fusion Middleware Securing Applications with Oracle Platform Security Services, 12c (12.2.1.4.0)

F29738-04

Contents

8 Configuring the Identity Store

9 Configuring the Security Store

13 Introduction to Oracle Fusion Middleware Audit Framework

14 Managing Audit

17 The Security Model

18 Developing with the Credential Store Framework

19 Developing with the User and Role API

20 Developing with the Identity Governance Framework

21 Developing with the Keystore Service

22 Developing with Oracle Fusion Middleware Audit Framework

23 Configuring Java EE Applications to Use OPSS

24 Configuring Java SE Applications to Use OPSS

Part V Reference

A OPSS Configuration File Reference

B File Store References

C Oracle Fusion Middleware Audit Framework Reference

D User and Role API Reference

E Administration with Scripts and MBeans

F OPSS System and Configuration Properties

Preface

This guide explains the features and administration of Oracle Platform Security Services (OPSS).

Audience

The intended audience of this guide are experienced Java developers, domain administrators, deployers, and application managers who want to understand and use OPSS.

This guide includes several parts, each of which groups related major topics. Parts I through III are relevant to security administrators. Parts IV contains information about the 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*
- *Securing Web Services and Managing Policies with Oracle Web Services Manager*
- *Administering Oracle Internet Directory*
- *Developing Fusion Web Applications with Oracle Application Development Framework*
- *Administering Oracle Entitlements Server*
- *Understanding the WebLogic Scripting Tool*
- *WLST Command Reference for Infrastructure Security*

- *Creating Schemas with the Repository Creation Utility*
- For links to API documentation, see OPSS API References.

For a comprehensive list of Oracle documentation or to search for a particular topic within Oracle documentation libraries, see `http://docs.oracle.com`.

Conventions

The following text conventions are used in this guide:

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 summarizes the main changes introduced in the 12.2.1.x releases.

Changes in This Document for Release 12.2.1.4.0

There are no updates to this guide for the Oracle Fusion Middleware 12.2.1.4.0 release.

Changes in This Document for Release 12.2.1.3.0

Release 12.2.1.3.0 includes the following changes:

- When migrating application policies for an application using the `migrateSecurityStore` WLST command, the behavior of how policies are merged using the `overwrite` parameter has been changed. See Migrating Application Policies with migrateSecurityStore.
- The Keystore Service (KSS) has been updated to support Subject Alternative Name (SAN) SSL certificates. See Managing Certificates with WLST and Managing Certificates with Fusion Middleware Control.
- The sections “Upgrading Security to 12.2.1.*x*” and “Compatibility Table for 11g and 12c Versions” have been moved to Securing Datastores in *Planning an Upgrade of Oracle Fusion Middleware*.
- Added an example that illustrates how you can set up a domain that uses either third-party CA signed or internal CA signed certificates throughout the domain in place of the demonstration CA certificates. See Setting Up a Security Hardened Domain: An Example.
- Clarified the procedures for migrating credentials in the same domain and across domains. See Migrating Credentials with migrateSecurityStore.

Changes in This Document for Release 12.2.1.2.0

In release 12.2.1.2.0, the changes to this document include:

- A procedure that describes how to synchronize the local keystore with the security store. See Synchronizing the Local Keystore with the Security Store.

- Procedures that describe how to replace demonstration certificates with third-party or domain CA signed certificates. Demonstration CA certificates should not be used in a production environment. See Replacing Demonstration CA Signed Certificates.
- Troubleshooting information about database connection errors. See Database Connection Exception.

Changes in This Document for Release 12.2.1.1.0

In release 12.2.1.1.0, this document has been changed as follows:

- The procedures in “Upgrading Security to 12.2.1.*x*” have been updated and streamlined to improve usability.

> **Note:**
>
> In 12.2.1.3.0, these procedures have been moved to Securing Datastores in *Planning an Upgrade of Oracle Fusion Middleware*.

New Features in Release 12.2.1.0.0

The new features and major changes introduced in release 12.2.1.0.0 include the following:

- Support for IBM DB2 and Microsoft SQL server databases. OPSS supports two new databases as repositories for security stores. See Supported File, LDAP, and Database Stores.
- The ability to create database views of audit records at registration time using the Dynamic Metadata Model. See Audit Views Created at Registration.
- New audit `createIAUView` and `getIAUViewInfo` commands, that allow you to create and to get information about audit database views. See Audit Configuration WLST Commands in *WLST Command Reference for Infrastructure Security*.
- A new `merge.jdkcacerts.with.trust` property, which specifies whether to return public certification authority certificates in the `kss://system/publicacerts` keystore with a keystore query to `kss://system/trust`. See About Certificates.
- The OPSS REST API.

Part I
Understanding Security Concepts

This part contains the following chapters:

- Introduction to Oracle Platform Security Services
- Understanding Users and Roles
- Understanding Identities, Policies, Credentials, Keys, Certificates, and Audit
- About the Security Store

1

Introduction to Oracle Platform Security Services

Oracle Platform Security Services (OPSS) provides development teams a portable and integrated framework to secure Java Platform Standard Edition (Java SE) and Java Platform Enterprise Edition (Java EE) applications.
This chapter incudes the following sections:

The scope of this document does *not* include Oracle Web Services security. For information about Oracle Web Services security, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

- What Is OPSS?
- OPSS Architecture Overview
- OPSS for Developers
- ADF Security Overview

What Is OPSS?

OPSS is the underlying security platform that provides security to Oracle Fusion Middleware products, including Oracle WebLogic Server, service-oriented architecture (SOA) applications, Oracle WebCenter, Oracle Application Development Framework (Oracle ADF) applications, and Oracle Entitlements Server (OES).

OPSS provides an abstraction layer in the form of application programming interfaces (APIs) that insulate developers from security and identity management implementation details. 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 services across the enterprise.

OPSS is supported on WebLogic Server and is installed with Oracle Fusion Middleware Infrastructure.

OPSS Main Features

OPSS complies with the following standards: Role-Based Access Control (RBAC), Java EE, Java Authorization and Authentication Services (JAAS), and Java Authorization Contract for Containers (JACC).

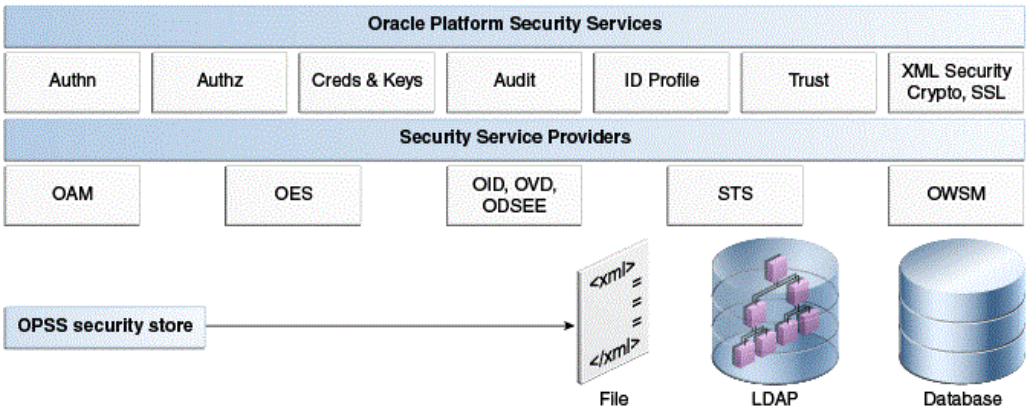
Built upon these standards, OPSS provides an integrated security platform that supports:

- Authentication
- Identity assertion
- Authorization
- Managing roles and role mappings
- Managing application policies and credentials
- Managing keys and certificates
- Audit
- Identity Virtualization
- Security APIs

OPSS Architecture Overview

OPSS includes the WebLogic Server security providers and the Oracle Fusion Middleware security frameworks. Figure 1-1 illustrates the OPSS architecture.

Figure 1-1 The OPSS Architecture



The OPSS architecture has the following layers:

- The security services, which includes authorization, authentication, credentials, identity and trust, Secure Sockets Layer (SSL), and cryptographic services. For authentication, OPSS uses WebLogic Server Authentication providers.
- The WebLogic Server security providers. The Security Services Provider Interface (SSPI) provides Java container security, resource-based authorization for the environment, and APIs for implementing security providers. A module implementing any of these interfaces can be plugged into the framework to provide a particular type of security service, such as custom authentication or a particular role mapping.
- The security store with one of three types of repositories: file, Lightweight Directory Access Protocol (LDAP), or database.

See also:

The Security Service Provider Interfaces (SSPIs) in *Understanding Security for Oracle WebLogic Server*

Benefits of Using OPSS

OPSS offers many benefits, including the following:

- Allows developers to focus on application and domain problems
- Supports enterprise deployments
- Supports LDAP servers and SSO systems
- Is certified on WebLogic Server
- Integrates with Oracle products and technologies
- Offers a consistent security experience for developers and security administrators
- Provides a uniform set of APIs for all types of applications
- Optimizes development time by offering abstraction layers
- Provides a simplified application security maintenance
- Allows changing security rules without affecting application code
- Eases administration
- Integrates with identity management systems

OPSS provides security for Java EE applications, Oracle Fusion Middleware applications, and Java SE applications. It also provides the tools to administer all security in the enterprise, and allows changing security configurations without modifying application code.

Using OPSS APIs, developers secure all types of applications and integrate them with other security systems, such as LDAP, databases, and custom security components.

Ready-to-use, the default security provider (embedded LDAP server) is available, but domains can use identity data in other LDAP repositories.

OPSS for Developers

The following sections summarize the main security features that you use in Java applications:

- About Java EE Application Security
- About Java SE Application Security

About Java EE Application Security

Java EE applications can use several interfaces to access and maintain security data, including those provided with Credential Store Framework API, User and Role API, Identity Governance Framework API, and Keystore Service API. Using these interfaces Java EE applications set and retrieve user attributes, and manage policies, keys, and certificates.

Java EE applications can use authentication and authorization declaratively, with specifications in the `web.xml` file, or programmatically, with calls to the `isUserInRole` and `isCallerInRole` methods.

Java EE applications can use custom authentication providers and control authentication between Java servlets and Enterprise JavaBeans (EJB) using roles and enterprise groups.

See also:

Security Integration Use Cases

About the User and Role API

Configuring Java EE Applications to Use OPSS

About Java SE Application Security

Most of the OPSS features available for Java EE applications are also available for Java SE applications, but there are some differences that apply to Java SE applications only, including the following:

- Applications must use the `AppSecurityContext.JpsStartup.start` method before calling any security operations.
- Application security configuration is defined in the `jps-config-jse.xml`file by default installed in the following location:

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

  To specify a different location, use the `oracle.security.jps.config` property:

  ```
  -Doracle.security.jps.config=pathToConfigFile
  ```
- Applications use standard JAAS login modules by implementing a custom authentication provider that calls the login module.
- The following Java archive (JAR) file must be added to the class path:

  ```
  $ORACLE_HOME/oracle_common/modules/oracle.jps_12.2.1/jps-manifest.jar
  ```

See also:

Authentication Providers in *Developing Security Providers for Oracle WebLogic Server*

Using OPSS in Java SE Applications

ADF Security Overview

Oracle ADF is an end-to-end Java EE framework that simplifies development by providing infrastructure services and a visual development experience. Oracle ADF is integrated with Oracle JDeveloper.

ADF Security is based on the JAAS security model, uses OPSS and permission-based authorization, and simplifies the configuration of application security with visual editors and ADF Security wizard.

During the development of an Oracle ADF application, the authentication providers are configured with Oracle WebLogic Server Administration Console, and policies are kept in a file.

To summarize, ADF security allows you to:

- Control the granularity of declarative security
- Simplify permission with the role hierarchy
- Access to Oracle ADF resources
- Integrate with JDeveloper for quick development and test cycles

See also:

Enabling ADF Security in a Fusion Web Application in *Developing Fusion Web Applications with Oracle Application Development Framework*

Securing Oracle ADF Applications

- Oracle ADF Application Security

Oracle ADF Application Security

Oracle ADF simplifies the development of Java EE applications with JDeveloper by minimizing the code that implements the application infrastructure. This helps developers focus on application features.

Oracle ADF leverages container authentication and uses JAAS based authorization to control access to Oracle ADF resources. Policies may include specific application roles and JAAS authorization permissions. Oracle ADF connection credentials are stored in the security store.

Oracle ADF and WebCenter applications include WebLogic Server Authentication providers and may include a single sign-on solution such as Oracle Single Sign-On.

See also:

Overview of Single Sign-On with Microsoft Clients in *Administering Security for Oracle WebLogic Server*

Securing Oracle ADF Applications

2

Understanding Users and Roles

This chapter defines terms used throughout this document, and it introduces the role category and role mapping. Special roles include the authenticated and anonymous roles. This chapter includes the following sections:

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

Terminology

This section defines OPSS terms used throughout this document.

Users and Groups

A *user* is an end-user accessing a service. A *group* is a subset of users and other groups, so a group has a hierarchical structure.

An *authenticated user* is a user whose credentials have been validated. To authenticate users, OPSS uses WebLogic Server Authentication providers.

An *anonymous user* is an authenticated user who is permitted access to only unprotected resources.

Application Stripe

An *application stripe* or *stripe* (when the application is understood) is a subset of policies. An application chooses the policies to use by specifying a stripe in the security store. Several applications can use the same stripe.

Roles

An *enterprise role* or *enterprise group* is a collection of users and groups.

An *application role* is a collection of users and other application roles. Application roles are defined in application policies and they are not necessarily known to a Java container.

Application roles can be mapped many-to-many to enterprise roles. For example, the `employee` enterprise role (in the identity store) can be mapped to the `helpdesk service request` application role (in one stripe) and to the `self service HR` application role (in some other stripe).

Principal

A *principal* is the identity to which a policy grants permissions. A principal can be a user, an external role, or an application role.

Policies

An *application policy* is a policy that specifies a set of permissions for principals, such as viewing application web pages or modifying application reports. Application policies must have at least one principal and can specify either separate permissions or a permission set, but not both.

A *system policy* is a policy that specifies a set of permissions for a principal or a codesource. The scope of system policies is the entire domain. System policies grant permissions to codesources and principals, while application policies grant permissions to principals only. Application and system policies are specified in the `jazn-data.xml` file.

Subject

The *subject* is a collection of principals and user credentials, such as passwords and cryptographic keys. The authentication service populates the subject with users, groups, application roles, and credentials. The OPSS subject is critical in identity propagation across domains.

OPSS Configuration File

The OPSS configuration file where all security services for the entire domain are specified. The specifications in the OPSS configuration file apply to all servers and applications running in the domain.

By default, the OPSS configuration file is the `jps-config.xml` file (for Java EE applications) and the `jps-config-jse.xml` file (for Java SE applications) and they are located in the `$DOMAIN_HOME/config/fmwconfig` directory.

OPSS Context

The *context* defines a collection of services instances common to a domain. Contexts are specified in the OPSS configuration file.

Security Stores

The *identity store* is the repository of enterprise users and groups and must be LDAP. Ready-to-use, users and groups can be stored in the default provider (embedded LDAP server).

The *security store* is the repository of system and application policies, credentials, audit data, keys, and certificates used by all applications running in a domain.

The *policy store* refers to the portion or the security store where application and system policies are kept. The *credential store* refers to the portion of the security store

where credentials are kept. The *keystore* refers to the portion of the security store where keys and certificates are kept. The *truststore* refers to a keystore where trusted certificates of third-party certificate authorities are kept.

For information about audit terms, see Audit Terminology.

> **See also:**
>
> Role Mapping
>
> About the Authenticated Role
>
> About the Anonymous User and Role
>
> Setting the Application Stripe

Role Mapping

OPSS supports the many-to-many mapping of application roles to enterprise groups. This allows users in enterprise groups to access application resources as specified by application roles.

Mapping an application role to an enterprise group rewrites the permission of the enterprise group as the union of its permissions and those of the mapped role. Therefore, the mapping may augment the permissions of the enterprise group but never removes any permission from it.

After deploying the application, you map application roles to enterprise groups with WebLogic Scripting Tool (WLST), Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control), or Oracle Entitlements Server (OES).

> **See also:**
>
> Managing Application Roles for information about managing roles with Fusion Middleware Control.

Permission Inheritance and the Role Hierarchy

Roles can be structured hierarchically by the relation “is a member of," and a role can have as members users or roles.

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`. In addition, `roleA` may have its own particular permissions.

> **See also:**
>
> grantAppRole and revokeAppRole in *WLST Command Reference for Infrastructure Security*
>
> *Administering Oracle Entitlements Server*

- Role Hierarchy Example

Role Hierarchy Example

The following example illustrates a role hierarchy of nested application users and roles. In the example, the `developerAppRole` role has the following members:

```
developer
developer_group
managerAppRole
directorAppRole
```

The `directorAppRole` role has the following members:

```
developer
developer_group
```

The relevant portions of the `jazn-data.xml` file specifying this hierarchy follows:

```
<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 of the five users and roles in the example.

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	NA	P1 and P3 (both inherited)
developer_group	NA	P1 and P3 (both inherited)

About the Role Category

A role category is a collection of application roles. A role category contains no hierarchical structure.

The following example 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 `<role-category>` but within `<extended-attributes>` of the corresponding application role. The role category name is case-insensitive. The role category can be managed with the `RoleCategoryManager` interface.

See also:

- Managing a Role Category in *Administering Oracle Entitlements Server*
- *Java API Reference for Oracle Platform Security Services*

About the Authenticated Role

OPSS supports the authenticated role. This role is available to all applications and need not be declared in the OPSS configuration fie. The permissions granted to the authenticated role follow from the enterprise groups and application roles of which it is a member, that is, it is granted inherited permissions only. A typical use of the authenticated role is to allow authenticated users access to common application resources. You configure this role in the Java servlet filter and the Enterprise JavaBeans (EJB) interceptor.

See also:

Configuring the Filter and the Interceptor

About the Anonymous User and Role

OPSS supports the anonymous user and the anonymous role. The permissions granted to them follow from the enterprise groups and application roles of which they are a member, that is, they are granted inherited permissions only. A typical use of the anonymous user and role is to allow unauthenticated users to access public, unprotected resources. You can configure this role in the Java servlet filter and the EJB interceptor.

See also:

Configuring the Filter and the Interceptor

About Administrative Users and Roles

A WebLogic Server administrator is any user member of the `Administrators` group, and any user in a WebLogic Server security realm can be added to this group. Generally, there is no default name for an administrator.

After a domain is configured, any member of the `Administrators` group can manage users in the `Administrators` group. The tools to manage these accounts are Oracle WebLogic Server Administration Console, WLST, and Fusion Middleware Control.

See also:

Install WebLogic Server in a Secure Manner in *Securing a Production Environment for Oracle WebLogic Server*

Users, Groups, And Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*

Add users to groups in *Oracle WebLogic Server Administration Console Online Help*

Using the WebLogic Scripting Tool in *Understanding the WebLogic Scripting Tool*

Managing User Accounts

For information about user accounts and passwords, see:

- Manage users and groups in *Oracle WebLogic Server Administration Console Online Help*
- Securing the WebLogic Server Host in *Securing a Production Environment for Oracle WebLogic Server*
- Managing the Embedded LDAP Server in *Administering Security for Oracle WebLogic Server*
- Configuring the Password Validation Provider in *Administering Security for Oracle WebLogic Server*

3

Understanding Identities, Policies, Credentials, Keys, Certificates, and Audit

Identities, policies, credentials, keys, and audit are fundamental to securing applications. This chapter introduces these basic concepts.
This chapter includes the following sections:

- Authentication Basics
- Policies Basics
- Credentials Basics
- Keys and Certificates Basics
- Audit Basics

Authentication Basics

OPSS uses WebLogic Server Authentication providers, components that validate users based on a user name/password combination or a certificate. Authentication providers make user identity information available (in subjects) to other components in the domain.

Java EE applications can use LDAP or DB authentication providers. Ready-to-use, Java SE applications use a file identity store, but the store can be configured to use an LDAP server or a database.

For information about Oracle WebLogic Server security, see Authentication in *Understanding Security for Oracle WebLogic Server*.

The following sections describe WebLogic Server Authentication providers:

- WebLogic Server Authentication Providers
- Identity Store Types and WebLogic Server Authentication Providers

WebLogic Server Authentication Providers

By default, users and groups are stored in the WebLogic Server Default Authenticator. This authentication provider uses `cn` as the default attribute.

The data stored in any LDAP authentication provider can be accessed by the User and Role API to query user profile attributes, but custom code may be required to query identity repositories that are not LDAP. Within an authentication provider, a group name must be unique.

The following sections explain how to set up multiple authentication providers:

- Support for Multiple Authentication Providers
- Additional Authentication Methods

> **See also:**
>
> How an LDAP X509 Identity Assertion Provider Works in *Administering Security for Oracle WebLogic Server*
>
> Configuring the SAML Authentication Provider in *Administering Security for Oracle WebLogic Server*

- Support for Multiple Authentication Providers
- Additional Authentication Methods

Support for Multiple Authentication Providers

WebLogic Server allows the configuration of multiple authentication providers in a given context. One of them must be an LDAP authentication provider.

The Default Authenticator has the control flag set (by default) to `REQUIRED`. To initialize the identity store with an LDAP authentication provider other than the Default Authenticator, change the control flag of the Default Authenticator or switch the order of the authentication providers.

OPSS initializes the identity store with the LDAP authentication provider according to the following algorithm:

1. Consider the set of LDAP authentication providers configured in the context. The context must contain at least one LDAP authentication provider.
2. Within that set, consider those that have set the maximum flag. The flag ordering used to compute this subset is the following:

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

3. Within that subset, consider the first provider in the context.

The LDAP authentication provider singled out in step 3 is the one used to initialize the identity store.

If a service instance initialization value is provided explicitly in the service instance configuration, then the value configured takes precedence over the default one.

> **See also:**
>
> *Securing a Production Environment for Oracle WebLogic Server*
>
> OPSS API References

Additional Authentication Methods

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

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 Server users.

> **See also:**
>
> Identity Assertion Providers and LoginModules in *Understanding Security for Oracle WebLogic Server*
>
> Overview of Single Sign-On with Microsoft Clients in *Administering Security for Oracle WebLogic Server*
>
> Creating a Kerberos Identification for WebLogic Server in *Administering Security for Oracle WebLogic Server*

Identity Store Types and WebLogic Server Authentication Providers

The information in this section applies only to Java EE applications.

For supported versions of identity store types, see Oracle Fusion Middleware Supported System Configurations.

Table 3-1 lists the WebLogic Server Authentication providers used with each identity store type:

Table 3-1 Identity Store Types and Authentication Providers

Store Type	WebLogic Server Authentication Provider
Oracle Internet Directory	OracleInternetDirectoryAuthenticator
Oracle Virtual Directory	OracleVirtualDirectoryAuthenticator
Open LDAP	OpenLDAPAuthenticator
Oracle Unified Directory	iPlanetAuthenticator
Oracle Directory Server Enterprise Edition	iPlanetAuthenticator
Sun Java System Directory Server	iPlanetAuthenticator
Novell eDirectory	NovellAuthenticator

Table 3-1 (Cont.) Identity Store Types and Authentication Providers

Store Type	WebLogic Server Authentication Provider
Tivoli Access Manager	OpenLDAPAuthenticator
Active Directory	ActiveDirectoryAuthenticator
Active Directory AM	ActiveDirectoryAuthenticator
Active Directory 2008	ActiveDirectoryAuthenticator
Oracle DB	CustomDBMSAuthenticator ReadOnlySQLAuthenticator SQLAuthenticator
Oracle Identity Cloud Service	OracleIdentityCloudIntegrator

Any LDAP authentication provider other than the Default Authenticator requires that you set the UseRetrievedUserNameAsPrincipal flag. Ready-to-use, this flag is set in the Default Authenticator.

See also:

Configuring the LDAP Identity Store in Java SE Applications

Using an OpenLDAP Identity Store

Authentication Providers in *Developing Security Providers for Oracle WebLogic Server*

Configuring Authentication Providers in *Administering Security for Oracle WebLogic Server*

Configure authentication and identity assertion providers in *Oracle WebLogic Server Administration Console Online Help*

Policies Basics

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

A Java Authorization and Authentication Services (JAAS) policy extends policies by allowing an optional list of principals, so that permissions are granted to code run by a user represented by those principals.

An application policy is a collection of JAAS policies that applies to the application only (in contrast to policies that apply to all applications in the domain).

A system policy grants permissions to users and groups, or to code, such as a URL or a JAR file.

The OPSS authorization service provides a central repository of system and application 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 the `web.xml` or `ejb-jar.xml` files) get mapped to enterprise users and groups and used by application policies.

The policy store refers to the portion of the security store where application and system policies are kept. The type of the policy store can be file, LDAP, or DB. A file store is an XML file. The only LDAP policy store type supported is Oracle Internet Directory.

During development and by default, application policies are specified in the `jazn-data.xml` file.

When you deploy the application with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control), policies can be automatically migrated into the security store.

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

See also:

Migrating the Security Store with Fusion Middleware Control

Reassociating the Security Store

The JAAS/OPSS Authorization Model

Credentials Basics

The OPSS credential service provides a central repository of artifacts 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.

The credential store refers to the portion of the security store where credentials are kept. The type of the credential store can be file, LDAP, or DB. A file credential store is the `cwallet.sso` file. The only LDAP credential store type supported is Oracle Internet Directory.

An application can use either the credential store or its own credential store. The credential store can be a file, LDAP, or DB.

When you deploy the application with Fusion Middleware Control, credentials can be automatically migrated into the security store.

OPSS provides the Credential Store Framework that includes an API that applications can use to create, read, update, and manage credentials programmatically.

See also:

Migrating the Security Store with Fusion Middleware Control

Reassociating the Security Store

Keys and Certificates Basics

The keystore service (KSS) keystore provides a central repository for the keys and certificates used by a domain components and applications. This eliminates the need to associate keystores with individual applications, and it provides a single user interface that allows you to manage keystore data uniformly in the domain.

The keystore repository can be file, DB, or LDAP. The keystore can be reassociated from one type to another.

See also:

Reassociating the Security Store

Managing Keys and Certificates

Audit Basics

Oracle Fusion Middleware Audit Framework provides the audit store, a central repository of audit records for the domain. Use this framework to audit events triggered by configuration changes as well as operational activity for components and deployed applications. The audit store can be file or database.

See also:

Introduction to Oracle Fusion Middleware Audit Framework

4

About the Security Store

This chapter introduces the security store types supported, the package requirements for application security, and the OPSS support for Federal Information Processing Standards (FIPS).
This chapter includes the following sections:

- Supported File, LDAP, and Database Stores
- Packaging Requirements
- FIPS Support in OPSS

Supported File, LDAP, and Database Stores

OPSS supports the following repositories:

- Security store and keystores:
  - Database-based
    - Oracle Database, Express Edition
    - Oracle Database, Standard Edition
    - Oracle Database, Standard Edition One
    - IBM DB2
    - Microsoft SQL Server
- Identity store-any LDAP authentication provider supported by Oracle WebLogic Server. File identity stores are supported in Java SE applications only.
- Audit store:
  - File-based - XML file
  - Database-based
    - Oracle Database, Express Edition
    - Oracle Database, Standard Edition
    - Oracle Database, Standard Edition One
    - IBM DB2
    - Microsoft SQL Server

For supported versions, see Oracle Fusion Middleware Supported System Configurations..

If you are using Oracle Internet Directory version 10.1.4.3 with OPSS, then the patch that fixes bug number 8351672 is required. For a list of patches to various versions of Oracle Internet Directory, see Using an LDAP Security Store.

> **See also:**
>
> Managing Policies
>
> Managing Credentials
>
> Managing Keys and Certificates
>
> Managing Audit

Packaging Requirements

Application policies are specified in the `jazn-data.xml` file, and application credentials are specified in the `cwallet.sso` file. Package these files in the `META-INF` directory of the application Enterprise ARchive (EAR) file. At application deployment, you typically migrate those policies and credentials to the security store.

> **See also:**
>
> Packaging a Java EE Application Manually
>
> Deploying Secure Applications

FIPS Support in OPSS

FIPS-140 is enabled in the entire Oracle Fusion Middleware stack. For information about FIPS, see FIPS-140 Support in Oracle Fusion Middleware in *Administering Oracle Fusion Middleware*.

Part II
Basic OPSS Administration

This part contains the following chapters:

5

Security Administration

This chapter describes the main tasks you carry out to manage application security and the tools you use to accomplish those tasks.
It contains the following sections:

- OPSS Administration: Main Steps
- Security Management Tools
- Security Practices with Fusion Middleware Control
- Security Practices with WebLogic Server Administration Console
- Security Practices with OES

OPSS Administration: Main Steps

Application security administration is an iterative process that incudes the following main tasks:

- Packaging and deploying applications
- Managing application roles and users
- Managing application and system policies
- Managing application credentials
- Managing application keys and certificates
- Managing audit

See also:

Deploying Secure Applications for information about packing security with an application

Managing Application Policies

Managing Application Roles

Managing Credentials

Managing Keys and Certificates

Managing Audit

Administration with Scripts and MBeans

Security Management Tools

To administer security, use any of the following tools:

- Oracle WebLogic Server Administration Console
- Fusion Middleware Control
- WebLogic Scripting Tool (WLST)
- Oracle Entitlements Server (OES)

The tool you use depends on the type of data and the kind of store.

OPSS does not support automatic backup or recovery of server files. It is recommended that all server configuration files be periodically backed up. For information about backup, see Introduction to Backup and Recovery in *Administering Oracle Fusion Middleware*.

Users and Groups

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

If your domain uses the Default Authenticator, then the Administration Server *must* be running for an application to access identity data with the User and Role API. Otherwise, if it uses an LDAP server different from the Default Authenticator, then use the utilities of that LDAP server to manage users and groups.

Policies, Credentials, Keys, and Certificates

Policies, keys, credentials, and certificates are stored in the same kind of storage (file, LDAP, or DB). The tools to manage these artifacts are:

- WebLogic Server Administration Console, for identities.
- Fusion Middleware Control, WLST, or OES, for policies and credentials.
- WLST, for keys and certificates.

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

> **See also:**
>
> Getting Started Managing Oracle Fusion Middleware in *Administering Oracle Fusion Middleware*

Security Practices with Fusion Middleware Control

This section addresses only OPSS security-related operations. For other security administrative operations, see WebLogic Server Security in *Administering Oracle WebLogic Server with Fusion Middleware Control*.

Use Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control):

- Post-installation and before you deploy the application to reassociate the security store.
- Post-installation and before you deploy the application to define OPSS properties.
- At application deployment, to configure the automatic migration of application policies and credentials to the security store.
- After application deployment, to:
  - Manage application policies.
  - Manage credentials.
  - Manage users and groups.
  - Specify the mapping from application roles to users, groups, and application roles.
- Manage system policies for the domain.
- Manage OPSS properties for the domain.

> **See also:**
>
> Deploying Oracle ADF Applications to a New Environment
>
> Reassociating the Security Store with Fusion Middleware Control
>
> Configuring Security Providers with Fusion Middleware Control
>
> Migrating the Security Store
>
> Managing the Policy Store
>
> Managing Application Roles
>
> Managing System Policies
>
> Managing Credentials

Security Practices with WebLogic Server Administration Console

Use WebLogic Server Administration Console to:

- Start and stop WebLogic servers.
- Configure WebLogic servers and domains.
- Deploy applications.
- Configure failover support.
- Configure WebLogic Server domains and WebLogic Server realms.
- Manage WebLogic Server Authentication Providers.
- Enable single sign-on in Microsoft clients, Web browsers, and HTTP clients.
- Manage administrative users and administrative policies.

See also:

Configuring Existing WebLogic Domains in *Understanding the WebLogic Scripting Tool*

Understanding WebLogic Server Deployment in *Deploying Applications to Oracle WebLogic Server*

Failover and Replication in a Cluster in *Administering Clusters for Oracle WebLogic Server*

Starting and Stopping Servers in *Administering Server Startup and Shutdown for Oracle WebLogic Server*

Secure servers and resources in *Oracle WebLogic Server Administration Console Online Help*

Security Practices with WLST

All security configuration tasks you do with WebLogic Server Administration Console, you can also do with WLST, including domain configuration and application deployment.

A Java Virtual Machine (JVM) instance points to at most one `jps-config.xml` file. All WLST commands called within the instance use the configuration file first obtained, regardless of the configuration location passed to subsequent commands.

See also:

WLST Command Reference for Infrastructure Security

Security Practices with OES

OES provides a large number of functions to configure and maintain authorization, including the ability to:

- Search application roles and the role hierarchy.
- Manage application policies and the role hierarchy.
- View the role hierarchy.
- Manage application role mappings.

For information about OES, see Introduction to Oracle Entitlements Server in *Administering Oracle Entitlements Server*.

6

Deploying Secure Applications

You typically deploy applications to Oracle WebLogic Server with Oracle WebLogic Server Administration Console or Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control). You can also start an application without restarting the server by setting its bits in a location known to WebLogic Server. This application start is known as *hot deployment*.
The recommendations in this chapter apply to Java EE applications using OPSS.

This chapter includes the following sections:

> **See also:**
>
> Deploying Applications in *Administering Oracle Fusion Middleware*
>
> *Developing Fusion Web Applications with Oracle Application Development Framework*
>
> Securing Oracle ADF Applications
>
> Configuring Java EE Applications to Use OPSS

- Developing Oracle ADF Applications
- Choosing the Tool for Deployment
- Deploying Oracle ADF Applications to a New Environment
- Deploying Standard Java EE Applications
- Deploying Audit-Aware Applications
- Migrating from a Test to a Production Environment

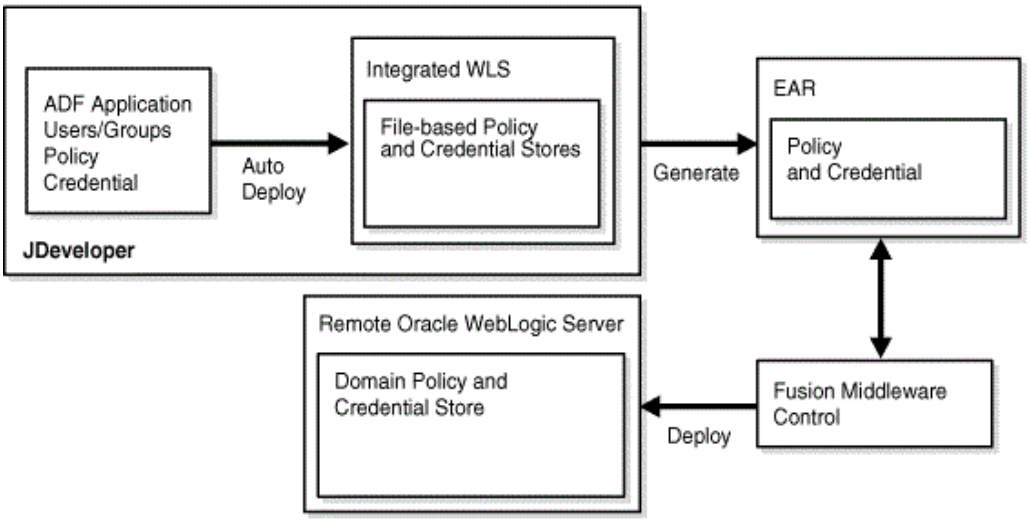
Developing Oracle ADF Applications

When developing an Oracle Application Development Framework (Oracle ADF) application you:

- Use JDeveloper to develop the application and Oracle ADF Wizard to configure security for the application.
- Use JDeveloper to copy the application users, roles, policies, and credentials to the integrated WebLogic Server, into which you deploy the application during the test cycles.
- Create the application Enterprise ARchive (EAR) file that packs application policies and credentials.
- Deploy the EAR file to WebLogic Server using Fusion Middleware Control.

Figure 6-1 illustrates the workflow for developing an Oracle ADF application.

Figure 6-1 WorkFlow for Developing an Oracle ADF Application



During development, you deploy your application with Oracle JDeveloper to the embedded WebLogic Server. After the application transitions to test or production environments, you deploy it with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control), WebLogic Server Administration Console, or by a 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.

> **See also:**
>
> Deploying Secure Applications with Fusion Middleware Control

Choosing the Tool for Deployment

Table 6-1 lists the tools you use to deploy Java EE applications.

Table 6-1 Tools to Deploy Java EE Applications

Java EE Application	Use
Non-Oracle ADF	WebLogic Server Administration Console, Fusion Middleware Control. The recommended tool is WebLogic Server Administration Console.
Oracle ADF	Fusion Middleware Control or WebLogic Scripting Tool (WLST). The recommended tool is Fusion Middleware Control.

See also:

Deploying Secure Applications with Fusion Middleware Control

- Deploying Secure Applications with Fusion Middleware Control

Deploying Secure Applications with Fusion Middleware Control

This section describes the security configurations available when you deploy a Java EE application that uses OPSS with Fusion Middleware Control.

The content displayed on the **Configure Application Security** page varies according to what is packaged in the EAR file:

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

The settings in this page control the migration of application policies and credentials (packed in application EAR file) to the security store.

- Migrating Application Policies at Deployment
- Migrating Application Credentials at Deployment

Migrating Application Policies at Deployment

To migrate the packed application policies at deployment, deploy the application to just one WebLogic Server. If you deploy an application with packed policies to multiple Managed Servers, then your deployment must include the Administration Server, so that the domain `system-jazn-data.xml` file is updated with the packed policies.

- If you are deploying the application for the first time, then you want to migrate application policies to the security store, so choose **Append** in the **Application Policy Migration** area.

  If for some reason you do not want the migration to take place, then choose **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, then choose **Append** to merge the packed policies with the ones in the domain, or **Ignore** to prevent policy migration.

  The option **Ignore** is chosen when redeploying an application or when you want to preserve changes made to the security store during previous deployments.
- If you choose **Append**, then specify which grants and roles should be migrated. The basic distinction is between Oracle ADF application roles and grants and development-time only roles and grants.

  To migrate Oracle ADF application roles and grants, and not to migrate development-time only security roles and grants, check **Migrate only application roles and grants. Ignore identity store artifacts**. Check it when you are deploying to a production environment. Note that you must map application roles to enterprise groups after the application deployment.
- When you choose **Append**, then specify a particular stripe (different from the default stripe, which is the application name) into which the application policies are migrated, by entering the name of that stripe in the text field **Application Stripe ID**.
- If nothing is specified, then the default settings are **Append** (in deployments) and **Ignore** (in redeployments).

Migrating Application Credentials at Deployment

- If you are deploying the application for the first time, then you want to migrate application credentials to the credential store. Therefore, choose **Append** in the **Application Credential Migration** area.
- Choose **Ignore** (default value) to prevent credential migration.

> **Note:**
>
> Application code using credentials may not work if you ignore credential migration. Choose **Ignore** when the credentials were created with the same map and key, but with different values.

- **Overwrite** is supported *only* when WebLogic Server is running in development mode.

Deploying Oracle ADF Applications to a New Environment

When an Oracle ADF application transitions to a test or production environment, you typically deploy it with Fusion Middleware Control to leverage all ADF security features that the framework offers.

The following sections explain the tasks involved when the application transitions to a new environment:

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

Deploying to a Test Environment

Before deploying an application that uses a file security store, verify that grants contains no duplicate permissions. If a duplicate permission (one that has the same name and class) appears in a grant, then the migration reports and error and halts. To resolve, edit the `jazn-data.xml` file and remove duplicated permissions.

When you deploy an Oracle ADF application with Fusion Middleware Control, the following processes take place:

- Application policies packed with the application are automatically migrated to the security store.
- Application credentials packed with the application are automatically migrated to the credential store.
- The bootstrap credentials needed to access LDAP repositories during migration are created.

Identities packed with the application are *not* migrated. You must configure an authentication provider (with WebLogic Server Administration Console), update identities, as appropriate, and map application roles to enterprise users and groups (with Fusion Middleware Control).

When you deploy to a domain with LDAP security stores and want to preserve application data integrity, Oracle recommends that you deploy the application at the cluster level or to just one Managed Server.

When you deploy an application to multiple Managed Servers, include the Administration Server so that data is migrated as expected.

> **See also:**
>
> Typical Administrative Tasks After Deployment

- Typical Administrative Tasks After Deployment

Typical Administrative Tasks After Deployment

After deploying the application, use Fusion Middleware Control or WebLogic Server Administration Console to:

- Manage security providers
- Create and customize application policies
- Create and customize application roles
- Manage system policies
- Manage credentials
- Manage keystores
- Manage audit data

When you undeploy an application with Fusion Middleware Control from a server running in production mode, the application policies are removed from the security store. If you use any other tool to undeploy the application, then those policies must be removed manually.

Credentials are *not* deleted when you undeploy the application.

See also:

- Configuring Security Providers with Fusion Middleware Control
- Managing Application Policies
- Managing Application Roles
- Managing System Policies
- Managing Credentials with Fusion Middleware Control
- Managing Keystores with Fusion Middleware Control
- Managing the Audit Store

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 WebLogic Server Administration Console or WLST commands, or programmatically, with deployment descriptors.

A Java EE application deployed to WebLogic Server is a WebLogic resource, so you set security for the application the same way that you do for any other WebLogic resource.

See also:

Securing Resources Using Roles and Policies for Oracle WebLogic Server

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

Oracle WebLogic Server Administration Console Online Help:

- Use roles and policies to secure resources

Securing WebLogic Web Services for Oracle WebLogic Server:

- Overview of Web Services Security

Developing Applications with the WebLogic Security Service:

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

Deploying Audit-Aware Applications

Audit-aware components refer to components integrated with Oracle Fusion Middleware Audit Framework, whose audit policies can be configured and whose events can be audited.

To use of this framework, you must register the application at deployment.

Registration

Configure audit registration parameters in the OPSS deployment descriptor file packaged with the application EAR file. Files are processed automatically by audit registration when you deploy the application.

Packaging Requirements

Package the following configuration files with the application EAR file:

- The event definitions file describing the auditable events for the application
- Translation files containing localizable elements

See also:

Migrating Audit Data

Creating Audit Definition Files

Registering the Application with the Audit Service

Introduction to Oracle Fusion Middleware Audit Framework

Migrating from a Test to a Production Environment

The recommendations that follow apply to Java EE applications that use Java Authorization and Authentication Services (JAAS) authorization, such as Oracle ADF, service-oriented architecture (SOA) applications, and Oracle WebCenter applications, and they do not apply to Java EE applications that use standard authorization.

The recommended tool to deploy applications is Fusion Middleware Control, and the user must have the appropriate permissions, including the permission to seed a schema in an LDAP repository.

File security stores are not recommended in production environments.

Migrating to a production environment includes:

- Migrating Identities
- Migrating Policies and Credentials
- Migrating Audit Data
- Migrating Keys and Certificates with migrateSecurityStore
- Migrating Identities
- Migrating Policies and Credentials
- Migrating Audit Data
- Migrating Keys and Certificates with migrateSecurityStore

Migrating Identities

The configuration of authentication providers in the test environment must be duplicated in the production environment. This task may include:

- Using WebLogic Server Administration Console to configure authentication providers and to provision users and groups.
- Setting in the production environment any particular provider configuration used in the test environment.
- Migrating Identities with migrateSecurityStore

Migrating Identities with migrateSecurityStore

You migrate identity data from a source repository to a target repository with the `migrateSecurityStore` WLST command, for example, when moving from a test environment that uses a file identity store to a production environment that uses an LDAP identity store.

The command does not require a connection to a running server to operate. The configuration file you pass to the `configFile` argument need not be an actual domain configuration file, but it only specifies the source and target repositories of the migration.

To migrate identities, use one of the following:

```
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"])
```

where:

- `configFile` specifies the relative or absolute path of the `jps-config.xml` configuration file.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument, where the source store is specified. The embedded LDAP server cannot be the source store.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument where the target store is specified. The target store must be LDAP. For list of supported types, see Identity Store Types and WebLogic Server Authentication Providers.
- `dstLdifFile` specifies the relative or absolute path to the LDAP Data Interchange format (LDIF) file created. Applies only when the target store is LDAP. Notice that the LDIF file is not imported into the LDAP server and requires manual editing.
- `overwrite` specifies whether to overwrite data in the target store. Set to `true` to overwrite target data. Set to `false` not to overwrite target data. Optional. If unspecified, it defaults to `false`.

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

The passwords in the generated LDIF file are all set to the string *weblogic*, and in case the target is LDAP, they are set to the string *change*. Before importing the LDIF file into the target LDAP store, change these passwords to real ones.

Migrating Policies and Credentials

In a production environment, it is strongly recommended that you reassociate the security store to an LDAP or DB store.

This section explains how to migrate policies and credentials when you deploy the application. For information about migrating security data after deployment, see Migrating Policies with migrateSecurityStore and Migrating Credentials with migrateSecurityStore.

If the application is hot deployed, then the migration of data in the `jazn-data.xml` file to the 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 redeployed, then any changes introduced in the `jazn-data.xml` file are *not* migrated to the security store.

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

To preserve GUIDs during migration, set the `jps.approle.preserveguid` parameter.

When you transition an application from a test to a production environment, it is critical that you know the answer to the following question:

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

Then, to deploy an application to a production environment:

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

     You can choose both **Append** and check **Migrate only application roles and grants. Ignore identity store artifacts**, even when the mapping of application roles have been modified in the test environment. Note that choosing this combination migrates application policies but disregards the maps to test enterprise groups. Later on, 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 deployment by choosing the option **Ignore** under the **Application Credential Migration** area in Fusion Middleware Control's page **Configuration Application Security**. Otherwise, choose **Append**.
2. Use the `migrateSecurityStore` command to migrate modified data:
   - If you chose to **Ignore** application policy migration, then migrate application and system policies from the test to the production LDAP. For information about the procedure, see the example in Migrating Policies with migrateSecurityStore.
   - If you chose to **Ignore** application credential migration, then migrate credentials from the test to the production LDAP. For information about the procedure, see the example in Migrating Credentials with migrateSecurityStore.
3. 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 it is necessary, then modify the production data, as appropriate.

- Migrating Policies with migrateSecurityStore
- Examples for Migrating Policies with migrateSecurityStore
- Migrating Credentials with migrateSecurityStore
- Examples for Migrating Credentials with migrateSecurityStore

Migrating Policies with migrateSecurityStore

By default, the `migrateSecurityStore` command re-creates GUIDs and it may take a long time to migrate a large number of policies. Therefore, when moving from a test to a production environment, consider migrating policies and credentials with an alternate procedure that uses bulk operations. For information about security store backup, see Backing Up and Recovering LDAP Security Stores.

To migrate policies with `migrateSecurityStore`, assemble a configuration file where the source and target are specified.

Here is a complete example of a configuration file, named `t2p-policies.xml`, illustrating the specification of policy sources in LDAP, database, and file repositories, and of policy targets in LDAP and DB repositories:

```
<?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 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 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 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 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>

 <!-- target LDAP policy store instance -->
 <serviceInstance provider="ldap.policystore.provider"
name="policystore.ldap.target">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your target LDAP directory structure; B. OID with OVD, if your
target LDAP is OVD; C. ldap://myDestHost.com:3060 with the URL and port number
of your target 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>

<!-- target DB policy store instance -->
 <serviceInstance provider="db.policystore.provider"
name="policystore.db.target">
<description>Replace: myDestDomain and myDestRootName to appropriate values
 according to your target 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 target
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 target LDAPs or DBs-->
 <serviceInstance location="./bootstrap" provider="credstoressp"
```

```
name="bootstrap.credstore">
  <description>Replace location with the full path of the directory where the
cwallet.sso file is located; typically found in targetDomain/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="LDAPtargetContext">
 <serviceInstanceRef ref="policystore.ldap.target"/>
 </jpsContext>

<jpsContext name="DBtargetContext">
 <serviceInstanceRef ref="policystore.db.target"/>
 </jpsContext>

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

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

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

where *`UserName`* and *`Password`* specify the user account name and password to access the target database.

Examples for Migrating Policies with migrateSecurityStore

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

- The `t2p-policies.xml` file is located on the target system in the directory where the command 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, then before using the command, restructure manually the system policy directory in the production environment to match the corresponding structure in the test environment.

To migrate policies from a test (or source) LDAP store to a production (or target) LDAP store, call `migrateSecurityStore` in the target system:

```
>migrateSecurityStore(type="policyStore",
                      configFile="t2p-policies.xml",
                      src="LDAPsourceContext",
                      dst="LDAPtargetContext")
```

To migrate policies from a test (or source) file store to a production (or target) LDAP store, call `migrateSecurityStore` in the target system:

```
>migrateSecurityStore(type="policyStore",
                      configFile="t2p-policies.xml",
                      src="XMLsourceContext",
                      dst="LDAPtargetContext")
```

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

```
>migrateSecurityStore(type="policyStore",
                      configFile="t2p-policies.xml",
                      src="DBsourceContext",
                      dst="DBtargetContext")
```

Migrating Credentials with migrateSecurityStore

The `migrateSecurityStore` command re-creates GUIDs and it may take a long time to migrate a large number of credentials. Therefore, when moving from a test to a production environment, consider migrating policies and credentials with an alternate procedure that uses bulk operations. For information about store backup, see Backing Up and Recovering LDAP Security Stores.

To migrate credentials with `migrateSecurityStore`, assemble a configuration file where the source and target are specified.

Here is a complete example of a configuration file, named `t2p-credentials.xml`, illustrating the specification of credential sources in LDAP, DB, and file repositories, and of credential targets in LDAP and DB repositories:

```
<?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 credential provider</description>
 </serviceProvider>

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

<serviceProvider
class="oracle.security.jps.internal.credstore.rdbms.DbmsCredentialStoreProvider"
```

```
name="db.credentialstore.provider" type="CREDENTIAL_STORE">
 <description>DB credential provider</description>
 </serviceProvider>

</serviceProviders>

<serviceInstances>
 <!-- Source file 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 source
credential store cwallet.sso is located in the source file system; typically located
in sourceDomain/config/fmwconfig/</description>
 </serviceInstance>

<!-- Source LDAP 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 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>

 <!-- target LDAP credential store instance -->
 <serviceInstance provider="ldap.credentialstore.provider"
name="credential.ldap.target">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your target LDAP directory structure; B. ldap://myDestHost.com:3060 with
the URL and port number of your target 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>

<!-- target DB credential store instance -->
 <serviceInstance provider="db.credentialstore.provider" name="credential.db.target">
<description>Replace: myDestDomain and myDestRootName to appropriate values according
to your target 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 target
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 target 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 targetDomain/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="LDAPtargetContext">
 <serviceInstanceRef ref="credential.ldap.target"/>
 </jpsContext>

<jpsContext name="DBtargetContext">
 <serviceInstanceRef ref="credential.db.target"/>
 </jpsContext>

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

For command syntax in this scenario, see Migrating One Credential Map with migrateSecurityStore Across Domains.

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

Examples for Migrating Credentials with migrateSecurityStore

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

To migrate credentials from a test (or source) LDAP store to a production (or target) LDAP store, call `migrateSecurityStore` in the target system:

```
>migrateSecurityStore(type="credStore",
                      configFile="t2p-credentials.xml",
                      src="LDAPsourceContext",
                      dst="LDAPtargetContext")
```

To migrate credentials from a test (or source) file store to a production (or target) LDAP store, call `migrateSecurityStore` in the target system:

```
>migrateSecurityStore(type="credStore",
                      configFile="t2p-credentials.xml",
                      src="FileSourceContext",
                      dst="LDAPtargetContext")
```

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

```
>migrateSecurityStore(type="credStore",
                      configFile="t2p-credentials.xml",
                      src="DBSourceContext",
                      dst="DBtargetContext")
```

Migrating Audit Data

Audit data consists of component event definitions, attribute table mappings, and audit policies, and this information resides in the audit store.

Use the `migrateSecurityStore` WLST command with the following syntax to migrate audit data between source and target repositories:

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

where:

- `configFile` specifies the absolute or relative location of a configuration file. This configuration file is created just for the migration and serves no other purpose. This file contains two contexts that specify the source and target stores.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. It is the source data store.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. It is the target data store.

- `overWrite` indicates whether to overwrite data in the target store. Set to `true` to always overwrite data. Set to `false` not to overwrite data unless specific conditions are met. Optional. If unspecified, it defaults to `false`. Note that:
  - System definitions are never overwritten regardless of the value of the flag.
  - If `overwrite` is `true`, then component definitions in the target store are replaced with the definitions in the source store.
  - If `overwrite` is `false`, then versions of the component definition in source and target store are compared. If they have the same major version and the minor version in the source component definition is higher, then the component definition in the target store is replaced with the definition in the source store. Otherwise, overwriting is skipped.

Migrating Keys and Certificates with migrateSecurityStore

Keys and certificates are migrated in two distinct scenarios:

- When source and target keystores lie in the same domain: the source and target keystores use the same encryption key.
- When source and target keystores lie in different domains: the source and target keystores use distinct encryption keys.

The following sections explain key migration in these scenarios:

Migrating Keys and Certificates in the Same 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 target service instances, and then use `migrateSecurityStore`. Note that a single configuration file is sufficient to specify source and target contexts when the keystores reside in the same domain.

The following example illustrates how to specify keystore sources in LDAP, DB, and file stores, and keystore targets in LDAP and 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 credential provider</description>
 </serviceProvider>
```

```
<!-- provider for file, LDAP, and DB keystores -->
<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 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 source keystore keystores.xml is located in the source file system; typically
located in sourceDomain/config/fmwconfig/</description>
 </serviceInstance>

<!-- Source LDAP keystore 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 keystore 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>

<!-- target LDAP keystore instance -->
 <serviceInstance provider="keystore.provider" name="keystore.ldap.target">
<description>Replace: A. myDestDomain and myDestRootName to appropriate values
according to your target LDAP directory structure; B. ldap://myDestHost.com:3060 with
the URL and port number of your target 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>

<!-- target DB keystore instance -->
```

```
 <serviceInstance provider="keystore.provider" name="keystore.db.target">
<description>Replace: myDestDomain and myDestRootName to appropriate values
according to your target 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 target
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 target 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 targetDomain/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="LDAPtargetContext">
 <serviceInstanceRef ref="keystore.ldap.target"/>
 </jpsContext>

<jpsContext name="DBtargetContext">
 <serviceInstanceRef ref="keystore.db.target"/>
 </jpsContext>

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

Note that because the migration involves LDAP or DB stores, the file includes the `bootstrap_credstore_context` context that specifies the location of the `cwallet.sso` bootstrap credential file.

Examples for Migrating Keys and Certificates in the Same Domain

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

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

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

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

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

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

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

Migrating Keys and Certificates across Different Domains

To migrate keystore data when source and target keystores are located in different domains, two different configuration files are required because the encryption keys used to encrypt the keystores are different. Therefore the bootstrap credential stores must be distinct.

When using the `migrateSecurityStore` WLST command, recall that:

- The context of the source keystore in the configuration file is specified in the `srcConfigFile` parameter.
- The context of the target keystore in the configuration file is specified in the `configFile` parameter.

Example

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

```
>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 contains the following chapters:

7

Life Cycle of Security Artifacts

This chapter explains the security artifacts that can be seeded in the security store when you create or extend a WebLogic Server domain. It also describes how to backup and recover security stores.

This chapter includes the following sections:

> **Note:**
>
> Procedures for upgrading security artifacts from earlier releases to 12.2.1.*x* and for upgrading a shared security store are described in Securing Datastores in *Planning an Upgrade of Oracle Fusion Middleware*.

- How Security Artifacts Are Seeded
- About Fusion Middleware Domains
- Creating Fusion Middleware Domains
- Layered Component Security Artifacts
- Backing Up and Recovering the Security Store

How Security Artifacts Are Seeded

In 11g domains, the specification of product-specific security artifacts is bundled in the application Enterprise ARchive (EAR) file and those artifacts are migrated to the security store at application deployment, provided the deployment descriptors are set appropriately.

In 12c domains, the specification of product-specific security artifacts is configured in a product template, which provides a way to seed those artifacts to the security store when the domain is created.

> **See also:**
>
> Layered Component Security Artifacts

About Fusion Middleware Domains

Fusion Middleware domains are created or extended with the Java Required Files (JRF) template. The template specifies the provisioning of artifacts in the security store. Specifically, the process creates:

- OPSS security artifacts.
- The `cwallet.sso` bootstrap file seeded with an encryption key.
- Keystores, including the truststore, a demonstration keystore, and a domain identity keystore used by Oracle WebLogic Server.
- A trust service wired to the truststore.
- Three data sources: one for the OPSS schema, one for the OPSS audit viewer schema, and one for the OPSS audit append schema.
- Configurations for database security and audit stores.

The JRF template does not create Managed Servers. In Fusion Middleware domains, all resources are targeted *only* to the Administration Server. If, at a later time, you add a Managed Server to the domain, then you must apply the JRF template to the target resources to that Managed Server also. To target OPSS and audit data sources to a Managed Servers after domain reconfiguration, use the `applyJRF` WebLogic Scripting Tool (WLST) command.

> **See also:**
>
> Layered Component Security Artifacts
>
> applyJRF in *WLST Command Reference for Infrastructure Components*

Creating Fusion Middleware Domains

Production Fusion Middleware domains require database security stores. The database can be a new database or a database associated with some other Fusion Middleware domain.

The following sections explain how to create Fusion Middleware domains:

- Using a New Database Instance
- Sharing a Database Instance
- Using a New Database Instance
- Sharing a Database Instance

Using a New Database Instance

To create or expand Fusion Middleware domains associated with a new database:

1. Create a new database schema. For information about database-based security stores, see Prerequisites to Using the Database Security Store.

2. Use the Fusion Middleware Configuration Wizard to create or expand a domain, as explained in Creating a WebLogic Domain in *Creating WebLogic Domains Using the Configuration Wizard*.

   This task includes supplying information about the database schema to use, such as the one created in step 1, and choosing to use the JRF Template. The database schema associated with the domain you create must be a new schema.

   When you use the JRF template, three data sources are automatically created: one for the OPSS schema, one for the OPSS audit viewer schema, and one for the OPSS audit append schema.

> **See also:**
>
> Template Tools in *Domain Template Reference*
>
> About Fusion Middleware Domains

Sharing a Database Instance

The following procedure uses WLST commands only. To create an expanded Fusion Middleware domain associated with a domain database:

1. Assuming that `domain1` uses the `db1` database (to which other domains want to join), use the `exportEncryptionKey` command to export the encrypt key from `domain1` to a specified location:

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

2. Create a script similar to the following that will create a domain that shares the `db1` database:

   ```
   ## 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('password')
   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/
LocalSvcTblDataSource')
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()
```

> **Note:**
>
> In the `setSharedSecretStoreWithPassword` command, use the same values for `location` and `password` that you used in the `exportEncryptionKey` command in step 1.

3. Start all the servers in `domain1` and in the new domain. Both domains now share the same security store.

> **Note:**
>
> - You can also use this script to create other Fusion Middleware domains like `domain1`. To do so, delete the lines starting with `setSharedSecretStoreWithPassword` from the script.
> - Be sure to backup the encryption key for the domain using the `exportEncryptionKey` WLST command and keep it in a secure location in case the domain fails and needs to be recreated.

> **See also:**
>
> exportEncryptionKey in *WLST Command Reference for Infrastructure Security*

Layered Component Security Artifacts

To streamline the seeding and processing of security artifacts, components consuming OPSS must 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 life cycle of security artifacts.	./security/component-security-info.xml
`jazn-data.xml`	Optional. Specifies 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 data.	./security/audit/component_events.xml
`component_events_xlf.jar`	Optional. Specifies the localized audit data.	./security/audit/component_events_xlf.jar

OPSS security artifacts bundled with a product template require the `component-security-info.xml` file that indicates how artifacts are handled.

Backing Up and Recovering the Security Store

This section describes how to back up and recover the security store.

This section contains the following topics:

- Configuration Files for Back Up
- Backing Up and Recovering a Database-Based Security Store
- Backing Up and Recovering LDAP Security Stores
- Recommendations

> **Note:**
>
> You can use the `migrateSecurityStore` WLST command to back up and recover security data. To back up security data, migrate security data to a file store. To recover it, migrate the file store to the target security store.

See also:

Administering Oracle Fusion Middleware

- Performing a Backup
- Recovering an Oracle WebLogic Server Domain

Migrating Identities

Migrating Policies and Credentials

Migrating Audit Data

Migrating Keys and Certificates with migrateSecurityStore

Configuration Files for Backup

In addition to backing up the security store, you must also back up the following configuration and data files:

```
DOMAIN_HOME/config/config.xml
DOMAIN_HOME/config/fmwconfig/jps-config.xml
DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml
DOMAIN_HOME/config/fmwconfig/cwallet.sso
DOMAIN_HOME/config/fmwconfig/keystores.cml
DOMAIN_HOME/config/fmwconfig/audit-store.xml
DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml
DOMAIN_HOME/config/fmwconfig/ids-config.xml
DOMAIN_HOME/config/fmwconfig/mbeans/jps_mbeans.xml
DOMAIN_HOME/config/fmwconfig/bootstrap/cwallet.sso
```

In these path names, *DOMAIN_HOME* represents the directory in which you created your domain. The default domain home location is *ORACLE_HOME*`/user_projects/domains/domain_name`.

Backing Up and Recovering a Database-Based Security Store

The procedure in this section uses Oracle Database Utility Recovery Manager (RMAN), a tool used to automate backup strategies and recoveries, and to duplicate databases.

Use the following procedure to back up a database-based security store on host A to a database-based security store on host B. The security store on host A has the `jdbc` URL set to `proddb`, and the security store in host B has the `jdbc` URL set to `testdb`. The procedure sets the domain to work with the cloned database-based security store on host B.

To back up the database-based security store:

1. Set up the `testdb` database on host B:
   a. Create the `inittestdb.ora` file to contain the following lines:

      ```
      #
      db_name=testdb
      #
      ```

   b. Add `testdb` to the `listener.ora` file:

      ```
      SID_LIST_LISTENER = (SID_LIST=(SID_DESC=(SID_NAME=testdb)
      (GLOBAL_DBNAME=testdb)(ORACLE_HOME=/u01/app/oracle/product/11.2.0/
      dbhome_1))
      ```

   c. Add `testdb/proddb` to the `tnsnames.ora` file:

      ```
      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)(PORT=YYYY))
      (CONNECT_DATA=(SERVER=DEDICATED)(SERVICE_NAME=testdb)))
      ```

   d. Restart the listener:

      ```
      lsnrctl stop, lsnrctl start
      ```

   e. Start the new instance using the `pfile` file in the `nomount` mode:

      ```
      $ export ORACLE_SID=testdb

      $ sqlplus / as sysdba

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

2. Use RMAN to clone the `proddb` database to the `testdb` database:
   a. Add add `testdb/proddb` to the `tnsnames.ora` file:

      ```
      proddb=(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=myhostA.com)(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)))
      ```

   b. Make sure that the `proddb` database is using the `spfile` file. If it is not, then generate a binary `spfile` from the `init` file by login in as the `sysdba` user and running `create spfile from pfile`. Then, restart the server.

   c. Restart the listener:

      ```
      lsnrctl stop, lsnrctl start
      ```

   d. Decide how to generate the names for the duplicate database files. Specifically, how to name the control files, data files, online redo log files, and temporary files.

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

      Run RMAN to clone the `proddb` database to the `testdb` database:

      ```
      $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 RMAN completes with no errors.

3. Verify that the `testdb` database works as expected by switching your domain to use the backed up database as the security store:
   a. Stop WebLogic Server.
   b. Change the `jdbc` URL from `proddb` to `testdb` in the `{domain}/config/fmwconfig/jps-config.xml`, `{domain}/config/fmwconfig/jps-config-jse.xml`, and `{domain}/config/jdbc/*xml` files.
   c. Restart WebLogic Server.
   d. Ensure that the domain security works as expected.

Backing Up and Recovering LDAP Security Stores

LDAP security store is not recommended for 12c release. It may be used in environments upgraded from 11g.

Use the procedure in this section to back up a source LDAP store to a target LDAP store.

1. In the source LDAP system create an LDAP Data Interchange format (LDIF) file by running `ldifwrite`:

   ```
   >ldifwrite connect="srcOidDbConnectStr" basedn="cn=jpsnode"
   ldiffile="srcOid.ldif"
   ```

   This command writes all entries under the `cn=jpsnode` node to the `srcOid.ldif` file.

   Move this file to the target LDAP system.

   The Oracle Internet Directory Database Password Utility is prompted to complete the ldifwrite command.

2. In the target LDAP system, ensure that the schema has been seeded.
3. In the target LDAP system, verify that there are no schema errors or bad entries by running `bulkload`:

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

   If duplicated distinguished names (common entries between the source and the target directories) are detected, then review them to prevent unexpected results.

   The Oracle Internet Directory Database Password Utility is prompted to complete the bulkload command.

4. Load data into the target LDAP, by running `bulkload`:

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

See also:

Administering Oracle Internet Directory:

- Dumping Data from Oracle Internet Directory to a File by Using ldifwrite
- Migrating LDAP Data Using LDIF File and Bulk Loader

Recommendations

Oracle recommends that you back up the security store and the configuration files periodically, and when any of the following events occurs:

- After first install.
- Before and after an upgrade.
- Any time you run the `rolloverEncryptionKey` WLST command.
- After you create new security data such as policies, credentials, or keys and keystores.

For more information on the configuration files, see Configuration Files for Backup.

8

Configuring the Identity Store

This chapter explains how to configure and use the identity store, and how to query it programmatically.
This chapter includes the following sections:

> **See also:**
>
> Overview of the Identity Directory API in *Developing Applications with Identity Governance Framework*

- About the Identity Store
- Configuring the Identity Store Provider
- Configuring the Identity Store
- Querying the Identity Store Programmatically
- Configuring SSL for the Identity Store

About the Identity Store

The identity store stores users and groups, and the service lets you query that data. By default, it supports querying a single LDAP identity store. You can configure the service to use a virtualized identity store that lets you query multiple LDAP identity repositories instead of just one. For information about identity virtualization, see Configuring the Identity Store .

Depending on the configuration, the service uses a file or (one or more) LDAP servers as the repository of identities. When the service is configured for LDAP, by default, it queries a single LDAP, but you can configure the service to query multiple LDAPs.

The service is available in Java SE environments. For information about virtualization in Java SE applications, see Configuring Virtualization in Java SE Applications.

Configuring the Identity Store Provider

Before using the identity store, you must configure the identity store provider. OPSS supports both file and LDAP identity store providers, as the following configuration example illustrates:

```
<serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
 class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
      <description>LDAP 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 you set Active Directory as the identity store provider, then set the `USERNAME_ATTR` and `USER_LOGIN_ATTR` properties to `sAMAccountName` in `jps-config.xml` (or `jps-config-jse.xml`) if you want to override the default value (`cn`). For example:

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

> **Note:**
>
> If you set `virtualize` to `true`, then do not set the `user.login.attr` and `username.attr` properties.

> **See also:**
>
> Task 3, Configuring the Identity Store Provider

Configuring the Identity Store

The following sections explain how to configure the identity store:

- Identity Store Parameters
- Understanding the Service Configuration
- Configuring Split Profiles
- Configuring Custom Authentication Providers
- Configuring Virtualization in Java SE Applications

> **See also:**
>
> OPSS System and Configuration Properties

- Identity Store Parameters
- Understanding the Service Configuration
- Configuring Split Profiles

- Configuring Custom Authentication Providers
- Configuring Virtualization in Java SE Applications

Identity Store Parameters

The following sections explain the use of the identity store configuration parameters:

- Query Parameters
- Global Connection Parameters
- Back-End Connection Parameters
- Query Parameters
- Global Connection Parameters
- Back-End Connection Parameters

Query Parameters

Use the following parameters to configure queries to multiple LDAPs:

- The `virtualize` property - This property can be either `true` (multiple LDAPs lookup) or `false` (single LDAP lookup). If unspecified, it defaults to `false`.
- Global Connection Parameters (when the `virtualize` property 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 is unspecified, then OPSS uses a default value.
- 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 default values.

Global Connection Parameters

Table 8-1 shows the global parameters. For a list of connection pool parameters, see Configuration Parameters for IDS in *Developing Applications with Identity Governance Framework*.

Table 8-1 Global LDAP Identity Store Parameters

Parameter	Default Value
`group.create.bases`	same as user.create.bases
`group.filter.object.classes`	groupofuniquenames
`group.mandatory.attrs`	No default value
`group.member.attrs`	uniquemember
`group.object.classes`	groupofuniquenames
`group.search.bases`	No default value
`group.selected.create.base`	No default value
`group.selected.search.base`	No default value
`groupname.attr`	cn

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

Parameter	Default Value
`max.search.filter.length`	No default value
`search.type`	No default value
`user.create.bases`	If only one authentication provider, then it uses the create base value. If multiple ones, then no default value is set.
`user.filter.object.classes`	inetorgperson
`user.login.attr`	uid
`user.mandatory.attrs`	No default value
`user.object.classes`	inetorgperson
`user.search.bases`	Same as group.search.bases
`username.attr`	cn

See also:

Table F-9

Back-End Connection Parameters

These parameters are specific to your particular LDAP store.

See also:

LDAP Identity Properties

Policy Store Service Properties

Understanding the Service Configuration

LDAP authentication providers are configured with Oracle WebLogic Server Administration Console or WebLogic Scripting Tool (WLST). At runtime, the server passes the configuration details to OPSS.

In WebLogic Server domains, you can configure multiple authentication providers in a given context. By default, the first authentication provider in the list is used to initialize the identity store. For information about authentication providers, see Support for Multiple Authentication Providers.

To query multiple LDAPs requires setting up the `virtualize` property.

The following sections explain several configurations:

- Configuring the Service for a Single LDAP

Configuring the Service for a Single LDAP

The following example illustrates the configuration of a single LDAP service instance:

```
<!-- 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>
```

Configuring the Service for Multiple LDAPs without Virtualization

In cases when the `virtualize` property cannot be set, configure the service to query more than one LDAP and override the configuration in WebLogic Server. To specify multiple LDAPs, use a comma separated list of LDAP URLs:

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

Configuring the Service for Multiple LDAPs with Fusion Middleware Control

To configure the service for multiple LDAPs with Fusion Middleware Control:

1. Choose the domain in the navigation pane on the left.
2. Go to Security, then Security Provider Configuration.
3. Expand the **Identity Store Provider** section of the page.
4. Click **Configure**.
5. The Identity Store Configuration page appears.
6. Under Custom Properties, click **Add**.
7. Add the new property:

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

Be sure to also add this property to the service instance in the default context of the OPSS configuration file.

8. Click **OK**.

Configuring the Service with WLST

To configure and use virtualization using WLST:

1. Create a script file to connect to the Administration Server in the domain of interest. You must specify the `userName`, `userPass`, `localHost`, and `portNumber` attributes for this operation. For information about configuring services with scripts, See Configuring Services with Scripts.
2. Go to `$ORACLE_HOME/common/bin`.
3. Run `wlst.sh`.

   For example, if the domain configuration file contains the `idstore.ldap` authentication provider, then the following command configures the provider for multiple LDAPs lookup:

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

Configuring the Timeout Setting with WLST

To set adapter timeout using WLST:

1. Run the listAdapters command to obtain the list of adapters.
2. Run the modifyLDAPAdapter command to set the timeout for each adapter to, for example, 120 seconds:

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

3. Restart WebLogic Server.

> **See also:**
>
> modifyLDAPAdapter in *WebLogic Scripting Tool Command Reference for Identity and Access Management*

Configuring Other Parameters

Optionally, update the configuration in the `jps-config.xml` file to set query parameters listed in Identity Store Parameters. These parameters are optional and have default values.

Restarting Servers

After configuring queries to multiple LDAPs, restart WebLogic Administration Server and Managed Servers.

Configuring Single and Multiple LDAPs

The following example illustrates the configuration of a single LDAP:

```
<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>
```

The following example illustrates the configuration of a multiple LDAPs:

```
<serviceProviders>
       <serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
   class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
           <description>LDAP IdentityStore Provider</description>
       </serviceProvider>
</serviceProviders>
<serviceInstances>
       <!-- IDstore instance connecting to multiple ldap  -->
       <serviceInstance name="idstore.virtualize" provider="idstore.ldap.provider">
       <!-- indicates using WLS ldap authentication providers -->
           <property name="idstore.config.provider"
  value="oracle.security.jps.wls.internal.idstore.WlsLdapIdStoreConfigProvider"/>
       <!-- enable virtualization -->
           <property name="virtualize" value="true"/>
   <!-- ldap properties (if not supplied, then it uses 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 `true`, and this allows queries to multiples LDAP directories.
- The `extendedProperty` element allows you to set front-end parameters to override default values.

> **See also:**
>
> Identity Store Parameters

Configuring Split Profiles

Identity virtualization supports split profiles, which lets applications access identity attributes when they are stored in more than one LDAP repository.

This feature requires additional configuration explained in Configuring Adapters for Identity Virtualization .

Configuring Custom Authentication Providers

OPSS supports WebLogic Authentication providers to access identities. If the available providers are not suitable to your particular LDAP server, then, typically, you customize one. This section explains how to configure and use a custom authentication provider.

When using a custom LDAP authentication provider, the following configuration illustrates how to specify the LDAP type so that the provider can find the proper LDAP plug-in by overriding `idstore.type` 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" />
    <serviceInstanceRef ref="myGenericLDAPName"/>
  </serviceInstance>
  <serviceInstance name="myGenericLDAPName" provider="idstore.ldap.provider">
    <!-- overrides the 'idstore.type' property -->
    <property name="idstore.type" value="ACTIVE_DIRECTORY" />
  </serviceInstance>
```

To override additional LDAP provider instances, insert similar entries. For information about provider configuration, see Configuring Security Providers with Fusion Middleware Control.

Configuring Virtualization in Java SE Applications

For Java SE applications, you set all configurations in the `jps-config-jse.xml` file. According to your needs, edit this file to:

1. Define a new service instance.
2. Add the new service instance to the context and replace any previously defined instances.
3. Enable `virtualize`.

> **See also:**
>
> Configuring the LDAP Identity Store in Java SE Applications

Querying the Identity Store Programmatically

To programmatically query the identity store, use OPSS APIs to obtain a context. This context acts like a bridge to obtain the store instance. Subsequently you use the User and Role API to query the store instance:

```
 try {
         //find the JPS context
         JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
         JpsContext ctx = ctxFactory.getContext();

         //find the JPS IdentityStore service instance
         //(assuming the back-end 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();
 } catch (Exception e) {
  e.printStackTrace()
}
```

> **See also:**
>
> Configuring Services with MBeans

Configuring SSL for the Identity Store

You can use Secure Sockets Layer (SSL) connections between the identity store and the LDAP server(s). Both the Identity Directory API and the User and Role API can operate with multiple LDAPs.

When the connection to the identity store originates at a client in WebLogic Server, then the SSL configuration is handled by the server.

See also:

Administering Oracle Fusion Middleware:

- Setting Up One-Way SSL to the LDAP Security Store
- Setting Up SSL in Identity Store Services

9

Configuring the Security Store

The security store is the central repository of system and application-specific policies, credentials, keys, and audit data used by all applications running in a WebLogic Server domain.
This chapter includes the following sections:

- About the Security Store
- Using an LDAP Security Store
- Using a Database-Based Security Store
- Reassociating the Security Store
- Migrating the Security Store
- Configuring Security Providers with Fusion Middleware Control

About the Security Store

The security store is the central repository of system and application-specific policies, credentials, and keys. This centralization facilitates the administration and maintenance of policies, credentials, and keys.

The type of the security store can be file, LDAP, or database. You can reassociate it from file to LDAP or database, from database to LDAP or database, or from LDAP to LDAP or database. Ready-to-use, the security store is a database store.

In Java EE applications, the security data is packaged with the application Enterprise ARchive (EAR) file, and it can be migrated to the security store when you deploy the application.

When a WebLogic Server domain uses policies from the security store, Java Authorization Contract for Containers (JACC) policies and the Java Security Manager become unavailable to all Managed Servers in that domain.

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.

> **See also:**
>
> Reassociating the Security Store with Fusion Middleware Control
>
> Reassociating the Security Store with reassociateSecurityStore
>
> Migrating the Security Store with Fusion Middleware Control
>
> Migrating the Security Store with migrateSecurityStore
>
> Java EE and WebLogic Security in *Understanding Security for Oracle WebLogic Server*

- Environments with Multiple Servers

Environments with Multiple Servers

Production WebLogic Server domains with several server instances (Administration and Managed Servers) on the same host or distributed across multiple machines, must use an LDAP or a database security store. File-based providers are not supported in production environments.

> **See also:**
>
> Policy Store Service Properties
>
> Credential Service Properties

Using an LDAP Security Store

Production environments typically use LDAP security stores. The only LDAP supported is Oracle Internet Directory.

OPSS does *not* support enabling referential integrity on LDAP 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 to:

1. Choose first **Administration**, then **Shared Properties**, and then **General**.
2. In the Enable Referential Integrity list, choose **Disabled**.

> **Note:**
>
> Depending on the version, the following Oracle Internet Directory patches are required:
>
> - Patch 10.1.4 to fix bug 9093298
> - Patch 11.1.x to fix bug 8736355
> - Patch 11.1.x and 10.1.4.3 to fix bug 8426457
> - Patch 10.1.4.3 to fix bug 8351672
> - Patch 10.1.4.3 to fix bug 8417224
> - Patch 11.1.1.6.0 to fix bug 13782459

For information about supported Oracle Internet Directory versions, see Oracle Fusion Middleware Supported System Configurations.

The following sections explain how to set up an LDAP security store:

- Prerequisites to Using the LDAP Security Store
- Resetting the LDAP User Password

> **See also:**
>
> Setting Up One-Way SSL to the LDAP Security Store in *Administering Oracle Fusion Middleware*
>
> Properties Common to All LDAP Servers

- Prerequisites to Using the LDAP Security Store
- Resetting the LDAP User Password

Prerequisites to Using the LDAP Security Store

To ensure the proper access to LDAP, set a node in the server directory as explained in this section.

When you use Fusion Middleware Control to reassociate to an LDAP store, the tool automatically provides bootstrap credentials in the `cwallet.sso` file.

To set a node in an LDAP server:

1. Create an LDAP Data Interchange format (LDIF) file (`jpstestnode.ldif`) specifying the following entries:

```
dn: cn=jpsroot
cn: jpsroot
objectclass: top
objectclass: OrclContainer
```

The distinguished name of the root node *jpsroot* must be distinct from any other distinguished name. Some LDAP servers enforce case-sensitivity by default. Multiple WebLogic Server domains can share a root node. This node need not be created at the top level, but the LDAP administrator must have read and write access to all nodes below it.

2. Import this data into the LDAP server with the `ldapadd` utility:

```
>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 with the `ldapsearch` utility:

```
>ldapsearch -h ldap_host -p ldap_port -D cn=orcladmin -w password -s base
-b "cn=jpsroot" objectclass="orclContainer"
```

4. Run `oidstats.sql` to generate database statistics for optimal database performance:

```
>$ORACLE_HOME/ldap/admin/oidstats.sql
```

You need to run this utility only once after the initial provisioning.

> **See also:**
>
> Specifying Bootstrap Credentials Manually
>
> About Oracle Internet Directory Database Statistics Collection Tool in *Reference for Oracle Identity Management*

Resetting the LDAP User Password

Use the procedure in this section to reset the LDAP user password.

1. Create an LDIF file with a content that specifies the new password:

```
dn: <UserDN>
changetype: modify
replace: userPassword
userPassword: new_password
```

where `userDN` stands for the distinguished name of the administrator.

2. Use `ldapmodify` to apply the specifications in the created file as in the following example, which uses specifications in the `updatePassword.ldif` file:

```
ldapmodify -h oid_hostName -p oid_port -D "cn=orcladmin" -w
orcladmin_password -f updatePassword.ldif
```

3. Run `modifyBootStrapCredential` to update the password in the bootstrap wallet.

See also:

- Setting the Server Mode by Using ldapmodify in *Administering Oracle Internet Directory*
- modifyBootStrapCredential in *WLST Command Reference for Infrastructure Security*

Using a Database-Based Security Store

A database-based security store is recommended in production environments. To configure the security store, use Fusion Middleware Control or WebLogic Scripting Tool (WLST). The database security store and the domain must be in the same data center.

For information about the database versions supported, see Oracle Fusion Middleware Supported System Configurations.

For information about the OPSS and audit schemas support of Edition-Based Redefinition (EBR), see Creating an Edition on the Server for Edition-Based Redefinition (Optional) in *Upgrading to the Oracle Fusion Middleware Infrastructure*.

The following sections explain how to set up a database security store:

- Prerequisites to Using the Database Security Store
- Maintaining a Database Security Store
- Resetting the OPSS Schema Password
- Setting Up an SSL Connection to the Database Security Store

Prerequisites to Using the Database Security Store

To use a database repository for the security store, first use Oracle Fusion Middleware Repository Creation Utility to create the required OPSS schema and to seed some initial data. See About the Repository Creation Utility in *Creating Schemas with the Repository Creation Utility*.

When using Repository Creation Utility to create the OPSS schema, choose all schemas whose names contain the following suffixes:

- `_OPSS`
- `_IAU`
- `_IAU_APPEND`
- `_IAU_VIEWER`
- `_STB`

Maintaining a Database Security Store

This section describes some of the tasks that you follow to maintain a database security store.

A database security store maintains a change log that should be periodically purged. To purge it, use the provided SQL `opss_purge_changelog.sql` script, which will purge change logs older than 24 hours, or connect to the database and run the `delete` utility (with the appropriate arguments):

```
SQL>delete from jps_changelog where createdate < (select(max(createdate) - 1)
from jps_changelog);
SQL>Commit;
```

To enhance performance when accessing a database security store, run the `DBMS_STATS` package to gather statistics about the physical storage of database tables and indexes. This information, stored in the data dictionary, is then used to optimize the execution plan for SQL statements accessing analyzed objects.

When loading large amount of data into a database security store, such as when creating thousands of new application roles, it is recommended that your run `DBMS_STATS` within short periods and concurrently with the loading activity. Otherwise, when the loading activity is small, then run `DBMS_STATS` just once or according to your needs.

The following example 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 database schema created with Repository Creation Utility.

Script Examples

The following example runs the `DBMS_STATS` command every 10 minutes:

```
#!/bin/sh
i=1
while [ $i -le 1000 ]
do
echo $i
sqlplus dev_opss/password@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 example also runs `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_invalidate=
>FALSE);',
interval => 'SYSDATE+(10/24/60)');
COMMIT;
END;
/
```

To stop the `DBMS_STATS` started by this 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 run a command like the following (which assumes that the query returned the job number 31):

```
EXEC DBMS_JOB.remove(31);
```

Resetting the OPSS Schema Password

To reset the OPSS schema password:

1. Use the database `ALTER USER` command to reset the password in the database. Remember the new password entered, as it will be used in the next two steps.
2. Use Oracle WebLogic Server Administration Console to update the password that the data source uses to connect to the OPSS schema with the new password.
3. Use the `modifyBootStrapCredential` WLST command to update the `cwallet.sso` bootstrap file with the new password.

> **See also:**
>
> Creating a JDBC Data Source in *Administering JDBC Data Sources for Oracle WebLogic Server*
>
> modifyBootStrapCredential in *WLST Command Reference for Infrastructure Security*

Setting Up an SSL Connection to the Database Security Store

Establishing a one- or two-way SSL connection to a database security store is optional and explained in section Configuring SSL for the Database in *Administering Oracle Fusion Middleware*.

Reassociating the Security Store

Reassociating the security store is the process that relocates security data from one repository to another one. The source type can be file, LDAP, or database. The target type can be LDAP or database.

Reassociation changes the repository while preserving the integrity of the data stored. This operation can take place at any time after the domain has been created, and it is carried out with either Fusion Middleware Control or the `reassociateSecurityStore` WLST command as explained in the following sections:

- Reassociating the Security Store with Fusion Middleware Control
- Reassociating the Security Store with reassociateSecurityStore
- Reassociating the Security Store with Fusion Middleware Control
- Reassociating the Security Store with reassociateSecurityStore

Reassociating the Security Store with Fusion Middleware Control

Reassociation migrates the security store (policies, credentials, keys, and audit data) from one repository to another and reconfigures security providers. For information about the procedure, see Task 2, Migrating the Security Store.

Note the following points:

- Before reassociating to a target LDAP store, ensure that your setup satisfies the Prerequisites to Using the LDAP Security Store.
- Before reassociating to a target database store, ensure that your setup satisfies the Prerequisites to Using the Database Security Store.
- Before reassociating and if a one-way SSL to a target LDAP is required, then follow the instructions in Setting Up One-Way SSL to the LDAP Security Store in *Administering Oracle Fusion Middleware*.
- After reassociating to an LDAP store, to secure access to the root node of the LDAP store, follow the instructions in Securing Access to LDAP Nodes.
- Reassociation updates the `jps-config.xml` and `jps-config-jse.xml` files with the new configuration: it deletes old provider configuration, inserts the new provider configuration, and moves data from the source to the target store.
- If the target store is LDAP, then the information is stored under the domain distinguished name according to the following format:

  ```
  cn=<domain_name>,cn=JpsContext,<JPS ROOT DN>
  ```

  If your configuration relies on the domain distinguished name, then do not delete this node from the LDAP Server.
- Securing Access to LDAP Nodes

Securing Access to LDAP Nodes

The procedure explained in this section is optional and performed only to enhance the security to access LDAP servers.

An access control list (ACL) is a list that specifies who can access information and what operations are allowed on the LDAP objects. The control list is specified at a node, and its restrictions apply to all entries under that node.

Use ACL to control the access to data stored in an LDAP repository. Typically, you specify this list at the root node of the store.

To specify an ACL at a node in the LDAP repository:

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 the root node of the store, and `userDN` stands for the distinguished name of the administrator (the same distinguished name that was entered to perform reassociation).

2. Use `ldapmodify` to apply these specifications to the Oracle Internet Directory:

```
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)
```

See also:

ldapmodify in *Reference for Oracle Identity Management*

Reassociating the Security Store with reassociateSecurityStore

The security store can be reassociated with the `reassociateSecurityStore` WLST command. For information about this command, see reassociateSecurityStore.

Migrating the Security Store

Applications can specify their own policies and these policies are stored in the application stripe (in the security store) when you deploy the application to WebLogic Server. Each application running in the domain uses one stripe, and more than one application can use the same stripe. In a file security store, stripes are specified in the `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` file under the element `<applications>`.

Migrating the security store is the process that relocates the policies, credentials, audit data, and keys from one repository to another one. The source type can be file, LDAP, or database. The target type can be LDAP or DB. The OPSS binaries and the target security store must have compatible versions. For information about version issues, see Incompatible Versions of Binaries and Security Store.

The following sections explain how to migrate application security to the security store:

- Migrating the Security Store with Fusion Middleware Control
- Migrating the Security Store with migrateSecurityStore
- Migrating the Security Store with Fusion Middleware Control
- Migrating the Security Store with migrateSecurityStore

Migrating the Security Store with Fusion Middleware Control

Applications can migrate security data specified in the `jazn-data.xml` application file to the security store when you deploy the application to WebLogic Server with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control). Policies can also be removed from the security store when the application is undeployed and updated when the application is redeployed.

Set the `jps.deployment.handler.disabled` system property to `true` to disable the migration of policies and credentials at deployment for *all* applications regardless of particular settings in the `weblogic-application.xml` files.

> **See also:**
>
> Task 2, Migrating the Security Store

Migrating the Security Store with migrateSecurityStore

You can migrate identities, policies, system policies, and credentials, from a source repository to a target repository with the `migrateSecurityStore` WLST command.

This command does not require a connection to a running server to operate. Therefore, the configuration file passed to the `configFile` argument need not be an actual domain configuration file, but assembled only to specify the source and target repositories of the migration.

> **Note:**
>
> The `migrateSecurityStore` command re-creates GUIDs and takes a long time to migrate a large volume of data. Consider using instead Oracle Internet Directory bulk operations to migrate large volume stores. For information about the procedure, see Backing Up and Recovering LDAP Security Stores.
>
> If migrating a large volume of data to an IBM DB2-based security store, you need to set the following configuration parameters on the DB2 database:
>
> - `update db cfg using MAXLOCKS AUTOMATIC`
> - `update db cfg using LOCKLIST AUTOMATIC`

The following sections explain how to use this command:

- Migrating All Policies with migrateSecurityStore
- Migrating System Policies with migrateSecurityStore
- Migrating Application Policies with migrateSecurityStore
- Migrating All Credentials with migrateSecurityStore in the Same Domain

- Migrating One Credential Map with migrateSecurityStore in the Same Domain
- Migrating All Credentials with migrateSecurityStore Across Domains
- Migrating One Credential Map with migrateSecurityStore Across Domains

> **See also:**
>
> Managing Policies with WLST
>
> migrateSecurityStore Usage Examples

- Migrating All Policies with migrateSecurityStore
- Migrating System Policies with migrateSecurityStore
- Migrating Application Policies with migrateSecurityStore
- Migrating All Credentials with migrateSecurityStore in the Same Domain
- Migrating One Credential Map with migrateSecurityStore in the Same Domain
- Migrating All Credentials with migrateSecurityStore Across Domains
- Migrating One Credential Map with migrateSecurityStore Across Domains
- Migrating Audit Data with migrateSecurityStore
- migrateSecurityStore Usage Examples

Migrating All Policies with migrateSecurityStore

To migrate *all* policies (system *and* application policies, for all applications) use one of the following syntaxes:

```
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"])
```

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain a contexts that specify:
  - The source store
  - The target store

  - The bootstrap credentials

    The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

    For information about extracting the keys used in a domain, see the `exportEncryptionKey` command in *WLST Command Reference for Infrastructure Security*.

    For information about storing a key into a wallet, see the `importEncryptionKey` command in *WLST Command Reference for Infrastructure Security*.

    For information about creating wallets, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.

- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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 when an incompatible artifact is detected. Optional. If unspecified, then it defaults to `false`.
- `overwrite` specifies whether to overwrite data in the target store. Set to `true` to overwrite target data. Set to `false` not to overwrite target data. Optional. If unspecified, then it defaults to `false`.

The contexts you specify in `src` and `dst` must be defined in the passed configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

Migrating System Policies with migrateSecurityStore

To migrate *just* system policies use one of the following syntaxes:

```
migrateSecurityStore.py -type globalPolicies
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-overwrite trueOrfalse]

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

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain contexts that specify:
  - The source store
  - The target store
  - The bootstrap credentials

  The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

  For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about creating wallets, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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, then it defaults to `false`.
- `overwrite` specifies whether to overwrite data in the target store. Set to `true` to overwrite target data. Set to `false` not to overwrite target data. Optional. If unspecified, then it defaults to `false`.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

Migrating Application Policies with migrateSecurityStore

To migrate *just* application-specific policies for an application, use one of the following syntaxes:

```
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")
```

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain contexts that specify:
  - The source store
  - The target store
  - The bootstrap credentials

  The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

  For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*

  For information about creating wallets, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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, then it defaults to `false`.
- `srcApp` specifies the name of the application whose policies are migrated.
- `dstApp` specifies the name of the application whose policies are being written. If unspecified, then it defaults to the name of the source application. Optional.
- `migrateIdStoreMapping` specifies whether enterprise policies should be migrated. The default value is `true`. To migrate just application policies, set it to `false`. Optional.
- `overWrite` specifies whether a target application policy stripe matching a source application policy stripe should be overwritten by or merged with the source application policy stripe.

Set to `true` to overwrite the target application policy stripe; set to `false` to merge the new policy artifacts in the source application policy stripe into the target application policy stripe. If unspecified, it defaults to `false`.

During merging, if a policy artifact with the same name exists in the target application stripe, then the migration of the policy artifact is skipped. Only the new resource actions added to a permission set, new members added to an application role, and new actions added to a resource type are merged into the target policy artifact.

- `mode` specifies whether the migration should stop and signal an error upon encountering a duplicate principal or a duplicate permission in a policy. Either do not specify or set to `lax` to allow the migration to continue when it encounters duplicate items, to migrate just one of the duplicated items, and to log a warning to this effect. Optional.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

If the input does not follow these syntax requirements, then the command execution fails. In particular, the input must satisfy the following requisites: (a) the `jps-config.xml` file is found in the passed location, (b) the `jps-config.xml` file includes the passed contexts, and (c) the source and the target context names are distinct.

Migrating All Credentials with migrateSecurityStore in the Same Domain

To migrate *all* credentials use one of the following syntaxes:

```
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"])
```

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain contexts that specify:
  - The source store
  - The target store
  - The bootstrap credentials

  The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

  For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

For information about creating wallets, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.

- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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 data in the target store. Set to `true` to overwrite target data. Set to `false` not to overwrite target data. Optional. If unspecified, then it defaults to `false`.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

Migrating One Credential Map with migrateSecurityStore in the Same Domain

To migrate *just* one credential map, use one of the following syntaxes:

```
migrateSecurityStore.py -type folderCred
                        -configFile jpsConfigFileLocation
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-srcFolder map1]
                        [-dstFolder map2]
                        [-overWrite trueOrFalse]
                        [-skip trueOrFalse]

migrateSecurityStore(type="folderCred",
                     configFile="jpsConfigFileLocation",
                     src="srcJpsContext",
                     dst="dstJpsContext",
                     [srcFolder="map1"],
                     [dstFolder="map2"],
                     [overWrite="trueOrFalse"],
                     [skip="trueOrFalse"])
```

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain contexts that specify:
  - The source store
  - The target store

  - The bootstrap credentials

    The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

    For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

    For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

    For information about creating a wallet, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.

- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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, then it defaults to `false`.
- `srcFolder` specifies the name of the map containing the credentials to migrate. Optional. If unspecified, then 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, then it defaults to the map passed to `srcFolder`.
- `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 unspecified, then it defaults to `false`. When `false` and if a matching is detected, then the source credential is disregarded and a warning is logged.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

Migrating All Credentials with migrateSecurityStore Across Domains

To migrate *all* credentials across domains, use one of the following syntaxes:

```
migrateSecurityStore.py -type credStore
                        -configFile "/target_domain/config/fmwconfig/jps-config.xml
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-skip trueOrfalse]
                        -srcConfigFile "/source_domain/config/fmwconfig/jps-config.xml
                        [-overwrite trueOrfalse]

migrateSecurityStore(type="credStore",
                     -configFile "/target_domain/config/fmwconfig/jps-config.xml,
                     src="srcJpsContext",
```

```
                    dst="dstJpsContext",
                    [skip="trueOrfalse"],
                                [srcConfigFile="alternConfigFileLocation"],
                    [overwrite="trueOrfalse"])
```

where:

- `configFile` refers to the configuration file in the destination domain where credentials are being migrated to. This configuration file should be specially assembled and must contain contexts that specify:
  - The target store
  - The bootstrap credentials

  The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

  For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about creating wallets, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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`.
- `srcConfigFile` refers to the configuration file on the source domain from where credentials are copied.
- `overwrite` specifies whether to overwrite data in the target store. Set to `true` to overwrite target data. Set to `false` not to overwrite target data. Optional. If unspecified, then it defaults to `false`.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration.

Migrating One Credential Map with migrateSecurityStore Across Domains

To migrate *just* one credential map, use one of the following syntaxes:

```
migrateSecurityStore.py -type folderCred
                        -configFile "/target_domain/config/fmwconfig/jps-
config.xml
```

```
                        -src srcJpsContext
                        -dst dstJpsContext
                        [-srcFolder map1]
                        [-dstFolder map2]
                        -srcConfigFile "/source_domain/config/fmwconfig/jps-config.xml
                        [-overWrite trueOrFalse]
                        [-skip trueOrFalse]

migrateSecurityStore(type="folderCred",
                     -configFile "/target_domain/config/fmwconfig/jps-config.xml,
                     src="srcJpsContext",
                     dst="dstJpsContext",
                     [srcFolder="map1"],
                     [dstFolder="map2"],
                     -srcConfigFile "/source_domain/config/fmwconfig/jps-config.xml,
                     [overWrite="trueOrFalse"],
                     [skip="trueOrFalse"])
```

where:

- `configFile` specifies the location of a configuration file relative to the directory where the command is run. This configuration file should be specially assembled and must contain contexts that specify:
  - The source store
  - The target store
  - The bootstrap credentials

  The bootstrap context specifies the location of the `cwallet.sso` file, which contains the keys needed to access the source and target stores, and to decrypt and encrypt security data.

  For information about extracting keys used by a domain, see exportEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about storing a key into a wallet, see importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

  For information about creating a wallet, see Common Wallet Operations in *Administering Oracle Fusion Middleware*.
- `src` specifies the name of a context in the configuration file passed to the `configFile` argument. The case of the string passed must match the case of the context in the configuration file.
- `dst` specifies the name of another context in the configuration file passed to the `configFile` argument. 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, then it defaults to `false`.
- `srcFolder` specifies the name of the map containing the credentials to migrate. Optional. If unspecified, then 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, then it defaults to the map passed to `srcFolder`.

- `srcConfigFile` refers to the configuration file on the source domain from where credentials are copied.
- `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 unspecified, then it defaults to `false`. When `false` and if a matching is detected, then the source credential is disregarded and a warning is logged.

The contexts you specify in `src` and `dst` must be defined in the configuration file, 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 command determines the locations of the source and the target repositories involved in the migration. For more examples, see Migrating Credentials with migrateSecurityStore.

Migrating Audit Data with migrateSecurityStore

Use the `migrateSecurityStore` WLST command to migrate audit data to a different security store. For information about the procedure, see Migrating Audit Data

migrateSecurityStore Usage Examples

For complete examples illustrating the use of `migrateSecurityStore`, see the following sections:

- Migrating Policies with migrateSecurityStore
- Migrating Credentials with migrateSecurityStore
- Migrating Audit Data
- Migrating Keys and Certificates with migrateSecurityStore

Configuring Security Providers with Fusion Middleware Control

Follow the instructions in this section to migrate the security store, to configure the identity store provider and security services, and to manage login modules and properties with Fusion Middleware Control.

- Task 1, Opening the Security Provider Configuration Page
- Task 2, Migrating the Security Store
- Task 3, Configuring the Identity Store Provider
- Task 4, Configuring Security Services
- Task 5, Managing Login Modules
- Task 6, Managing Properties and Property Sets

Task 1, Opening the Security Provider Configuration Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Security Provider Configuration.** The **Security Provider Configuration** page is displayed.

Task 2, Migrating the Security Store

1. Expand **Security Store Provider** and **Security Stores**.
2. Click **Change Store Type**. The **Configure Security Stores** page is displayed. In this page, enter the target repository parameters.
3. Click **OK**.

Task 3, Configuring the Identity Store Provider

1. Expand **Security Store Provider** and **Identity Store Provider**.
2. Click the **Configure** button. The **Identity Store Configuration** page is displayed. In this page, enter add or edit properties, as appropriate.
3. Click **OK**.

Task 4, Configuring Security Services

1. Expand **Security Store Provider** and **Security Services**.
2. Click a pencil icon to configure a provider. The provider's page is displayed.
3. In this page, enter the required fields.
4. Click **OK**.

Task 5, Managing Login Modules

1. Expand **Security Store Provider** and **Login Modules**. The table of configured login modules is displayed.
2. Click **Create** to create a new login module. The **Create Login Module** page is displayed. Enter the login module parameters and click **OK**.
3. Click **Edit** to modify a login module. The **Edit Login Module** page is displayed. Modify parameters and click **OK**.
4. Click **Delete** to remove a login module. Confirm deletion.

Task 6, Managing Properties and Property Sets

1. Expand **Security Store Provider** and **Advanced Properties**.
2. Click the **Configure** button. The Advanced Properties page is displayed.
3. In this page, do any of the following:
   - Click **Add** to add a new property.
   - Click **Edit** to modify a property.
   - Click **Delete** to removed a property.
   - Click **Add Property Set** to add a new property set.
   - Click **Add Property** to add a property to a set.
   - Click **Edit Property** to modify a property in a property set.
   - Click **Delete** to remove a property set or a property in a property set.

10

Managing Policies

This chapter explains how to manage policies with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control), WebLogic Scripting Tool (WLST), and Oracle Entitlements Server (OES).
This chapter includes the following sections:

- Determining the Security Store Characteristics
- Managing the Policy Store
- Managing Policies with Fusion Middleware Control
- Managing Policies with WLST
- Refreshing the Policy Cache
- Principals and Roles in WLST Commands
- Application Stripe in WLST Commands
- Managing Application Policies with OES

Determining the Security Store Characteristics

Use the `listSecurityStoreInfo` WLST command to determine several attributes of the security store. For information about this command, see listSecurityStoreInfo in *WLST Command Reference for Infrastructure Security*.

Managing the Policy Store

To avoid unexpected authorization failures and to manage policies effectively, note the following points:

- 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 data referencing a deleted user, then these artifacts are left dangling and, potentially, be inadvertently inherited if another user with the same name is created at a later time.

Similar considerations apply to when a user name is changed: all policies (grants, permissions, groups) referring to old data must be updated so that authorization works as expected with the changed data.

- When applied, policies use case-sensitivity in names. 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. Oracle recommends that:
  - When provisioning a policy, spell the names of users and groups used in the policy *exactly* as they are spelled in the identity store.
  - When entering a user name at runtime, enter a name that matches *exactly* the case of a name in the identity store.
- Resource type, resource, or entitlement names can contain printable characters only and they cannot start or end with a white space.
- Authorization failures are not shown in the console by default. To have authorization failures (such as `JpsAuth.checkPermission` failures) displayed in the console, set the `jps.auth.debug` system variable to `true`.

The following sections explain how to manage policies with Fusion Middleware Control, WLST, and OES. Typical operations include:

- Managing Policies with Fusion Middleware Control
- Managing Policies with WLST
- Managing Application Policies with OES

> **See also:**
>
> Failure to Get Permissions - Case Mismatch
>
> User Gets Unexpected Permissions
>
> Characters in Policies

Managing Policies with Fusion Middleware Control

Fusion Middleware Control allows you to manage system and application policies in a WebLogic Server domain as explained in the following sections:

- Managing Application Policies
- Managing Application Roles
- Managing System Policies

- Managing Application Policies
- Managing Application Roles
- Managing System Policies

Managing Application Policies

Follow the instructions in this section to manage application policies withFusion Middleware Control.

Task 1, Opening the Application Policies Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Application Policies**. The **Application Policies** page is displayed. The **Policy Store Provider** area is read-only and displays the provider currently used in the domain.

Task 2, Searching Application Policies

In the **Search** area, choose an application stripe, enter a string to match (a principal name, principal group, or application role), and click the search button. The results of the search are displayed in the table at the bottom of the page.

Task 3, Creating an Application Policy

Choose an application stripe, and click **Create**. The **Create Application Grant** page is displayed. In this page, add principals and permissions to the grant, as appropriate:

1. To add permissions, in the **Permissions** area click **Add** to display the **Add Permission** dialog.

   In the **Search** area of that dialog, first choose **Permissions** or **Resource Types**. If you chose Permissions, then identify permissions matching a class or resource name, and determine the **Permission Class** and **Resource Name**. If you chose Resource Types, 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 chose is displayed in the table in the **Permissions** area.

2. To add principals, click **Add** in the **Grantee** area to display the **Add Principal** dialog.

   In the **Search** area of that dialog, choose a **Type**, enter strings to match principal names and display names, and click the search button. The result of the query is displayed in the **Searched Principals** table. Then choose one or more rows from that table, and click **OK** to return to the **Create Application Grant** page. The principals you chose are displayed in the table in the **Grantee** area

3. Click **OK** to return to the **Application Policies** page. The new policy is displayed in the table at the bottom of the page.

Task 4, Creating an Application Policy Like Another One

1. Choose a policy.
2. Click **Create Like**. The **Create Application Grant Like** page is displayed and the table of permissions is filled in with the data extracted from the chosen policy.
3. Modify those values, as appropriate, and then click **OK**.

Managing Application Roles

Follow the instructions in this section to manage application roles withFusion Middleware Control.

Task 1, Opening the Application Roles Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Application Roles**. The **Application Roles** page is displayed. The **Policy Store Provider** area is read-only and displays the provider currently used in the domain.

Task 2, Searching Application Roles

In the **Search** area, choose an application stripe, enter a string to match, and click the search button. The results of the search are displayed in the table at the bottom of the page.

Task 3, Creating an Application Role

Click **Create** to display the **Create Application Role** page.

You need not enter data in this page all at one time. 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 specify the role mapping at a later time.

In the area **General**:

- In the **Role Name** text field enter the name of the role.
- In the **Display Name** text field, optionally, enter the name to display for the role.
- In the **Description** text field, optionally, enter a description of the role.
- In the **Members** area, specify the users, groups, or other application roles into which the role is mapped.

Task 4, Adding Application Roles to a Role

1. Choose a role and click **Add**. The **Add Principal** dialog is displayed.
2. Choose a **Type** (application role, group, or user), enter a string to match principal names, and click the search button. The result of the search is displayed in the **Searched Principals** table. Choose one or more principals from that table.
3. Choose one or more principals to which you want to add the role.
4. Click **OK** to return to the **Create Application Role** page. The new application role is displayed in the **Members** table.

Task 5, Creating an Application Role Like Another One

1. Choose a role.

2. Click **Create Like**. The **Create Application Role Like** page is displayed and some entries are filled in with data extracted from the role you chose.
3. Modify the list of roles and users, as appropriate, and then click **OK**.

To understand how permissions are inherited in the role hierarchy, see Permission Inheritance and the Role Hierarchy.

Managing System Policies

Follow the instructions in this section to manage system policies with Fusion Middleware Control.

- Task 1, Opening the System Policies Page
- Task 2, Searching System Policies,
- Task 3, Creating a System Policy

Task 1, Opening the System Policies Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **System Policies**. The **System Policies** page is displayed. The **Policy Store Provider** area is read-only and displays the provider currently used in the domain.

Task 2, Searching System Policies

In the **Search** area, choose a type, enter a string to match, and click the search button. The results of the search are displayed in the table at the bottom of the page.

Task 3, Creating a System Policy

1. Click **Create**. The **Create System Grant** page is displayed.
2. Choose type of policy to create: **Principal** or **Codebase**. The steps that follow assume you chose **Principal**.
3. To add permissions, click the **Add** button. The **Add Permission** dialog is displayed. Choose a permission to add to the policy being created.
   - 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.
   - Choose a permission to add. Details are rendered in the read-only **Customize** area.
   - Click **OK** to return to the **Create System Grant** page. The permission is displayed in the **Permissions** table.
4. Click **OK** to return to the System Policies page.
5. The table in the **Permissions for Codebase** area is read-only and it displays the resource name, actions, and permission class associated with the new system policy.

Managing Policies with WLST

An online WLST command is a command that requires a connection to a running server. Unless otherwise stated, commands listed in this section are online commands and operate on the security store, regardless of its type. There are a few commands that are offline which do not require a running server to operate.

Read-only commands can be performed only by users in the following WebLogic Server groups: `Monitor`, `Operator`, `Configurator`, or `Admin`. Read/write commands can be performed only by users in the following WebLogic Server groups: `Admin` or `Configurator`. All commands are available with the installation of Oracle WebLogic Server.

You can run WSLT commands in interactive or in script mode. In interactive mode, you enter the command at a command-line prompt. In script mode, you write the commands in a text file and run the script, much like the directives in a shell script.

All class names specified in commands must be fully qualified path names.

OPSS provides the following commands to administer application policies:

- 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 Migrating the Security Store with migrateSecurityStore

> **See also:**
>
> Security Practices with WLST
>
> Principals and Roles in WLST Commands
>
> Application Stripe in WLST Commands

- reassociateSecurityStore

reassociateSecurityStore

The `reassociateSecurityStore` WLST command migrates the security store from a source to a target store and resets service configurations in the `jps-config.xml` and `jps-config-jse.xml` files to the target repository. This command is supported in only the interactive mode.

The source store can be a file, LDAP, or DB security store. The target store can be a new store or an existing store in some other domain (see optional `join` argument below). When the target is a store in some other domain, you specify whether to append the source data to the target store (see optional `migrate` argument below).

The version of the source store must be equal to or greater than the version of the target store. If 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 data. If the check fails on some artifacts, then the command allows skipping the migration of incompatible artifacts by setting the `skip` argument to `true`. If this argument is not `true` and incompatible artifacts are detected, then the command terminates.

The command resets the bootstrap credentials (see `admin` and `password` arguments below). For an alternate way to reset bootstrap credentials, see the `modifyBootStrapCredential` and `addBootStrapCredential` commands.

Command Syntax

The command syntax varies according to the type of the target store. When the target is an LDAP 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 security 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 `join`, `migrate`, and `skip` arguments are next summarized:

- The `migrate` argument is relevant only when `join` is `true`. Otherwise it is ignored. Therefore, if migration is desired, then set both `join` and `migrate` to `true`.
- The `keyFilePath` and `keyFilePassword` arguments are required when `join` and `migrate` are both `true`.
- When the `join` and `migrate` arguments are both `true`, then if `skip` is `true`, then the migration of incompatible artifacts with the target store is skipped. If `skip` is `false`, then the command terminates when it finds any incompatible artifacts. Skipping is supported for generic credentials only.

The argument descriptions are:

- `domain`: specifies the name of the domain where the target store is located.
- `admin` in case of an LDAP target, specifies the administrator's user name on the target server. Use the format: `cn=usrName`.

  In case of a DB security store, it is required only when the database has a data source protected with user and password. In this case, this argument 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 in `admin`. It is required in case of an LDAP target.

  In case of a DB security store, it is required only when the database has a protected data source. In this case, it specifies the password associated with the user specified in `admin`. If specified, then `admin` must also be specified.
- `ldapurl` specifies the URI of the LDAP server. Use the format `ldap//host:port`, if you are using the default port, or `ldaps://host:port`, if you are using an anonymous Secure Sockets Layer (SSL) or one-way SSL. The secure port must be configured to handle the desired SSL connection mode, and must be distinct from the default (nonsecure) port.
- `servertype` specifies the kind of the target store. Valid types are `OID`, `DB_ORACLE`.
- `jpsroot` specifies the root node in the target LDAP repository under which all data is migrated. The format is `cn=nodeName`.

- `join` specifies whether the target store is a store in some other domain. Optional. Set to `true` to share a target store in some other domain. Set to `false` otherwise. If unspecified, it defaults to `false`. The use of this argument allows multiple WebLogic Server domains to point to the same security store, but note that:
  - Joining to a security store is supported only when you create a new domain.
  - Merging two distinct security stores in two domains is not supported.
  - If `join` is `true`, then you must export the OPSS encryption keys from one domain and import them into the other domain.

> **Note:**
>
> To export and import encryption keys use the following procedure. For an alternate procedure, see `keyFilePath` argument.
>
> Assume that Domain1 has a security store and Domain2 has reassociated to Domain1's security store with `join` set to `true`. Then:
>
> 1. Use the `exportEncryptionKey` WLST command to extract the key from Domain1 into the `ewallet.p12` file. The value of the `keyFilePassword` argument passed must be used later when you import that key into the second domain.
> 2. Use the `importEncryptionKey` WLST command to import the extracted `ewallet.p12` file into Domain2. The value of the `keyFilePassword` argument must be identical to the one used when the `ewallet.p12` file was generated.
> 3. Restart Domain2's server.
>
> For information about the export and import commands, see exportEncrytionKey and importEncryptionKey in *WLST Command Reference for Infrastructure Security*.

- `migrate` is meaningful only if `join` is `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, then it defaults to `false`.
- `skip` is meaningful only if both `join` and `migrate` are `true`, otherwise it is ignored. Specifies whether to skip the migration of incompatible artifacts. Set to `true` to skip appending incompatible artifacts to the target store and not to terminate the command. Set to `false` to terminate the command upon encountering an incompatible artifact in the source store. Optional. If unspecified, it defaults to `false`.
- `datasourcename` specifies the Java Naming and Directory Interface (JNDI) name of the Java Database Connectivity (JDBC) data source. The value 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 `ewallet.p12` file for the target domain is located. The content of this file is encrypted and secured by the value passed to `keyFilePassword`. It is required only if `join` is `true`.

If `join` is `true`, then the encryption keys must be exported from one domain and imported in the other. These tasks are carried out automatically when you use the `keyFilePath` and `keyFilePassword` arguments.

Assume that Domain1 has a security store and Domain2 reassociates to Domain1 security store with `join` set to `true` and key file arguments. Then first run the `reassociateSecurityStore` WLST command with the appropriate argument values, and then restart Domain2's server. For an alternate procedure to export and import encryption keys, see Note in the description of `join` argument.

- `keyFilePassword` specifies the password to secure the `ewallet.p12` file. Required only if `join` is `true`.
- `jdbcurl` specifies the JDBC URL used by a Java SE application to connect to the database. Applies only to Java SE applications. Required. Must be used with the `jdbcdriver`, `dbUser`, and `dbPassword`. arguments.
- `jdbcdriver` specifies the class of the JDBC driver used to connect to the database. Required. Must be used with the `jdbcurl` argument.
- `dbUser` specifies the database user (in the credential store) to add to the bootstrap credentials. Required. Must be used with the `jdbcurl` argument.
- `dbPassword` specifies the password of the user specified by `dbUser`. Required. Must be used with the `jdbcurl` argument.
- `odbcdsn` specifies the Open Database Connectivity (ODBC) data source name used by the C Credential Store Framework API. Applies only to C programs.

Reassociation Examples

The following example illustrates how to reassociate to a DB security store:

```
reassociateSecurityStore(domain="targetDomain", 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 security store in `otherDomain` *without* migrating the contents of the source security store:

```
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 security store in `otherDomain` and *to migrate* the contents of the source security store to the target DB security store skipping over incompatible artifacts:

```
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")
```

Refreshing the Policy Cache

This topic applies to LDAP and DB security stores only. In case of a file store, the cache is updated after a few seconds.

OPSS optimizes the authorization process by caching security artifacts. When a security artifact is modified, the change becomes effective at different times depending on where the tool used to modified the artifact and the application are running:

- If both the application and the tool are running on the same host and in the same domain, then the change becomes effective immediately.
- Otherwise, if the application and the tool are running on different hosts or in different domains, then the change becomes effective *after* the store cache is refreshed. The frequency of the cache refresh is determined by the value of the `oracle.security.jps.ldap.policystore.refresh.interval` property. The default value is 10 minutes.

  Within a domain, any changes introduced with WLST 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.
- Authorization Scenarios Using Policy Refreshing

Authorization Scenarios Using Policy Refreshing

The following use case illustrates the authorization behavior in scenarios when (from a different domain or host) OES is used to modify security data, 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.

Consider a scenario where:

1. A user logs in to the application.
2. The user accesses the functionality secured by the application role.
3. From another host (or domain), the enterprise role is removed from the application role.

Then consider the following actions and outcomes:

- The user logs out from the application, and *immediately* logs back in. The user can still access the functionality secured by the application role, because the policy cache has not yet been refreshed with the change introduced in step 3.
- The user logs out from the application, and logs back in *after 10 minutes*. The user is not able to access the functionality secured by the application role, because the policy cache has been refreshed with the change introduced in step 3.
- The user does not log out and remains able to access the functionality secured by the application role *for 10 minutes*, because the policy cache has not yet been refreshed with the change introduced in step 3.

- 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, because the policy cache has been refreshed with the change introduced in step 3.

Principals and Roles in WLST Commands

Several commands require that you specify the principal name and class for a role involved in the operation, such as the following which adds a principal to the `myAppRole` role in the `myApp` application stripe:

```
grantAppRole.py -appStripe myApp
                -appRoleName myAppRole
                -principalClass myPrincipalClass
                -principalName myPrincipal
```

When 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 name-class pairs:

- Authenticated role
  - `authenticated-role`
  - `oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl`
- Anonymous role
  - `anonymous-role`
  - `oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl`

The following WLST commands require principal name and class specification:

- `grantAppRole`
- `revokeAppRole`
- `grantPermission`
- `revokePermission`
- `listPermissions`

Application Stripe in WLST Commands

Several commands require that you specify an application stripe. If the application does not have a version, then the application stripe defaults to the application name. Otherwise, if the application has a version, then the application name and the application stripe are not identical.

For example, the name of the `myApp` application with version 1 is `myApp(v1.0)`, but the application stripe name is `myApp#v1.0`. More generally, an application with name `appName(vers)` gets assigned the application stripe `appName#vers`. Pass a string with this last pattern as the application stripe name:

```
>listAppRoles myApp#v1.0
```

The following WLST commands require stripe specification:

- `createAppRole`
- `deleteAppRole`
- `grantAppRole`
- `revokeAppRole`
- `listAppRoles`
- `listAppRoleMembers`
- `grantPermission`
- `revokePermission`
- `listPermissions`
- `deleteAppPolicies`

Managing Application Policies with OES

OES allows you to manage and search application policies and other security data in a WebLogic Server domain.

For information about managing policies with OES, see the following topics in *Administering Oracle Entitlements Server*:

- Querying Security Objects
- Managing Policies and Policy Objects

11

Managing Credentials

OPSS includes the Credential Store Framework, a collection of interfaces that you use to create, read, update, and manage credentials in your applications. Credentials are kept in the credential store, and the framework supports credential encryption.
This chapter includes the following sections:

- Credential Types
- Encrypting Credentials
- Managing Credentials with Fusion Middleware Control
- Managing Credentials with WLST

Credential Types

A credential can hold user names, passwords, and tickets, and 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.

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 the map name corresponds with the name of an application. All credentials with the same map name define a group of credentials, such as the credentials used by an application. The pair of map and key names must be unique for all entries in the credential store.

A password can have any number of characters, but they cannot be empty nor null.

By default, the credential store is an Oracle wallet and it can store X.509 certificates.

Encrypting Credentials

OPSS supports storing encrypted data in file and LDAP 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 is unique and has domain scope. To enable the encryption of credentials in a file or LDAP store, set the following property in the credential store instance of the `jps-config.xml` file:

```
<property name="encrypt" value="true" />
```

In case of DB credential stores, data is always encrypted using a client-side key.

The Domain Encryption Key

When you set the `encrypt` property to `true`, OPSS uses an encryption key to encrypt new credentials entered in the credential store. This encryption key is a 128-bit Advanced Encryption Standard (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 generated, the credential store data is not encrypted immediately with the new key. Instead, data is encrypted (with the new key) only when it is written. This means 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 facilitate this, OPSS provides offline scripts to export, import, and restore keys in the `cwallet.sso` bootstrap file so that an encryption key generated in one domain can be carried over to all other domains sharing the credential store.

The following scenarios illustrate how to set encryption key in a cluster of two domains, Domain1 and Domain2. (In case of more than two domains, treat each additional domain as Domain2).

> **Note:**
>
> The following scenarios assume an LDAP credential store but the use of the `importEncryptionKey` and `exportEncryptionKey` commands to import and export keys across domains applies also to DB credential stores (in which data is always encrypted).

Scenario One

Assume that Domain1 has reassociated to an LDAP credential store, and Domain2 has *not yet joined* to that store. Then, to enable credential encryption on that store:

1. Set the `encrypt` property to `true` in Domain1's `jps-config.xml` file and restart the domain.
2. Use the `exportEncryptionKey` WebLogic Scripting Tool (WLST) command to extract the key from Domain1 into the `ewallet.p12` file. Note that the value of `keyFilePassword` passed to the command must be used later when you import that key into another domain.
3. Set the `encrypt` property to `true` in Domain2's `jps-config.xml` file.

At this point, complete the procedure in one of two ways. Both of them use the `reassociateSecurityStore` WLST command but with different syntaxes.

The first approach:

1. Use the `reassociateSecurityStore` WLST command to reassociate Domain2's credential store to that used by Domain1. Use the `join` argument and *do not use* the `keyFilePassword` and `keyFilePath` arguments.
2. Use the `importEncryptionKey` WLST command to import the extracted `ewallet.p12` file into Domain2. Note that the value of `keyFilePassword` must be identical to the one used when the `ewallet.p12` file was generated.
3. Restart Domain2's server.

The second approach:

1. Use the `reassociateSecurityStore` WLST command to reassociate Domain2's credential store to that used by Domain1. Use the `join`, `keyFilePassword`, and `keyFilePath` arguments.
2. Restart Domain2's server.

Scenario Two

Assume that Domain1 has reassociated to an LDAP credential store and Domain2 has *already joined* to that store. Then, to enable credential encryption on that store:

1. Set the encrypt property to `true` in Domain1's `jps-config.xml` file and restart the domain.
2. Use the `exportEncryptionKey` WLST command to extract the key from Domain1 into the `ewallet.p12` file. Note that the value of `keyFilePassword` passed to the command must be used later when you import that key into another domain.
3. Set the property encrypt to `true` in Domain2's `jps-config.xml` file.
4. Use the `importEncryptionKey` WLST command to write the extracted `ewallet.p12` file into Domain2. Note that the value of `keyFilePassword` must be identical to the one used when the file `ewallet.p12` was generated.
5. Restart Domain2's server.

> **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, you must import that key to each of the other domains in the cluster with the `exportEncryptionKey` and `importEncryptionKey` commands.

> **See also:**
>
> Managing Credentials with WLST
>
> reassociateSecurityStore
>
> exportEncrytionKey and importEncryptionKey in *WLST Command Reference for Infrastructure Security*

Managing Credentials with Fusion Middleware Control

Follow the instructions in this section to manage credentials with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control).

- Task 1, Opening the Credentials Page
- Task 2, Searching Credentials
- Task 3, Creating a Credential Map
- Task 4, Adding a Key to a Credential Map
- Task 5, Editing a Key

Task 1, Opening the Credentials Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Credentials**. The **Credentials** page is displayed. The **Policy Store Provider** area is read-only and displays the provider currently used in the domain.

Task 2, Searching Credentials

To display credentials matching a given key name, enter the string to match in the **Credential Key Name** text field, and then click the search button. The result of the search is displayed in the table at the bottom of the page.

Task 3, Creating a Credential Map

1. Click **Create Map**. The **Create Map** dialog is displayed.
2. Enter the name of the map for the new credential.
3. Click **OK** to return to the **Credentials** page. The new credential map name is displayed in the table.

Task 4, Adding a Key to a Credential Map

1. Click **Create Key**. The **Create Key** dialog is displayed.
2. In the **Select Map** menu, choose a map, enter a key in the text **Key** text field, and choose a type (Password or Generic) from the pull-down menu **Type**.
   - For a Password key, enter the fields Key, User Name, Password, Confirm Passwords.
   - For a Generic key, enter the required field Key and the credential information either as text (choose the **Enter as Text** radio button), or as a list of key-value pairs (choose the **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.

3. Click **OK** to return to the **Credentials** page. The new key is displayed under the map you chose.

Task 5, Editing a Key

1. Choose a key.
2. Click **Edit**. The **Edit Key** dialog is displayed.
3. Modify the 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.
4. Click **OK** to save your changes and return to the **Credentials** page.

Managing Credentials with WLST

Executing an online command requires a connection to a running server. Unless otherwise stated, the commands listed in this section are online and operate on a security store.

Read-only scripts can be performed only by users in the following WebLogic Server groups: `Monitor`, `Operator`, `Configurator`, or `Admin`. Read/write scripts can be performed only by users in the following WebLogic Server groups: `Admin` or `Configurator`. All WLST commands are available with the installation of Oracle WebLogic Server.

You can run WSLT commands in interactive or in script mode. In interactive mode, you enter the command at a command-line prompt. In script mode, you write the commands in a text file and run the script, much like the directives in a shell script.

OPSS provides the following commands to administer application credentials:

- updateCred
- createCred
- deleteCred
- modifyBootStrapCredential
- addBootStrapCredential
- exportEncryptionKey
- importEncryptionKey
- restoreEncryptionKey
- rollOverEncryptionKey

See also:

Security Practices with WLST

Managing Policies with WLST

12

Managing Keys and Certificates

The keystore service allows you to manage and administer keys and certificates for Secure Sockets Layer (SSL), message security, encryption, and other tasks that require special certificates.
This chapter includes the following topics:

- About the Keystore Service
- About Keystore Service Commands
- Managing Keystores with Fusion Middleware Control
- Managing Keystores with WLST
- About Certificates
- Managing Certificates with Fusion Middleware Control
- Managing Certificates with WLST
- Replacing Demonstration CA Signed Certificates
- How Fusion Middleware Components Use the Keystore Service

About the Keystore Service

The Keystore Service allows you to manage keys and certificates for SSL, message security, encryption, and other tasks that require a key or a certificate. Typical keystore management tasks include the following:

- Creating a keystore in the context of an application stripe, directly or by importing a keystore file from the file system.
- Viewing the list of keystores and choosing some for updating.
- Updating and deleting keystores.
- Changing the keystore password.
- Exporting and importing keystores.

The following topics introduce Keystore Service concepts:

- Structure of the Keystore Service
- Types of Keystores
- The Truststore

Structure of the Keystore Service

A keystore is uniquely identified by an application stripe and a keystore within that stripe. Keys and certificates are created in keystores within stripes. Stripe names must be unique in the security store, and keystore names within a stripe must be unique in the stripe. For example, `(stripe1,keystoreA)`, `(stripe1,keystoreB)`, and `(stripe2,keystoreA)` refer to three distinct keystores.

Applications can create more than one keystore within the application stripe.

A keystore can contain the following entries, referenced by a unique alias within the keystore:

- Asymmetric Keys, including public keys and private keys that are used with SSL. Public keys are wrapped within a certificate.
- Symmetric Keys, generally used for encryption.
- Trusted Certificates, used to establish trust with an SSL peer.

Types of Keystores

The Keystore Service allows you to create two types of keystores:

- Keystores protected by a policy

  These keystores are protected by policies and any access to them by runtime code is protected by codesource policies. The key data is encrypted with the domain encryption key.

- Keystores protected by password

  These 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 is encrypted with the keystore/key password with password-based encryption.

Oracle recommends that you use password-protected keystores. However, if your application requires a high security level, then consider using a keystore protected by a codesource policy. You can export, import, and restore keys to a wallet.

In domains with multiple servers, the only supported store types are LDAP or DB. Do not use the keystore service to manage passwords or keys. Instead, use the credential store for your application.

The Truststore

The truststore is a keystore that contains trusted certificates of most well-known third-party certificate authorities and a trusted certificate from the demonstration certification authority (CA), which is configured with the Keystore Service. If your application uses SSL, for example, it can point to the truststore for certificates, and you do not need a dedicated keystore to store them.

> **Caution:**
>
> The demonstration CA includes a hard-coded private key. Oracle recommends that you neither use nor trust the demonstration CA certificates in production environments. For more information about replacing the demo CA certificates, see the following topics:
>
> - Replacing Demo CA Certificates With Domain CA Signed Certificates
> - Replacing Demo CA Certificates With Third-Party CA Signed Certificates
> - Replacing the Demo CA Trust Service Certificate

The truststore is shared by all products and applications in a domain. The decision to add or remove trust for a product may affect other products in the domain. Consider creating a custom truststore only if your product's trust management requirements are not met by the truststore.

A truststore is preconfigured for all products and applications to use, and applications can configure multiple keystores, according to their needs.

One-Way SSL,

For one-way SSL, applications can use the truststore and you do not need to create a specific keystore.

Two-Way SSL

For two-way SSL, applications create a keystore to keep just their identity certificate and use the truststore for other certificates.

About Keystore Service Commands

The Keystore Service uses a dedicated set of 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 to all these commands is the `getOpssService` command, which gets an OPSS service command object that lets you:

- Execute commands for the service
- Obtain command help

The command syntax is:

```
variable = getOpssService(name='service_name')
```

In this command:

- `variable` stores the command object.
- *`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')
```

- Getting Help for Keystore Service Commands
- Keystore Service Command Reference

Getting Help for Keystore Service Commands

To obtain help for any Keystore Service command, start by obtaining a service command object. Then use this object in conjunction with the help command and the command in question.

To obtain the service command object and the list of all Keystore Service commands, enter:

```
svc = getOpssService(name='KeyStoreService')
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')
```

Keystore Service Command Reference

For syntax and reference information about the Keystore Service commands, see OPSS Keystore Service Commands in the *WLST Command Reference for Infrastructure Security*.

Managing Keystores with Fusion Middleware Control

Use the following tasks to manage keystores with Oracle Enterprise Manager Fusion Middleware Control.

- Task 1, Opening the Keystore Page
- Task 2, Creating a Keystore
- Task 3, Deleting a Keystore
- Task 4, Changing a Keystore Password

Task 1, Opening the Keystore Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.

Task 2, Creating a Keystore

1. Choose the stripe in which to create the keystore. If necessary, then create a stripe.
2. Click **Create Keystore**. The **Create Keystore** dialog is displayed.
3. In this dialog, enter the following data:
   - Keystore Name: a unique name.
   - Protection Type: the protection mechanism for the keystore. Choose Policy or Password. For a password-protected keystore, provide a valid password.
   - Grant Permission: check this box to grant permissions to code URL.
4. Click **OK**. The new keystore is displayed under the stripe you chose.

Task 3, Deleting a Keystore

When you delete a keystore, note that all certificates in it are also deleted.

1. Expand the stripe in which the keystore resides, and choose a row.
2. Click **Delete**. The **Delete Keystore** dialog is displayed.
3. If this is a password-protected keystore, then enter the password.
4. Click **OK**.

Task 4, Changing a Keystore Password

This task applies to password-protected keystores only.

1. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore.
2. Click **Change Password**. The **Change Keystore Password** dialog is displayed.
3. Enter the old and new passwords.
4. Click **OK**.

Managing Keystores with WLST

Use the following tasks to manage keystores with WebLogic Scripting Tool (WLST).

Task 1, Creating a Keystore

Use the `createKeyStore` WLST command. For example, assuming the stripe name is `teststripe1`, create a permission-based keystore:

```
svc.createKeyStore(appStripe='teststripe1', name='keystore1',
password='password',permission=true)
```

where *password* is the password for keystore1. Any combination of characters is allowed for a new stripe name, but it is recommended that you do not use the forward slash (/) in it. The keystore name must unique.

Task 2, Deleting a Keystore

Use the deleteKeyStore WLST command. For example, assuming the stripe is appstripe1, delete keystore1:

```
svc.deleteKeyStore(appStripe='appstripe1', name='keystore1', password='password')
```

where *password* is the password for keystore1.

Task 3, Changing a Keystore Password

Use the changeKeyStorePassword WLST command. For example, assuming the stripe name is system, change the password of keystore2:

```
svc.changeKeyStorePassword(appStripe='system', name='keystore2',
currentpassword='currentpassword', newpassword='newpassword')
```

where *currentpassword* and *newpassword* are the old and new passwords.

Task 4, Exporting a Keystore

Use the exportKeyStore WLST command.

To export a singe key to a file:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myorakey', keypasswords='keypassword1',
type='JKS',filepath='/tmp/file.jks')
```

To export multiple keys to a file, specify a comma-separated list of aliases and key passwords.

To export a symmetric key:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myorakey', keypasswords='keypassword1',
type='JCEKS',filepath='/tmp/file.jks')
```

To export to a wallet, use the OracleWallet type:

```
svc.exportKeyStore(appStripe='mystripe', name='keystore3',
password='password',aliases='myorakey1,myorakey2', keypasswords='',
type='OracleWallet',filepath='/tmp')
```

Task 5, Importing a Keystore

Use the importKeyStore WLST command.

To import a single key, such as myOrakey:

```
svc.importKeyStore(appStripe='mystripe', name='keystore2',
password='password',aliases='myOrakey', keypasswords='keypassword1', type='JKS',
permission=true, filepath='/tmp/file.jks')
```

To import multiple keys, specify a comma-separated list of aliases and key passwords.

To import keys from a wallet, use the OracleWallet type:

```
svc.importKeyStore(appStripe='mystripe', name='keystore4',
password='password',aliases='myorakey1,myorakey2', keypasswords='',
type='OracleWallet', permission=true, filepath='/tmp')
```

About Certificates

The Keystore Service (KSS) keystore supports the Java Keystore (JKS), Java Cryptography Extension Keystore (JCEKS), and Oracle wallet certificate formats. Typical certificate management tasks include the following:

- Creating a certificate for a key pair.
- Generating a Certificate Signing Request (CSR) for the certificate and saving it to a file.
- Sending the CSR to a certificate authority who verifies the sender, and signs and returns the certificate.
- Importing user and trusted certificates into the keystore, by either pasting it into a text field or importing it from the file system.

> **Note:**
>
> Keystore Service supports importing PEM/BASE64-encoded certificates only. You cannot import DER-encoded certificates or trusted certificates into a keystore.

- Exporting certificates or trusted certificates from the keystore to a file.
- Deleting certificates or trusted certificates from the keystore.

The following points regarding public CA certificates apply to domains upgraded to 12.2.1 and to new 12.2.1 Java Required Files (JRF) domains:

- Well-known public CA certificates are no longer available in the `trust` keystore in the `system` stripe.
- Use instead the `publiccacerts` keystore in the `system` stripe, which has been previously seeded with well-known public CA certificates from the Java SE Development Kit (JDK) `cacerts` file. Alternatively, import your own certificates as needed.
- The `merge.jdkcacerts.with.trust` property specifies whether to return public CA certificates in the `kss://system/ubliccacerts` keystore when you query the `kss://system/trust` keystore. Set to `true`, to have all `publicacerts` certificates returned with the query. Do not set or set to `false`, to have no `publicacerts` certificates returned with the query.

> **See also:**
>
> Trust Service Properties

Managing Certificates with Fusion Middleware Control

Use the following tasks to manage keystores with Fusion Middleware Control.

Task 1, Generating a Key Pair

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.
2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the keystore password and click **OK**. The **Manage Certificates** page is displayed.
4. Click **Generate Keypair**. The Generate Keypair dialog is displayed.
5. In this dialog, enter the following data:
   - Alias (required)
   - Common Name (required)
   - Subject Alternative Name
   - Organizational Unit
   - Organization
   - City
   - State
   - Country: Choose one from the drop-down list.
   - Key Type: Choose an algorithm from the drop-down list. The choices are Elliptic Curve Cryptography (ECC) or RSA.
   - Key Size: Choose a key size.
6. Click **OK**. The new certificate is displayed in the certificate list.
7. View the certificate details by clicking on the certificate alias.

The key pair is wrapped in a demonstration CA signed certificate and stored in the truststore. To use this certificate for SSL, applications must either use the truststore or import the demonstration CA certificate to a custom keystore.

Task 2, Generating a CSR for a Certificate

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.

2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the keystore password and click **OK**. The **Manage Certificates** page is displayed.
4. Choose the row corresponding to the certificate and click **Generate CSR**. The Generate CSR dialog appears
5. Do one of the following:
   - Copy and paste the entire CSR into a text file, and click **Close**.
   - Click **Export CSR** to save the CSR to a file.

Send the generated certificate to a certificate authority which will return a signed certificate.

Task 3, Importing a Certificate

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.
2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the password and click **OK**. The **Manage Certificates** page is displayed.
4. Click **Import**. The **Import Certificate** dialog is displayed.
5. Choose **Certificate** or **Trusted Certificate** from the drop-down.
6. Choose the alias from the drop-down.
7. Specify the certificate source. If using the **Paste** option, then copy and paste the certificate directly into the text field. If using the **Select a file** option, then click **Browse** to choose the file from the operating system.
8. Click **OK**. The imported certificate or trusted certificate is displayed in the list of certificates.

Task 4, Exporting a Certificate

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, that then to **Keystore**. The **Keystore** page is displayed.
2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the password and click **OK**. The **Manage Certificates** page is displayed.
4. Choose the row corresponding to the certificate and click **Export**. The certificate export dialog is displayed.
5. Do one of the following:
   - Copy and paste the entire certificate into a text file, and click **Close**.
   - Click **Export Certificate** to save the certificate to a file.

Task 5, Changing a Certificate Password

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.

2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the password and click **OK**. The **Manage Certificates** page is displayed.
4. Choose the row corresponding to the certificate and click **Change Password**. The Change Key Password dialog appears
5. Enter the old and new passwords and click **OK**.

Task 6, Deleting a Certificate

1. Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Keystore**. The **Keystore** page is displayed.
2. Expand the stripe in which the keystore resides. Choose the row corresponding to the keystore, and click **Manage**.
3. If the keystore is password-protected, then enter the password and click **OK**. The **Manage Certificates** page is displayed.
4. Choose the row corresponding to the certificate and click **Delete**. The Delete Certificate dialog is displayed. Click **OK**.

Managing Certificates with WLST

Use the following tasks to manage certificates with WLST.

Task 1, Generating a Key Pair

Use the `generateKeyPair` WLST command. For example, assuming an application stripe named `appstripe2`, the following command creates a key pair with the `myalias` alias using the ECC algorithm:

```
svc.generateKeyPair(appStripe='appstripe2', name='keystore2',
password='password', dn='cn=www.example.com', keysize='1024',
alias='myalias', keypassword='keypassword', algorithm='EC')
```

where *`password`* is the keystore password and *`keypassword`* is the password of the alias.

The key pair is wrapped in a demonstration CA certificate and stored in the truststore. If your application is not using the truststore, then you must import the demonstration CA certificate to a custom keystore.

The following example generates a keypair in keystore2 using the default RSA algorithm:

```
svc.generateKeyPair(appStripe='system', name='keystore2',
password='password', dn='cn=www.oracle.com', keysize='2048', alias='orakey',
keypassword='keypassword')
```

You can pass a Subject Alternative Name (SAN) extension using the optional `ext_san` argument. The format for the argument is `"type:value,...,type:value"`. Only the DNS type is supported.

The following example generates a keypair with a SAN extension in keystore2 using the default RSA algorithm:

```
svc.generateKeyPair(appStripe='system', name='keystore2',
password='password', dn='cn=www.oracle.com', keysize='2048', alias='orakey',
keypassword='keypassword',
ext_san='DNS:server1.oracle.com,DNS:www.oracle.com')
```

Task 2, Generating a CSR for a Key Pair

Use the `exportKeyStoreCertificateRequest` WLST command. For example, assuming an application stripe is `stripe1`, the following command generates a CSR from the `testalias` key pair:

```
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.

Task 3, Importing a Certificate

Use the `importKeyStoreCertificate` WLST command. For example, assuming the `appstripe1` application stripe, the following command imports a certificate with `mykey` alias 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.

Task 4, Exporting a Certificate

Use the `exportKeyStoreCertificate` WLST command. For example, assuming the appstripe1 application stripe, the following command exports a certificate with `mykey` alias 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.

> **Note:**
>
> The directory specified using the `filepath` parameter must exist before exporting the keystore.

Task 5, Changing a Certificate Password

Use the `changeKeyPassword` WLST command. For example, assuming the system1 system stripe, the following command deletes a certificate with `testkey` alias:

```
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.

Task 6, Deleting a Certificate

Use the `deleteKeyStoreEntry` WLST command. For example, assuming the `appstripe` application stripe, the following command deletes a certificate with `orakey` alias:

```
svc.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.

Replacing Demonstration CA Signed Certificates

Oracle highly recommends that you use third-party Certificate Authority (CA) signed certificates or domain CA signed certificates when you deploy applications to a production environment. By default, any certificates created using the OPSS keystore service in the domain are signed using the demonstration CA. These demonstrations certificates should never be used in a production environment. The private key of the demonstration certificate is available to all installations of WebLogic Server, therefore each installation can generate a demo signed CA certificate using the same key. As a result, you cannot trust these certificates.

A domain CA is a self-signed certificate that acts like a CA for a domain. Unlike a demonstration CA, the private key used in a domain CA certificate is unique to each domain, and provides more security. You can create a domain CA certificate and replace all the demonstration CA certificates in a domain as described in Replacing Demo CA Certificates With Domain CA Signed Certificates.

A third-party CA validates identities and issues certificates. To get the certificate, you must create a Certificate Request and submit it to the CA. The CA will authenticate the certificate requestor and create a digital certificate based on the request. To replace demonstration certificates with third-party CA signed certificates, see Replacing Demo CA Certificates With Third-Party CA Signed Certificates.

- Replacing Demo CA Certificates With Domain CA Signed Certificates
- Replacing Demo CA Certificates With Third-Party CA Signed Certificates

- Replacing the Demo CA Trust Service Certificate
- Setting Up a Security Hardened Domain: An Example
  The following example procedure applies the steps provided in the previous sections to illustrate how you can set up a domain that uses either third-party CA signed or internal CA signed certificates throughout the domain in place of the demonstration CA certificates.

Replacing Demo CA Certificates With Domain CA Signed Certificates

To replace all the demonstration CA signed certificates in the domain with domain CA signed certificates, use the following procedure:

1. Use the keytool command to create a JKS file that contains the custom CA key and certificate. For example:

   ```
   keytool -genkeypair -alias customca -keyalg RSA -keysize 2048 -dname "cn=customca,
   o=oracle" -validity 3650 -keystore /tmp/file.jks -storepass password1 -keypass
   password2
   ```

2. Import the JKS file into the KSS keystore `kss://system/castore` using the `svc.importKeyStore` keystore service online WLST command. Before using this command you must start WLST and connect to the server.

   ```
   connect("<wls adminuser>","<wls admin password>","t3://<host>:<port>")
   svc = getOpssService(name='KeyStoreService')
   :
   :
   wls:/base_domain/domainRuntime/> svc.importKeyStore(appStripe='system',
   name='castore', password='password1', aliases='customca',keypasswords='password2',
   type='JKS', permission=true, filepath='/tmp/file.jks')
   ```

   Note that the values of some parameters used in `svc.ImportKeyStore` WLST command are derived from the values specified in the `keytool` command in the previous step. The corresponding parameters, with the sample values used in these commands, are shown in the following table.

Keytool Parameter	svc.importKeyStore Parameter	Sample Value
`-alias`	`aliases=`	customca
`-keystore`	`filepath=`	/tmp/file.jks
`-storepass`	`password=`	password1
`-keypass`	`keypasswords=`	password2

3. Verify that the key pair and certificate have been imported correctly into the KSS store. The output of the command should contain an entry with the alias `customca`, as shown.

   ```
   wls:/base_domain/domainRuntime/> svc.listKeyStoreAliases(appStripe='system',
   name='castore', password='', type='Certificate')

   Already in Domain Runtime Tree

   democa
   customca
   ```

4. Export the certificate from `kss://system/castore`, and import it into `kss://system/trust`. The value of the `filepath` parameter must be the same in the export and import commands.

```
wls:/base_domain/domainRuntime/>
svc.exportKeyStoreCertificate(appStripe='system', name='castore',
password='', alias='customca', type='Certificate', filepath='/tmp/cert.txt')

Already in Domain Runtime Tree

Certificate exported.

wls:/base_domain/domainRuntime/>
svc.importKeyStoreCertificate(appStripe='system', name='trust', password='',
alias='customca', keypassword='', type='TrustedCertificate', filepath='/tmp/
cert.txt')

Already in Domain Runtime Tree

Certificate imported.
```

5. Verify that the certificate has been imported into the `kss://system/trust` store. The output of the command should include an alias by the name of `customca`, as shown.

```
wls:/base_domain/domainRuntime/> svc.listKeyStoreAliases(appStripe='system',
name='trust', password='', type='TrustedCertificate')

Already in Domain Runtime Tree

democa
olddemoca
customca
```

6. Add the following property to the `jps-config.xml` and `jps-config-jse.xml` files in *`DOMAIN_HOME`*`/config/fmwconfig` in the keystore service instance configured in the default context:

```
<property name="ca.key.alias" value="customca"/>
```

   To do so, follow the procedure in Configuring Services with Scripts to create the `updateServiceInstanceProperty.py` script.

   Execute the script as follows:

```
>cd $ORACLE_HOME/common/bin
>wlst.sh /tmp/updateServiceInstanceProperty.py -si keystore.db -key
ca.key.alias -value customca
```

7. Restart all the servers in the domain. The domain is now ready to function with the new custom CA certificate. The older demoCA certificate and key still exist in the domain, but are not used.

8. If you wish to use domain CA signed certificates in production, renew the certificates by executing the following online WLST command:

```
svc.listExpiringCertificates(days='9999', autorenew=true)
```

Replacing Demo CA Certificates With Third-Party CA Signed Certificates

For each demo certificate in the domain that needs to be replaced with a third-party CA signed certificate, do the following:

1. Generate a CSR using the alias of the certificate that needs to be replaced. Note that the alias must be of type "Certificate", and not "TrustedCertificate".
2. Submit the new CSR to a third-party Certificate Authority (CA). The CA will sign the public key in the CSR and return a CA signed certificate and its own certificate.

   Some CAs return a certificate chain containing both the CA signed certificate and its own certificate, instead of two separate certificates.
3. Import the CA signed certificate (or the certificate chain) using the alias of the certificate that is being replaced.
4. If the CA has provided its own certificate separately, import that CA certificate as a trusted certificate in the trust store used by the product or application. By default, most applications use the domain trust store `kss://system/trust` for trust.

> **See also:**
>
> Managing Certificates with Fusion Middleware Control
>
> Managing Certificates with WLST

Replacing the Demo CA Trust Service Certificate

The OPSS trust service certificate is stored in the following keystores in the `opss` stripe:

- `trustservice_ks` which is a keystore that contains the private key
- `trustservice_ts` which is a trust store that contains the certificate

By default, the trusted certificate is signed using the demonstration CA and has identical copies in both the keystores listed above. To replace it with domain CA or third party CA signed certificate, do the following:

1. Replace the certificate in stripe `opss` and keystore `trustservice_ks` as described in either (but not both) of the following sections:
   - Replacing Demo CA Certificates With Domain CA Signed Certificates
   - Replacing Demo CA Certificates With Third-Party CA Signed Certificates
2. Export the certificate from the keystore and import it into the trust store with the stripe `opss` and keystore `trustservice_ts` using the following WLST commands:

```
svc.exportKeyStoreCertificate(appStripe='opss', name='trustservice_ks',
password='', alias='trustservice', keypassword='', type='Certificate',
filepath='/tmp/cert.txt')

svc.deleteKeyStoreEntry(appStripe='opss', name='trustservice_ts', password='',
alias='trustservice', keypassword='')
```

```
svc.importKeyStoreCertificate(appStripe='opss', name='trustservice_ts',
password='', alias='trustservice', keypassword='',
type='TrustedCertificate', filepath='/tmp/cert.txt')
```

Setting Up a Security Hardened Domain: An Example

The following example procedure applies the steps provided in the previous sections to illustrate how you can set up a domain that uses either third-party CA signed or internal CA signed certificates throughout the domain in place of the demonstration CA certificates.

1. Replace the demo CA with the custom CA as described in Replacing Demo CA Certificates With Domain CA Signed Certificates.
2. Remove the `democa` trusted certificate from `kss://system/trust` and add the certificate from the third-party or internal CA. You can do so using the `deleteKeyStoreEntry` and `importKeyStoreCertificate` commands, respectively.
3. In the `system/demoidentity` and `opss/trustservice_ks` keystores, replace all `demoCA` certificates in the deployment with certificates signed by the third-party or internal CA. To do so, follow these steps for each certificate
   a. Generate a CSR using the alias of the certificate that needs to be replaced. To do so use the `exportKeyStoreCertificateRequest` command. For sample usage, see Task 2, Generating a CSR for a Key Pair.
   b. Submit the CSR to the third-party or internal CA and obtain a certificate.
   c. Import the third-party or internal CA signed certificate into the keystore using the same alias as the certificate it is replacing (it will replace the demo CA signed certificate).
4. Replace the certificate in `opss/trustservice_ts` as follows:
   a. Export the certificate with alias `trustservice` from `opss/trustservice_ks` to a file.
   b. Delete the trusted certificate with alias `trustservice` from `opss/trustservice_ts`.
   c. Import the certificate from the file into the keystore `opss/trustservice_ts` using the alias `trustservice`.

   For sample usage of these steps and the commands required, see step 2 in Replacing the Demo CA Trust Service Certificate.

How Fusion Middleware Components Use the Keystore Service

After configuring keystores in Oracle WebLogic Server, use them to generate keys and certificates. You can also configure Node Manager to use the Keystore Service.

See also:

Configuring SSL in *Administering Security for Oracle WebLogic Server*

Configure Keystores in *Oracle WebLogic Server Administration Console Online Help*

Configuring Node Manager to Use the OPSS Keystore Service in *Administering Oracle Fusion Middleware*

Administering Node Manager for Oracle WebLogic Server

Synchronizing the Local Keystore with the Security Store

All Oracle Fusion Middleware components keep keys and certificates in a central security store. However, because certain infrastructure components must be started before the security store is available, some components use a local file keystore instance instead.

To synchronize the local file keystore with the central security store, use the `syncKeyStores` command. Synchronization is a one-way procedure in which key data is read from the central security store and synchronized in the local file keystore.

syncKeyStores Usage

The usage of the `syncKeyStores` command is:

```
syncKeyStores(appStripe='system', keystoreFormat='KSS')
```

This command accesses the system stripe in the central security store and downloads its contents into a file named `keystores.xml` in `DOMAIN_HOME/config/fmwconfig` on the local system.

For more information, see `syncKeyStores` in *WLST Command Reference for Infrastructure Security*.

When to Synchronize the Keystores

To determine when to use the `syncKeyStores` command to synchronize the keystores, consider the following:

- If the keystore being updated belongs to WebLogic Server, then the keystore should be located under the system stripe.
- If you change the domain truststore, such as adding or removing a trusted certificate, you need to execute `syncKeyStores` to synchronize the local copy used by WebLogic Server with the central security store.
- You do not need to execute the `syncKeyStores` command if a layered component such as Oracle Web Services Manager, or a Java EE application, update the keystore. These

components/applications access their keys and certificates from the central security store directly.

13

Introduction to Oracle Fusion Middleware Audit Framework

The Oracle Fusion Middleware Audit Framework allows you to audit application events. Using this framework, you create events specific to your application, register the application at deployment, and generate audit reports.
This chapter includes the following topics:

- What Are the Audit Objectives?
- Audit Terminology
- About Auditing with Oracle Fusion Middleware Audit Framework
- Understanding Audit
- About Audit Attributes, Events, and Event Categories
- About Audit Definition Files
- About Mapping and Version Rules

What Are the Audit Objectives?

The objectives of audit are to comply with regulations, to monitor business activity, and to obtain data for risk analysis.

Compliance

To comply with regulations required in the enterprise and to allow the review of compliance policies, customers must audit identity information and user access events on applications and devices across the enterprise, including the following:

- User profile change
- Access rights change
- User access
- Operational activities, such as like application start and stop, upgrade, and backup

Monitoring

Audit data allows you to monitor activity, to create dashboards, and to build key performance indicators to observe the health of the various systems in the enterprise.

Analytics

Audit data analysis can be used to assess the efficacy of controls and risks. Based on historical data, a risk score is calculated and assigned to a user. Then, any runtime evaluation of a user access to systems can include risk scores as additional criteria to determine access permission.

Audit support across enterprises is not uniform. For example, there are no standards to generate audit records, format records, or define audit policies. As a result, audit solutions have a number of drawbacks:

- There is no centralized audit framework.
- Audit support is inconsistent from application to application.
- Audit data, audit policies, and configuration are scattered across the enterprise.
- Cross-component analysis of audit is complex and time-consuming.
- Scattered data, lack of consistency, and decentralization make the audit solutions fragile with idiosyncrasies.

Audit Terminology

This section introduces several audit terms used in this document.

Component

A *component* refers to an Oracle Fusion Middleware component.

Audit-Aware Components

An *audit-aware component* is a component that is integrated with Audit Framework, whose audit policies can be configured and whose events can be audited.

Audit Store

The *audit store* is a database that has a predefined audit schema and that stores audit events. After you configure the audit store, the audit loader periodically uploads data to this database. Audit data is cumulative and grows in size over time. Ideally, the audit store should be a database not used by other applications but used exclusively by audit. The audit store stores audit events generated by components as well as user applications integrated with Audit Framework.

Audit Definition File

An *audit definition file* is a file where an applications specify its specific audit rules (such as events and filters) that control audit.

Audit Events

An *audit event* is an event that is recorded by Audit Framework. This framework provides a set of generic events that you map to application audit common events,

such as authentication or policy change. It also allows you to define specific application events and to update audit configuration with Fusion Middleware Control or with WebLogic Scripting Tool (WLST) commands.

Audit Loader

The *audit loader* is a module of Oracle WebLogic Server that supports audit activity in the server. After you configured the audit store, the audit loader collects audit records of all running components and loads them to the audit store. For Java components, the audit loader starts when the container starts up. To upload events with the audit loader, register the system component with audit (with the `registerAudit` WLST command) or use the standalone audit loader.

Audit Policy

An *audit policy* specifies the events that Audit Framework captures for a particular component. You define policies at the component level (so that it applies to a particular component), or at the domain level (so that it applies to all components in the domain).

Bus-Stop Files

A *bus-stop file* is a local file that contains audit data records. Bus-stop files are simple text files that can be queried easily to look up specific audit events. If audit is configured in the domain, then the data in these files is periodically uploaded to the audit store after a configurable time interval. If audit is not configured in the domain, then the data is kept in bus-stop files.

You correlate and combine audit data from multiple components in a report, for example, when you want to identify authentication failures in all middleware components and instances.

By default, the bus-stop files are located the following directory:

Weblogic Domain Home/servers/*server_name*/logs/auditlogs

with sub-directories for each component bus-stop files. For example, OPSS bus-stop files are kept in the following directory:

Weblogic Domain Home/servers/*server_name*/logs/auditlogs/JPS

Event Filters

An *event filter* is a filter that controls whether the event is logged. For example, a successful login event to a component is logged only for a certain subset of users.

Audit Configuration MBeans

Audit configuration MBeans are the MBeans that manage audit configuration. For Java components and applications, these MBeans are present in WebLogic Administration Server and the audit configuration is centrally managed. For system components, each component has its separate MBeans.

See also:

- About the Audit Store
- How Audit Data Is Stored
- About Audit Definition Files
- Overview of the Audit Schema
- Running Standalone Audit Loader
- Managing Audit Policies
- Audit Events

About Auditing with Oracle Fusion Middleware Audit Framework

The following sections describe the Audit Framework support to audit components:

- Overview of Oracle Fusion Middleware Audit Framework
- About Components and Applications
- Overview of Oracle Fusion Middleware Audit Framework
- About Components and Applications

Overview of Oracle Fusion Middleware Audit Framework

The Audit Framework includes the following features:

- A uniform way to administer audits across Java components, system components, and applications.
- A Java component audit, including:
  - Support audit for applications that are not audit-aware.
  - The ability to search for audit data at any application level.
- Capturing authentication history and failures, authorization history, user management, and other common transaction data.
- Flexible policies including:
  - Previously seeded audit policies, which capture most common audit events, available for ease of configuration.
  - A tree-like policy structure.
- The ability to write your own reports based on the published audit schema. For information about audit analysis, see Using Audit Analysis and Reporting .
- Keeping audit data and files in a common location (the audit store), which simplifies record maintenance.
- A common audit record format including:

  - Baseline attributes such as outcome (status), event date-time, and user.
  - Event-specific attributes such as the authentication method, source IP address, target user, and resource.
  - Contextual attributes such as the execution context ID (ECID), and session ID.
- A common and unified way to configure audit policies for the entire domain.
- Oracle Fusion Middleware support, so that audit:
  - Can be used across Oracle Fusion Middleware components and services.
  - Integrates with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control).
  - Integrates with WLST.
- A dynamic metadata model that integrates with the Audit Framework and that allows applications to:
  - Register at any time.
  - Define and log specific audit events.
  - Upgrade definitions independent of release cycles by providing event definitions versions.

About Components and Applications

Oracle Fusion Middleware Audit Framework provides a centralized framework for all Oracle Fusion Middleware products. Specifically, it provides audit for the following applications and components:

- Middleware Platform - This includes Java components such as OPSS and Oracle Web Services Manager. All the deployed applications leveraging Java components benefit from audit, which happens at the platform level.
- Java EE applications - The framework provides audit for Java EE applications, including Oracle Java EE-based components, and applications and components can specify their own specific audit events.
- System Components - For system components, such as Oracle HTTP Server, the framework provides an end-to-end solution similar to that of Java components, including APIs for C and C++ applications.

> **See also:**
>
> Oracle Fusion Middleware Components in *Administering Oracle Fusion Middleware*.

Understanding Audit

The following sections explain fundamental audit concepts:

- The Audit Model
- Audit Setup: Main Steps
- About the Audit Store

The Audit Model

The audit model provides a standards-based, integrated framework for Java EE and SE applications and components across Oracle Fusion Middleware.

Dynamic Model

The Oracle Fusion Middleware Audit Framework features a dynamic audit model that lets applications manage audit event definitions and make version changes independent of release cycles. Audit event definitions can be dynamically updated at redeployment.

Application Life Cycle Support

The model supports all aspects of the application life cycle from design to development to deployment.

Application Registration

A versatile registration lets you register applications with audit in different ways:

- Declaratively, by packaging the configuration in the `META-INF` directory of the application Enterprise ARchive (EAR) file.
- Programmatically, by calling the audit registration methods.
- At the command line, by calling WLST audit commands.
- When you create a domain, by specifying security artifacts in a product template.

Distributed Environments

Oracle Fusion Middleware Audit Framework supports distributed environments with multiple servers. It monitors the audit store so changes in audit policies introduced in one server are synchronized with all other servers in the domain.

Consider, for example, a distributed environment consisting of an Administration Server and three Managed Servers. A single security store (that includes audit data) supports all the servers in the domain. When you change an audit policy in the Administration Server with Fusion Middleware Control, then those changes are automatically propagated and synchronized with all other servers in the domain.

See also:

Performing Declarative Audit Registration

registerAudit in *WLST Command Reference for Infrastructure Security*

How Security Artifacts Are Seeded

About the Audit Store

The audit store contains component event definitions, attribute table mappings, and audit policies.

The audit store includes:

- Audit registration that allows you:
  - Create, modify, and delete event definition entries.
  - Create attribute database mappings to store audit data.
- The service that retrieves event definitions and runtime policies.
- Audit MBean commands that allow you to look up and modify component audit definitions and runtime policies.

The Audit Framework requires a database to store audit data, and this database can be any of the supported ones. For information about supported types, see Supported File, LDAP, and Database Stores.

When a new application registers with audit, the following 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 tables
- Runtime audit policies

How Audit Data Is Stored

Audit data resides in intermediate or permanent storages.

- Intermediate storage, in bus-stop files. Each component instance writes to its own separate bus-stop file. Bus-stop files are text-based and easy to query.
- Permanent storage, in the audit store (if configured in the environment). Audit records generated by all components in the domain are written to the same store.

Advantages to Using a Database Store

Having the audit records stored in bus-stop files has some limitations:

- You cannot view domain-level audit data.
- You cannot obtain reports easily.

And there are advantages to using the audit store:

- It allows you to generate audit reports.
- The database store contains records from all components in the domain, whereas the bus-stop contains audit records for one component only.
- It improves performance.

About the Oracle Fusion Middleware Audit Framework

The Audit Framework provides a set of interfaces for any audit-aware components integrating with it. 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. These interfaces allow applications to specify event details and attributes needed to provide the context of the event they want to audit.

> **See also:**
>
> Developing with Oracle Fusion Middleware Audit Framework

Audit Setup: Main Steps

The following list includes the major tasks that you carry out to you set up and maintain audit in your environment:

- Understanding the audit architecture, the essential elements of the framework, the flow of actions, and the Audit Framework. For information about these tasks, see Audit Administration Tasks.
- Integrating applications with the framework. For information about integration, see Integrating Applications with the Oracle Fusion Middleware Audit Framework.
- Creating the audit definition file that specifies the application's audit events and how they map to the audit schema. For information about audit definition files, see Creating Audit Definition Files.
- Registering the application with audit. For information about audit registration, see Registering the Application with the Audit Service.
- Migrating audit information. For information about audit data migration, see Migrating Audit Data.
- Generating audit reports. For information about audit reporting, see Using Audit Analysis and Reporting .

Understanding the Runtime Audit Event Flow

If the audit store is not configured in your environment, then the audit records are kept in bus-stop files. An application does not stop execution if it is unable to record an audit event.

The audit event flow is best understood by looking at the following sequence that takes place when an audit event occurs within an application running in an environment where you have configured audit:

1. During application deployment or service start-up, a client Java EE application registers with audit.
2. The service reads the application audit definition file and updates definitions in the audit store.
3. When a user accesses the component or application, an audit function is called to audit the event.
4. The Audit Framework checks whether to audit events with this type, status, and attributes. If they must be audited, then the audit function is called to create the event and collect information such as the status, initiator, resource, and ECID.
5. The event is stored in a bus-stop file. Each application or component has its own bus-stop file.
6. The audit loader pulls the events from bus-stop files, formats the data using the application's metadata, and moves it to the audit store.

> **See also:**
>
> Using Audit Analysis and Reporting

About Audit Attributes, Events, and Event Categories

The Audit Framework supports a model that allows you to specify and define dynamically application audit attribute groups, categories, and events.

The following sections explain this support:

- Audit Attribute Groups
- Audit Events and Event Categories
- Audit Artifact Naming Requirements

- Audit Attribute Groups
- Audit Events and Event Categories
- Audit Artifact Naming Requirements

Audit Attribute Groups

Attribute groups provide broad classification of audit attributes and consist of three types: common, generic, and custom.

- The common attribute group contains system attributes common to all applications, such as the component type, system IP address, and host name. The `IAU_COMMON` database table contains attributes in this group.
- Generic attribute groups contain attributes for audit authentication and user provisioning.
- Custom attribute groups are those defined by an application to meet specific needs. The scope of attributes in a custom group is limited to a component. These attribute groups and attributes are stored in the `IAU_CUSTOMn` table, where n denotes an integer (1,2, and so on).

> **See also:**
>
> About Generic Attribute Groups
>
> About Custom Attribute Groups
>
> About Audit Attribute Data Types
>
> OPSS Event Attributes

- About Generic Attribute Groups
- About Custom Attribute Groups
- About Audit Attribute Data Types

About Generic Attribute Groups

A generic attribute group refers to a namespace and a version number, and contains one or more attributes. The following example illustrates an attribute group with the `authorization` namespace 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>
    ……
```

You refer to 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 `authorization` attribute group is stored in the `IAU_AUTHORIZATION` table with these columns:

- `IAU_CODESOURCE` as string
- `IAU_PRINCIPALS` as string
- `IAU_INITIATORGUID` as string

About Custom Attribute Groups

A custom attribute group refers to a namespace, a version number, and one or more attributes. Each custom attribute incudes:

- 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 an attribute that can have two or more values of the same data type. These attributes support all data types except for binary types.

The following example illustrates the definition of the `Accounting` attribute group with the `accounting` namespace 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 defines the `AccountBalance` attribute with multiple values:

```
<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>
```

About Audit Attribute Data Types

Table 13-1 shows the attribute data types supported and the corresponding Java object types:

Table 13-1 Audit Attribute Data Types

Attribute Data Type	Java Object Type	Notes
Integer	Integer	NA
Long	Long	NA

Table 13-1 (Cont.) Audit Attribute Data Types

Attribute Data Type	Java Object Type	Notes
Float	Float	NA
Double	Double	NA
Boolean	Boolean	NA
DateTime	java.util.Date	NA
String	String	Maximum length 2048 bytes
LongString	String	Unlimited length
Binary	byte[]	NA

Audit Events and Event Categories

An event category contains audit events in a functional area. For example, a session category may contain login and logout events significant to the life cycle of a user session. 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:**
>
> System Categories and Events

- About System Categories and Events
- About Component and Application Categories

About System Categories and Events

A system category references common and generic attribute groups and includes audit events. System categories are the base set of component event categories and events. Applications can refer to system categories and use the events in them to log audit events and set filter preset definitions.

The following example illustrates the definition of attributes, events, and the `UserSession` system category with an attribute referencing the common `AuthenticationMethod` attribute:

```
<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>
```

About Component and Application Categories

A component or application can extend system categories or define new component event categories.

The following example illustrates the definition of a category with the `AccountNumber`, `Date`, and `Amount` attributes from the `accounting` attribute group, and it includes the `purchase` and `deposit` 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"/>
    <Event displayName="deposit" name="deposit">
       <HelpText>depositing funds.</HelpText>
    </Event>
……
 </Category>
```

Extend system categories by creating category references in your application audit definitions, listing the system events that the category includes, and adding attribute references and events to the category reference.

The following example illustrates the definition of the `ServiceManagement` system category reference with the `ServiceTime` attribute, and the `restartService` event:

```
<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>
```

Audit Artifact Naming Requirements

The name of a category, an event, or an attribute must:

- Be an English word
- Be less than 26 characters
- Contain characters a-z, A-Z, and numbers 0-9 only
- Start with a letter

About Audit Definition Files

An audit definition file specifies the application's specific audit rules (such as events and filters). Audit definition files provide a way to translate event definitions to foreign languages. There are two types of audit definition files:

- The `component_events.xml` file. For information about this file, see About the component_events.xml File.
- Translation files, which are used to display audit definition in different languages. For information about translation files, see Translation Files.
- About the component_events.xml File

About the component_events.xml File

The `component_events.xml` file specifies the properties audit uses to log audit events, including the following:

- Basic properties
  - The component type, which applications use to register with audit and obtain a runtime auditor instance
  - Major and minor version of the application
- A custom attribute group
- Event categories with attribute references and events
- Component level filter definitions
- Runtime policies

The following example illustrates the definition of this 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 &amp;quot;NorthEast&amp;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

In addition, there are runtime properties you create with Fusion Middleware Control, WLST commands, or during registration. They include the following properties:

- `filterPreset`, to specify the audit filter level
- `specialUsers`, to specify the users to audit always
- `maxBusstopFileSize`, to specify the size of a bus-stop file

> **See also:**
>
> Managing Audit Policies

About Mapping and Version Rules

Audit registration applies certain rules to create the audit data for the application, and this data is used to maintain different versions of the audit definition and to generate reports.

The following sections explain how the registration works:

- What Are Version Numbers?
- About Custom Attribute to Database Column Mappings

- What Are Version Numbers?
- About Custom Attribute to Database Column Mappings

What Are Version Numbers?

An audit definition file has a major and a minor version number. Any change introduced to an audit event definition requires updating the version number. These numbers are used by audit registration to determine the compatibility of event definitions and attribute mappings between versions. These version numbers have no relation to Oracle Fusion Middleware version numbers.

Component Version

When you register a component, audit registration checks if this is a first-time registration or an upgrade.

In case of 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 store.
3. Generates the attribute-column mapping table and saves it in the audit store.

In case of an upgrade, the current version number for the component in the audit store is compared with the new version number to determine whether to proceed with the upgrade.

Java EE Application Version

To reset the version number after you modified an application audit definition, Oracle recommends that you:

- Increase the minor version number only when making changes in an audit definition that will work with the audit data created by the previous attribute database mapping table.

  For example, suppose the current definition version 2.1. When adding a new event that does not affect the attribute database mapping table, you change the version to 2.2, and leave the major version unchanged (major=2). Similarly, increase the minor version after adding a new attribute.
- Increase major version number when making changes where the new mapping table is incompatible with the previous table.

Changes becomes effective after you restart the server.

About Custom Attribute to Database Column Mappings

When you register a new component or application, audit registration creates an attribute-to-database column mapping table from the custom attributes, and then saves this table to the audit store.

If the number of custom attributes is greater than 100, then you must create additional tables manually. OPSS ships with the tables `IAU_CUSTOM` and `IAU_CUSTOM_01` only.

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. These tables are also used to generate audit reports from custom `IAU_CUSTOM` database table.

Use the `createAuditDBView` WLST command to generate a SQL file that creates a database view of the audit definitions for your component.

Understanding the Mapping Table for your Component

A custom attribute-database column mapping has the following properties: 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, the following definition file:

```
<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>
```

maps to:

```
<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 number of the attribute-database column mapping table matches the version number of the custom attribute group. This allows your application to maintain a backward compatibility of attribute mappings across audit definition versions.

See Also:

createAuditDBView in *WLST Command Reference for Infrastructure Security*

Oracle Fusion Middleware Audit Framework Reference

What Are Version Numbers?

14

Managing Audit

This chapter explains the main administration tasks and tools you use to manage the audit store, audit policies, and bus-stop files.
This chapter includes the following topics:

- Audit Administration Tasks
- Managing the Audit Store
- Managing Audit Policies
- Understanding Audit Time Stamps
- About Audit Logs and Bus-stop Files
- Audit Database Administration
- Best Practices for Audit Event Definitions

Audit Administration Tasks

Setting up audit in your environment involves the following major tasks:

- Planning the type of store to use for audit records and the store configuration details. For information about audit store management, see Managing the Audit Store .
- Configuring and maintaining audit policies so that audit events are generated. For information about audit policies, see Managing Audit Policies.
- Configuring audit reports and queries. For information about reporting, see Using Audit Analysis and Reporting .
- Registering applications. For information about application registration, see Registering the Application with the Audit Service.
- Migrating audit information. For information about audit data migration, see Migrating Audit Data.
- Administering the audit database, including increasing the database size that stores the generated audit data, and backing up and purging that data. For information about audit administration, see Audit Database Administration.

Managing the Audit Store

The audit store is a database that provides the scalability and high-availability needed to store audit records and that allows you to view and generate reports.

The following sections introduce the audit store and explain how to manage it:

- About Audit Data Sources
- Managing Bus-Stop Files
- Configuring Standalone Audit Loader

About Audit Data Sources

When you create a domain, the process generates the audit schema, a data structure required to store audit records in the database. It also sets up an audit data source in the server that uses the audit schema. If your environment is not set up with a database to store records, then audit records are kept in bus-stop files.

> **See also:**
>
> Bus-Stop Files

Managing Bus-Stop Files

After the bus-stop file reaches a certain size and all the data was uploaded to the database, the audit loader deletes the file from the file system. Specify the location and maximum size of bus-stop files, so that bus-stop files are automatically deleted. Deleting audit files manually is not recommended.

Bus-Stop File Locations

Bus-stop files for Java components are located in the following directory:

`$DOMAIN_HOME/servers/$SERVER_NAME/logs/auditlogs/Component_Type`

Bus-stop files for system components are located in the following directory:

`$ORACLE_INSTANCE/auditlogs/Component_Type/Component_Name`

Bus-Stop File Size

In Java components, the maximum size of a bus-stop file is set with the `audit.maxFileSize` property.

In system components, the maximum size of a bus-stop file is set in the `auditconfig.xml` file:

```
<serviceInstance name="audit" provider="audit.provider">
  <property name="audit.maxFileSize" value="10240" />
  <property name=" audit.loader.repositoryType " value="Db" />
</serviceInstance>
```

When you switch from a file to a database store for audit data, all the events collected in the files are moved to the database tables and the audit files are deleted.

Configuring Standalone Audit Loader

The standalone audit loader moves records from bus-stop files to the audit store periodically. The mechanism driving the audit loader depends on the application environment:

- Java EE components and applications use the audit loader functionality provided by OPSS runtime. The standalone audit loader is not needed in these environments.
- System components and non-Java applications use the audit loader functionality provided by the `StandAloneAuditLoader` command.
- Java SE applications also use the standalone audit loader depending on where the bus-stop files are written. For information about audit for Java SE applications, see Common Audit Scenarios in Java SE Applications.

The following sections explain how to set up and run the standalone audit loader:

- Configuring the Environment
- Running Standalone Audit Loader

- Configuring the Environment
- Running Standalone Audit Loader

Configuring the Environment

The following settings apply only to non-Java applications and system components.

Before you run the standalone audit loader, set the following audit loader parameters:

- `ORACLE_HOME`, the full path to the home directory
- `COMMON_COMPONENTS_HOME`, the full path to the Java Required Files (JRF) directory
- `ORACLE_INSTANCE`, the full path of an Oracle instance directory
- `auditloader.jdbcString`, the Java Database Connectivity (JDBC) connection string for the database where the audit data is stored
- `auditloader.username`, the name of the user who runs the audit loader

In addition, make sure that the password for the database schema user is available and stored. This password is specified once.

To specify the database schema user password, use the java `StandAloneAuditLoader` command with the `-Dstore.password=true` property:

```
$JDK_HOME/bin/java
    -classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.2.1/jps-manifest.jar
    -Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
    -Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
    -Dauditloader.username=username
```

```
    -Dstore.password=true
    oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

which will prompt you to enter a password. The command generates the `cwallet.sso` file containing the password you entered.

Running Standalone Audit Loader

To run the loader, use the `StandAloneAuditLoader` command:

```
$JDK_HOME/bin/java
    -classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.2.1/jps-manifest.jar
    -Doracle.home=$ORACLE_INSTANCE -Doracle.instance=$ORACLE_INSTANCE
    -Dauditloader.jdbcString=jdbc:oracle:thin:@host:port:sid
    -Dauditloader.username=username
    oracle.security.audit.ajl.loader.StandaloneAuditLoader
```

This command is typically scheduled to run automatically so that audit records are periodically uploaded to the audit store.

Managing Audit Policies

An audit policy is a declaration of the type of events that are recorded by Oracle Fusion Middleware Audit Framework for a particular component. In Java components, the audit policy is defined at the domain level. In system components, the audit policy is managed at the component instance level.

Note that you must update audit policies when you introduce changes to the application environment, such as adding or deleting components or users. Policy changes do not require server or instance restart.

The Audit Framework lets you configure audit policies and provides high control over the types of events and data being audited. You configure audit policies with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control), WebLogic Scripting Tool (WLST), or programmatically, as explained in the following sections:

- Managing Audit Policies with Fusion Middleware Control
- Managing Audit Policies with WLST
- Managing Audit Policies Programmatically

> **See also:**
>
> Audit Terminology
>
> Oracle Fusion Middleware Audit Framework Reference

- Managing Audit Policies with Fusion Middleware Control
- Managing Audit Policies with WLST
- Managing Audit Policies Programmatically

Managing Audit Policies with Fusion Middleware Control

Follow the instructions in this section to manage audit policies for Java and system components with Fusion Middleware Control.

Task 1, Opening the Audit Page

Log in to Fusion Middleware Control and go to *Domain*, then to **Security**, and then to **Audit Policy**. The **Audit Policy** page is displayed.

Task 2, Choosing Events to Audit for a Java Component

1. In the **Audit Component Name** drop-down, choose a Java component. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.

   > **Note:**
   >
   > The values you enter in the Audit Policy Settings area do not apply to Oracle Access Manager audit.

2. In the **Audit Level** drop-down, choose the audit level according to your needs:
   - None - Selects no event categor.ies
   - Low, Medium, High - Selects subsets of event categories representing predefined audit levels.
   - Custom - Selects custom filters.

   Check flags are displayed in the **Select for Audit** column, and events in the chosen categories will be audited. View the events within a flagged category by clicking on that row in the categories table. The table of events can be edited in a custom level only.

Task 3, Customizing Audit Policies for a Java Component

1. In the **Audit Component Name** drop-down, choose a Java component. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. In the **Audit Level** drop-down, choose **Custom**. All categories become available.

3. Choose categories and events to customize the audit policy by checking the box in the **Select For Audit** column.
4. Click **Apply** to save the changes.

Task 4, Creating Audit Filters for a Java Component

1. In the **Audit Component Name** drop-down, choose a Java component. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. A pencil icon in the **Edit Filter** column denotes that a filter is available for the event. Click on a pencil icon for an event. The **Edit Filter** dialog is displayed.

   Each filter attribute has a formal name and a display name. Either name is displayed in the Edit Filter edit dialog.
3. Create a filter condition by choosing an item from the **Condition** drop-down, an operator, and a value. Then click the **add** button. When you are finished adding conditions, click **OK**.
4. Optionally, under **Users to Always Audit**, specify a comma-separated list of users to audit events initiated by these users. Audit occurs regardless of the audit level or filters that have been specified. User names you enter in this field are not validated
5. Restart Oracle WebLogic Server on which the Java component is running for the changes to take place.

Task 5, Importing and Exporting Audit Policies for a Java Component

1. In the **Audit Component Name** drop-down, choose a Java component. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. At any time while editing a policy, do one of the following:
   - Click **Export** to save the current settings to a file
   - Click **Import** to load settings from a file

Task 6, Choosing Events to Audit for a System Component

1. Go to a system component home page. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. In the **Audit Level** drop-down, choose the audit level.
   - None, selects no event categor.ies
   - Low, Medium, High - Selects subsets of event categories representing predefined audit levels.
   - Custom - Selects custom filters.

   Check flags are displayed in the **Select for Audit** column, and events in the chosen categories will be audited. View the events within a flagged category by clicking on that row in the categories table. The table of events can be edited in a custom level only.

Task 7, Customizing Audit Policies for a System Component

1. go to a system component home page. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. In the **Audit Level** drop-down, choose **Custom**. All categories become available.
3. Choose categories and events to customize the audit policy by checking the box in the **Select For Audit** column.
4. Click **Apply** to save the changes.

Task 8, Creating Audit Filters for a System Component

1. Go to a system component home page. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. A pencil icon in the **Edit Filter** column denotes that a filter is available for the event. Click on a pencil icon for an event. The **Edit Filter** dialog is displayed. A filter attribute has a formal name and a display name. Either name is displayed in the Edit Filter edit dialog
3. Create a filter condition by choosing an item from the **Condition** drop-down, an operator, and a value. Then click the **add** button. When you are finished adding conditions, click **OK**.
4. Optionally, under **Users to Always Audit**, specify a comma-separated list of users to audit events initiated by these users. Audit occurs regardless of the audit level or filters that have been specified. User names you enter in this field are not validated. Particular users, such as a system administrator, will generate audit traffic anytime that user executes any auditable events for any component.
5. Restart WebLogic Server on which the system component is running.

Task 9, Importing and Exporting Audit Policies for a System Component

1. Go to a system component home page. The table of audit categories relevant to the component is displayed in the **Audit Policy Settings** area.
2. At any time while editing a policy, do one of the following:
   - Click **Export** to save the current settings to a file
   - Click **Import** to load settings from a file

Managing Audit Policies with WLST

Use this section manage audit policies with WLST.

- Viewing Audit Policies with WLST Commands
- Updating Audit Policies with WLST Commands
- Configuring Audit Policies Example

- Configuring Audit Events Example
- What Happens to Custom Configuration when the Audit Level Changes?

Viewing Audit Policies with WLST Commands

1. Connect to the WebLogic Server:

   ```
   >java weblogic.WLST
   >connect('userName', 'userPassword', 'url', 'adminServerName')
   ```

2. To view domain or Java component audit policies, use the `getAuditPolicy` command:

   ```
   (view domain audit policies)
   wls:/mydomain/serverConfig> getAuditPolicy()

   (view component audit policies)
   wls:/mydomain/serverConfig> getAuditPolicy(componentType="JPS")
   ```

3. To view a system component audit policies:

   a. Obtain the system component MBean name with the `getNonJava EEAuditMBeanName` command.

   b. Use the `getAuditPolicy` command:

      ```
      wls:/mydomain/serverConfig> getAuditPolicy

      (on="oracle.security.audit.test:type=CSAuditMBean,name=CSAuditProxyMBean"
      )
      ```

Updating Audit Policies with WLST Commands

1. Connect to WebLogic Server:

   ```
   >java weblogic.WLST
   >connect('userName', 'userPassword', 'url', 'adminServerName')
   ```

2. Go the hierarchy to access the domain of interest:

   ```
   wls:/mydomain/serverConfig>
   ```

3. Use the `setAuditPolicy` command to update the audit policy configuration.

4. For system components use the `setAuditPolicy` command and include an MBean name to update the audit policy configuration. The name for an audit MBean has the following format:

   ```
   oracle.as.management.mbeans.register:type=component.auditconfig,name=auditcon
   fig1,instance=INSTANCE,component=COMPONENT_NAME
   ```

   For example:

   ```
   oracle.as.management.mbeans.register:type=component.auditconfig,name=auditcon
   fig1,instance=instance1,component=oid1
   ```

5. For system components, such as Oracle HTTP Server or LDAPs, call `save` explicitly after issuing the `setAuditPolicy` or `importAuditConfig` commands.

   The following example illustrates this step: navigating to the root proxy MBean in the tree, using the `invoke` command to call `setAuditPolicy`, and using the `save` command:

```
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=instance
1')
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))
```

Configuring Audit Policies Example

Assume that the domain current policy audits a user named `user1`, and you want to add two names, `user2` and `user3`, to the list of users who are always audited, and remove `user1` from the list of users audited.

To accomplish these tasks, run the following command:

```
setAuditPolicy
   (filterPreset="None",addSpecialUsers="user2,user3",removeSpecialUsers="user1")
```

Configuring Audit Events Example

Assume that the domain current policy audits user logout events, and you would like to remove logout events and add audit login events.

To accomplish these tasks, run the following command:

```
setAuditPolicy (on="OHS mbean name")
(filterPreset="Custom",addCustomEvents="OHS:UserLogin",
removeCustomEvents="OHS:UserLogout")
```

Notice the `Custom` filter preset called to add and remove events.

What Happens to Custom Configuration when the Audit Level Changes?

When you configure audit 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 them.

The following sequence illustrates how changing the audit level affects audit data collection:

1. The custom audit level is set in a component's policy, and an audit filter is specified as part of the configuration.
2. At runtime, audit data is collected according to the filter.
3. The component's audit policy is then changed, for example, from the custom audit level to low with the `setAuditPolicy` command. The filter that was set up as part of the custom audit level remains in the audit configuration.
4. Thereafter, audit data is collected based on the low audit level.
5. The component's audit policy is changed back to the custom level, and an additional filter is added. This new filter is appended to the original filter. Unless the original filter is explicitly deleted, it remains part of the configuration.

6. Thereafter, audit data is collected based on both filters at the custom level.

Managing Audit Policies Programmatically

The Oracle Fusion Middleware Audit Framework provides a set of interfaces that applications can use to retrieve and update audit policies. For information about policy management, see Managing Audit Policies Programmatically.

Understanding Audit Time Stamps

Before Release 11.1.1.7.0, audit wrote out audit records using the application server's time zone. Starting with Release 11.1.1.7.0, audit and the server can use a different time zone. This means that:

- New sites see audit events written in Coordinated Universal Time (UTC).
- Sites that upgrade from release 11.1.1.6.0 continue, by default, to use the application server time zone for audit records unless you explicitly specify to use UTC.

Records in bus-stop files use UTC.

Audit Time Stamps at New Sites

Audit records in new installations use UTC time stamps. Use the `audit.timezone` property in the audit configuration to specify it:

```
<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 Time Stamps at Upgraded Sites

Audit records prior to release 11.1.1.7 used the application server time stamp. After upgrading to that release, the service configuration remains unchanged and, by default, the records are written using the application server time stamp.

If you wish to use the UTC after upgrading to 12*c*, then move the current records to avoid any potential inconsistencies and insert the `audit.timezone` service property in your configuration file.

> **See also:**
>
> Audit Service Properties

About Audit Logs and Bus-stop Files

The Audit Framework has several runtime components that log error messages. This framework provides a set of log files that you use to trace errors and to diagnose failures when the Audit Framework runs into exceptions.

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:

Logging Audit

Naming and Logging Audit Files

Understanding Audit Time Stamps

Audit Database Administration

This section explains the organization and administration of the audit schema in the following topics:

- Overview of the Audit Schema
- Base and Component Table Attributes
- Tuning Performance
- Planning Backup and Recovery
- Importing and Exporting Data
- Partitioning
- Performing Tiered Archival
- Creating Indexes on Custom Table Attributes Using Materialized Views
- Overview of the Audit Schema
- Base and Component Table Attributes
- Tuning Performance
- Planning Backup and Recovery
- Importing and Exporting Data
- Purging Data
- Partitioning
- Performing Tiered Archival
- Creating Indexes on Custom Table Attributes Using Materialized Views

Overview of the Audit Schema

The tables in the Audit Framework schema includes:

- Base data in a base table. This table contains the basic data for an audit record and is related to the component-specific tables with the `IAU_ID` attribute.
- Common attributes in the `IAU_COMMON` table. This table contains the basic data for an audit record and is related to the custom tables with the `IAU_ID` attribute.
- Generic attributes in dedicated tables.
- Custom attributes in the `IAU_CUSTOM_nnn` tables.

Not all these tables are used at the same time. The dynamic model uses the common and custom tables. The static (older) model uses the base table and component-specific tables.

> **See also:**
>
> About Audit Attributes, Events, and Event Categories
>
> The Audit Model

Base and Component Table Attributes

The audit loader stores two kinds of data records in tables:

- General information (such as time, event type, and event status), stored in one row of the common table (dynamic model) or the base table.
- Component-specific information, stored in one row of the custom table (dynamic model) or the component table.

The average audit record size is approximately 0.3 KB. Use this average when you plan to resize database tables, and in addition, monitor how your audit database size grows according to policies and level of activity.

The attributes of the base table are derived from the following file:

```
$ORACLE_HOME/modules/oracle.iau_12.2.1/components/generic/generic_events.xml
```

The attributes of the component table are derived from the following file:

```
$ORACLE_HOME/modules/oracle.iau_12.2.1/components/componentName/
component_events.xml
```

Table C-6 lists the attributes defined in the `IAU_BASE` base table. Table C-7 lists the attributes defined in the `IAU_COMMON` common table.

To get a list of attribute names for individual component tables, use the `listAuditEvents` WLST command.

> **See also:**
>
> The Audit Schema
>
> listAuditEvents in *WLST Command Reference for Infrastructure Security*.

Tuning Performance

For efficient queries, an index is created by default for the `IAU_TSTZORIGINATING` column in the base table and for each of the component-specific tables.

The default index in `IAU_BASE` is `EVENT_TIME_INDEX`, and in component tables is *`tableName`*`_INDEX` (such as `OVDCOMPONENT_INDEX`, `OIDCOMPONENT_INDEX`, `JPS_INDEX`).

Additional columns and indexes in the common table are:

- `IAU_TSTZORIGINATING`, indexed by `DYN_EVENT_TIME_INDEX`
- `IAU_AuditUser`, indexed by `DYN_USER_INDEX`
- `IAU_ComponentType`, indexed by `DYN_COMPONENT_TYPE_INDEX`
- `IAU_EventCategory`, indexed by `DYN_EVENT_CATEGORY_INDEX`
- `IAU_EventType`, indexed by `DYN_EVENT_TYPE_INDEX`

Planning Backup and Recovery

When planning the audit database backup, consider the following points.

- Growth rate of audit events

  Your audit policies determine the number of audit events the framework generates and the number of audit events generated daily determines how often you want to back up the store.

- 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.

Use Oracle Database Utility Recovery Manager (RMAN) to back up and recover an Oracle Database. Note that you need to back up the `IAU_DISP_NAMES_TL` translation table just once, because it typically does not change over time.

> **See also:**
>
> Backing Up and Recovering a Database-Based Security Store

Importing and Exporting Data

Import and export the audit schema to migrate data, for example, if you wish to combine several audit repositories into a single audit store, or if you wish to scale up the database. Use Oracle Data Pump to import and export data from an Oracle Database.

Purging Data

The framework provides the following SQL scripts to purge records from the audit store, in the directory `$MW_HOME/oracle_common/common/sql/iau/scripts`:

- `auditDataPurge.sql`
- `auditDeleteData.sql`
- `auditReclaimSpace.sql`

To delete records without taking any other action, use `auditDeleteData.sql`. This script has the following syntax:

```
auditDeleteData.sql audit_schema_user number_of_days_to_keep
```

For example, to delete records older that 100 days:

```
sqlplus> @auditDeleteData.sql DEV_IAU 100
```

To reclaim space, use `auditReclaimSpace.sql`. This script has the following syntax:

```
auditReclaimSpace.sql audit_schema_user
```

To delete audit records and reclaim space, use `auditDataPurge.sql`. This script has the following syntax:

```
auditDataPurge.sql audit_schema_user number_of_days_to_keep
```

For example, to delete all records older than 100 days and enable row movements to shrink space:

```
sqlplus> @auditDataPurge.sql DEV_IAU 100
```

Partitioning

Not all database systems support partitioning, all the tables in the audit schema are not partitioned by default.

Because audit data grows over time, if you store a large volume of data, then consider partitioning the audit schema to allow for easier archiving.

Benefits of partitioning include:

- Improved Performance: If a table is range-partitioned by time stamps, for example, then queries by time stamps can be processed on the partitions within that time frame only.
- Better Manageability: Partitions can be created on separate tablespaces (thus different disks). This lets you move older data to slower and larger disks, while keeping newer data in faster and smaller disks.

In addition, partitioning makes archival much easier.

- Increased Availability: If a single partition is unavailable, for example, and you know that your query can eliminate this partition from consideration, then the query can be processed without waiting for the unavailable partition to become available.

Performing Tiered Archival

Partitioning tables allows you to store individual partitions on different storage tiers. Typically, you 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.

Oracle Information Lifecycle Management (ILM) features streamlined data management with partitioning and compression.

Creating Indexes on Custom Table Attributes Using Materialized Views

Database views enable you to run queries against audit records. Furthermore, you can create indexes on custom tables for their attributes with indexable views. We recommend that applications create a simple view first and switch to an indexable view later on when needed.

To create an indexable view:

1. Specify the event attributes in the `component_events.xml` file. In Release 12c (12.2.1) the new `indexable` attribute has been added to the `AttributeType` element.

```
Component_events.xml
<?xml version="1.0"?>
<AuditConfig xmlns="http://xmlns.oracle.com/ias/audit/audit-2.0.xsd">
    <AuditComponent componentType="ApplicationAudit" major="1" minor="1">
        <Attributes ns="accounting" version="1.0">
            <Attribute name="TransactionType" displayName="Transaction Type"
type="string" order="1" indexable="true">
               <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" indexable="true">
                <HelpText>Account status.</HelpText>
            </Attribute>
        </Attributes>
    ...
</AuditComponent>
</AuditConfig>
```

2. Create indexable views either at registration or after registration.

   If at registration time, use any of the following:

   - Specify the `opss.audit.iauview` deployment parameter as `INDEXABLE` in `weblogic-application.xml`.
   - Pass the `createView=INDEXABLE` option to the `registerAudit` command.

- Implement the `AuditRegistrationExt` interface and returning `AuditRegistrationExt.TYPE.INDEXABLE` from the `getIAUViewSupportType` method.

If after registration time, use any of the following:

- Run the `createIAUView` command to switch to an `INDEXABLE` view.
- Generate a SQL script with the `createAuditDBView` command, to which you specify an `INDEXABLE` view type. Then run the script as administrator.

If you change event definitions, then drop the associated materialized view before redeployment.

Best Practices for Audit Event Definitions

This section provides guidelines for managing audit configuration and maximizing the value of the collected audit data, in the following topics:

- Guidelines for Naming Events
- Differentiating Events
- Event Categorization
- Use of Generic Attributes
- Use of Component Attributes
- Guidelines for Linking Across Components
- Updating Event Definitions

Guidelines for Naming Events

Ensure that all audit names conform to the following naming conventions:

- Do not use any of the following characters in component Type, Category, Event, or Attribute names: !, @, “, #, $, %, ‘, (,), *, +, comma, period, -, _, /, space, :, ;, <, =, >, ?, {, }, ~, ^.
- The space character is allowed in display names only.

When you name audit events:

- Do not prefix the event name with the component name.
- Try to make names as specific as possible. For example, use HTTPResponse instead of Response when defining an HTTP response code.

- Do not append the suffix Event to all event names. For example, instead of AuthenticationEvent, use Authentication.

Differentiating Events

To optimize the use of events:

- Define separate events for each operation. For example, instead of defining the event Policy with attributes Delete and Create, define the separate events PolicyCreate and PolicyDelete.
- Do not define separate events for the events success and failure. For example, do not define the separate events PolicyCreateSuccess and PolicyCreateFailure. Instead use the EventStatus attribute to record them.

Event Categorization

To categorize events:

- Refer 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.

Use of Generic Attributes

When you define generic attributes:

- Include the following event attributes:
  - Initiator - the user who performed the event action
  - MessageText - a short description of what happened.
  - EventStatus - the event outcome
  - FailureCode - the error code applicable to the failure.
- Use the attribute Resource - the object that was affected, such as the accessed URI or the granted permission.

Use of Component Attributes

The Audit Framework considers the union of all common attributes in each event and defines a database column for each of them. So try to define as many common attributes as possible.

Guidelines for Linking Across Components

Define enough information so that events generated by your component can be cross linked to other events.

Updating Event Definitions

The `component_events.xml` file provides version support. If you decide to change the event definitions, make sure to delete any associated materialized view.

15

Using Audit Analysis and Reporting

This chapter describes how to configure audit reporting and view audit reports.
This chapter includes the following topics:

- About Audit Reporting
- Audit Reporting with the Dynamic Metadata Model

About Audit Reporting

The Oracle Fusion Middleware Audit Framework offers two approaches to audit reporting. The approach you adopt is determined by the audit model that components use at your site:

- Dynamic Metadata Model

  This model was introduced in 11*g* Release 1 (11.1.1.6.0). Oracle Fusion Middleware installations starting with 12*c* (12.1.2) use this mode. For information about this model, see Audit Reporting with the Dynamic Metadata Model.

- Report Template Model

  This earlier model is used by system components. When you upgrade, the Oracle Fusion Middleware Audit Framework continues to use this model.

Audit Reporting with the Dynamic Metadata Model

Audit events are saved into the `iau_common` common attribute table and the *`iau_custom_nnn`* custom attribute tables. Oracle Platform Security Services 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.

To set up audit reporting with the Dynamic Metadata Model:

1. Register your application with audit. Audit registration creates audit views that lets you run queries on audit data. For information about generating audit views manually, see Registering the Application with the Audit Service.
2. Configure audit policies so that audit logs events to generate audit data. For information about audit logging, see Logging Audit Events Programmatically.
3. Configure the audit loader to ensure bus-stop files are migrated to the database. For information about bus-stop files, see Managing Bus-Stop Files.
4. If audit views were not created at registration, then:
   - Use `createAuditDBView` to generate a SQL script of audit definitions.
   - Log in to the database as the `IAU` schema user to create a view using the SQL script.
5. Configure your reporting application to query the view.

> **See also:**
>
> createAuditDBView in *WLST Command Reference for Infrastructure Security*
>
> Audit Views Created at Registration
>
> Manually Created Audit Views

Audit Views Created at Registration

Starting with Release 12*c* (12.2.1), database views of your audit data can be created at registration. The views supported are:

- Simple view - This view is based on the runtime database mappings of the component attributes against the columns in `IAU_COMMON` and `IAU_CUSTOM_*` tables.

  A Simple database view is created in the `IAU VIEWER` schema when a component registers with audit. This view type is available for all supported databases. Note that a simple view is automatically generated for the component when you create the schema with Oracle Fusion Middleware Repository Creation Utility.

- Indexable view - A simple view that is indexable. This view leverages Oracle materialized views for improved performance of reporting queries. These views are supported ready-to-use for Oracle Databases only.

  Use indexable views with care, as they can impact the audit loader performance. A component can switch to an indexable view with the `createIAUView` WebLogic Scripting Tool (WLST) command. Qualified event attributes must be present in the component audit definitions. For information about indexes, see Creating Indexes on Custom Table Attributes Using Materialized Views.

File audit repositories do not support views.

Manually Created Audit Views

To create audit views manually, use the `createAuditDBView` command. Here is the output that this command generated when it was called to create a view for the `ApplicationAudit` component:

```
-- 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';
```

Part IV
Developing with OPSS APIs

This part contains the following chapters:

- Integrating Application Security with OPSS
- The Security Model
- Developing with the Credential Store Framework
- Developing with the User and Role API
- Developing with the Identity Governance Framework
- Developing with the Keystore Service
- Developing with Oracle Fusion Middleware Audit Framework
- Configuring Java EE Applications to Use OPSS
- Configuring Java SE Applications to Use OPSS

16

Integrating Application Security with OPSS

This chapter describes a number of security use cases, including the propagation of identities in domains and across domains, and the typical life cycle of an Oracle Application Development Framework (Oracle ADF) application security.
This chapter includes the following sections:

- About Security Challenges
- Security Integration Use Cases
- The OPSS Trust Service
- Propagating Identities over HTTP
- Propagating Identities with the OPSS Trust Service
- Implementing a Custom Graphical User Interface
- Securing Oracle ADF Applications
- Code and Configuration Examples
- Propagating Identities with JKS

About Security Challenges

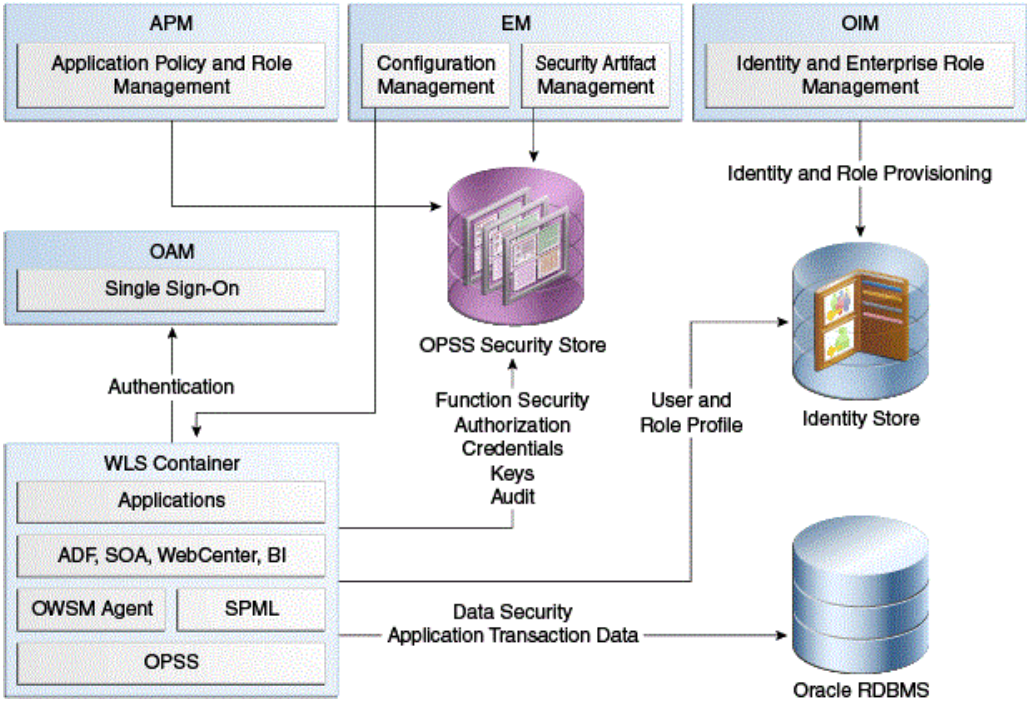
This chapter introduces use cases that solve several typical security challenges. Use these use cases as the departing point to solve your particular application security requirements. The audience includes developers, security architects, and security administrators, and the solutions presented offer both declarative and programmatic approaches.

The top security issues that you face include managing users, user passwords, and controlling access to resources. OPSS provides solutions to these challenges by supporting:

- External security artifacts separate from the application logic
- A declarative approach to security
- A complete user identity life cycle
- Policy-driven access controls

Figure 16-1 illustrates how applications access the security stores and the tools you use to manage them.

Figure 16-1 Applications, Security Stores, and Management Tools



See also:

OPSS Architecture Overview

Introducing Oracle Access Management in *Administering Oracle Access Management*

Developing Fusion Web Applications with Oracle Application Development Framework

Developing Applications with Oracle Security Developer Tools

OPSS API References

Security Integration Use Cases

The following sections introduce several use cases:

- Authentication
- Identities
- Authorization

- Credentials
- Audit
- Identity Propagation
- Administration and Management
- Integration

Each use case includes a brief description of the problem it attempts to solve, the security and features involved, and links to details.

- Authentication
- Identities
- Authorization
- Credentials
- Audit
- Identity Propagation
- Administration and Management
- Integration

Authentication

The following sections describe authentication use cases:

- Java EE Application Requiring Authenticated Users - Users must be authenticated to access a Java EE application.
- Java EE Application Requiring Programmatic Authentication - A Java EE application requires authenticating a user programmatically.
- Java SE Application Requiring Authentication - A Java SE application requires authenticating against the domain identity store.
- Java EE Application Requiring Authenticated Users
- Java EE Application Requiring Programmatic Authentication
- Java SE Application Requiring Authentication

Java EE Application Requiring Authenticated Users

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:

- A single LDAP store.
- Several LDAP stores of the same kind.
- Several LDAP stores of different kinds, such as, for example Microsoft Active Directory and Oracle Internet Directory.
- A single DB store.
- Several LDAP and DB 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:

- Configuring the appropriate authentication providers according to your particular set of user repositories.
- Configuring the authentication provider in case of a mixed LDAP or mixed LDAP and DB types.

According to the repository used, the details of this use case are divided into the following scenarios:

- Single user repository - Configure the appropriate WebLogic Server Authentication Provider.
- Multiple user repositories - Configure the LDAP providers.
- DB repositories - Configure the database providers.

For information about authentication providers, see WebLogic Server Authentication Providers.

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).

This use case requires using OPSS APIs to authenticate a user, and it features:

- Configuring authentication providers for a Java container
- Using the `oracle.security.jps.service.login.LoginService` interface to authenticate the user.

For information about authentication, see About Authentication in Java EE Applications.

Java SE Application Requiring Authentication

A Java SE application must authenticate users against the LDAP identity store used in the domain. The application code requesting authentication must be the 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 `oracle.security.jps.service.login.LoginService` interface. A Java SE application can use only one identity login module.

For information about using login modules programmatically, see Using the Login Modules in Java SE Applications.

Identities

The following sections describe identity use cases:

- Application Running in Two Environments - An Application, to run in two different environments, requires access user profile information in a single LDAP store.

- Application Accessing User Profiles in Multiple Stores - An Application requires access user profile information stored in multiple LDAP stores.
- Application Running in Two Environments
- Application Accessing User Profiles in Multiple Stores

Application Running in Two Environments

An application that can run in two different environments requires access to user profile information stored in an LDAP store. The LDAP server may differ with the environment.

More specifically, this use case assumes that:

- The application uses the `UserProfile.getEmail` method.
- In one environment, mail is configured in the Microsoft Active Directory server:

  ```
  mail.attr = msad_email
  ```

- In the other environment, mail is configured in the Oracle Identity Directory server:

  ```
  mail.attr = mail
  ```

For the application to retrieve the correct information without modifying the code and regardless of the environment (first or second) in which it runs, you must configure the identity store providers with the correct property in each of the environments.

In Microsoft Active Directory, set the identity store provider `name` property:

```
<property name="mail.attr" value="msad_mail">
```

In Oracle Identity Directory, set the identity store provider `name` property:

```
<property name="mail.attr" value="mail"
```

For information about identity store provider configuration, see Configuring the Identity Store Provider .

Application Accessing User Profiles in Multiple Stores

An application requires access to user profile information located in more than one LDAP store servers. In this scenario, you must configure the environment for multiple LDAP stores. For information about the procedure, see Configuring Single and Multiple LDAPs.

Authorization

The following sections describe authorization use cases:

- Java EE Application Accessible by Specific Roles - A Java EE application accessible only by users configured in web descriptors.
- Oracle ADF Application Requiring Fine-Grained Authorization - An Oracle ADF application requires fine-grained authorization.
- Application Securing Web Services - An requires securing web services.
- Java EE Application Requiring Codesource Permissions - A Java EE application requires codesource permissions.
- Non-Oracle ADF Application Requiring Fine-Grained Authorization - A non-Oracle ADF application requires fine-grained authorization.

Java EE Application Accessible by Specific Roles

For a Java EE application that allows access only to 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 information about declarative security, see the following topics in *Developing Applications with the WebLogic Security Service*:

- Using Declarative Security with Web Applications
- Using Declarative Security with EJBs

Oracle ADF Application Requiring Fine-Grained Authorization

For an Oracle ADF application that requires fine-grained authorization at the level of individual controls on the pages of the application, you must externalize policies and customize them after you application deployment.

> **See also:**
>
> Securing Oracle ADF Applications
>
> Enabling ADF Security in a Fusion Web Application in *Developing Fusion Web Applications with Oracle Application Development Framework*

Application Securing Web Services

The application requires securing web services with fine-grained authorization. For information about Web services, see Configuring Authorization Using Oracle Web Services Manager in *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Java EE Application Requiring Codesource Permissions

A Java EE application requires codesource permissions to perform specific actions. Typical examples are reading a credential from the credential store or looking up policies. For information about permissions, see Configure a Grant.

Non-Oracle ADF Application Requiring Fine-Grained Authorization

A non-Oracle ADF application requires fine-grained authorization checks. In this scenario:

- Place checks in the application code
- Configure the appropriate policies

For information about managing and checking policies programmatically, see The JAAS/OPSS Authorization Model.

Credentials

The following section describe a credential use case:

- Application Requiring Credentials to Access System - An Application requires credentials to access a back-end system.
- Application Requiring Credentials to Access System

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 Credential Store Framework API in application code
- Defining and enforcing system policies on codesource

> **See also:**
>
> Managing Credentials
>
> Provisioning Access Permissions
>
> Credential Store Framework API Examples
>
> Packaging Credentials with the Application

Audit

The following sections describe audit use cases:

- 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.
- Auditing Security-Related Activity
- Auditing Business-Related Activity

Auditing Security-Related Activity

The settings explained in this use case apply to all applications and components in a domain. Applications running in a domain require recording when policies, credentials, or keys are changed, and the policies evaluated in a particular time interval.

This use case features:

- Integrating applications with Oracle Platform Security Services Common Audit Framework.
- Defining application audit categories and events in security areas, and making the application audit-aware.
- Setting the appropriate filter level in each application.

> **See also:**
>
> Registering the Application with the Audit Service
>
> Logging Audit Events Programmatically

Auditing Business-Related Activity

The settings explained in this use case apply to all applications and components in a domain. Applications must 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.

To implement this scenario, you create audit events based on business needs, and the application logs the event activity in the audit store. In addition, you generate audit reports from audit events, manage runtime audit policies, and modify audit event definitions, as needed.

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, and deregistering applications after deployment.
- Managing audit artifacts with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) or WebLogic Scripting Tool (WLST) commands.

See also:

Registering the Application with the Audit Service

Logging Audit Events Programmatically

Creating Audit Definition Files

Managing Audit Policies

Identity Propagation

The following sections describe identity propagation use cases:

- Propagating the Executing User Identity
- Propagating a User Identity
- Propagating Identities Across Domains
- Propagating Identities over HTTP

- Propagating the Executing User Identity
- Propagating a User Identity
- Propagating Identities Across Domains
- Propagating Identities over HTTP

Propagating the Executing User Identity

A client application in a container requires propagating the executing user identity to a web service over the Simple Object Access Protocol (SOAP). The web service may be running in the same domain (same or different Managed Server), or in a different domain.

For information about OWSM, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Propagating a User Identity

A client application in container requires propagating a user identity (distinct from the executing user identity) to a web service over SOAP.

In this use case, the application gets the specific identity to propagate from the credential store and uses OWSM ability to propagate the identity to a remote server.

For information about OWSM, see *Securing Web Services and Managing Policies with Oracle Web Services Manager*.

Propagating Identities Across Domains

A client application in container requires propagating a user identity (stored in the security store) to a different WebLogic Server domain.

For information about domain trust, see Enabling Trust Between WebLogic Server Domains in *Administering Security for Oracle WebLogic Server*.

Propagating Identities over HTTP

A client application in container requires propagating identities over HTTP. The recommended implementation is to use the OPSS trust service.

> **See also:**
>
> Propagating Identities with the OPSS Trust Service
>
> Propagating Identities with JKS

Administration and Management

The following sections describe administration and management use cases:

- Application Requiring a Centralized Store
- Application Requiring a Custom Management Tool
- Application Running in a Multiple Server Environment

- Application Requiring a Centralized Store
- Application Requiring a Custom Management Tool
- Application Running in a Multiple Server Environment

Application Requiring a Centralized Store

An application requires a central repository of policies, credentials, audit configuration, trusts, and keys, and a set of tools to manage it. To implement this use case, use OPSS services and the tools to manage security services.

> **See also:**
>
> About the Security Store
>
> Managing Policies with Fusion Middleware Control
>
> Managing Policies with WLST
>
> Managing Credentials with Fusion Middleware Control
>
> Managing Credentials with WLST
>
> Managing Keys and Certificates

Application Requiring a Custom Management Tool

An application requires a custom tool to manage externalized security data. To implement this use case, create a custom graphical user interface that calls OPSS APIs to display and manage security data in a context meaningful to the application.

> **See also:**
>
> Implementing a Custom Graphical User Interface
>
> OPSS API References

Application Running in a Multiple Server Environment

An application running in a domain (where several server instances may be distributed across multiple machines) requires propagating security data changes when you initiate changes on the Administration Server and data on Managed Servers is refreshed based on caching policies. Changes must take effect in all components of the application regardless of where they are running. To implement this use case, use the OPSS MBeans or OPSS APIs to modify security data.

> **See also:**
>
> Environments with Multiple Servers
>
> Configuring Services with MBeans

Integration

A product requires multiple WebLogic Server domains to run and these domains must share a single central security store. To implement this use case, use the `reassociateSecurityStore` command to join to a store in some other domain, and the OPSS support for several domains to share a centralized security store.

Joining to a security store is supported only when you create a new domain.

> **See also:**
>
> reassociateSecurityStore
>
> Encrypting Credentials

The OPSS Trust Service

The OPSS trust service uses the Identity Asserter to provide and validate tokens and allows applications to propagate identities across HTTP-enabled applications.

> **See also:**
>
> Trust Service Properties
>
> updateTrustServiceConfig in *WLST Command Reference for Infrastructure Security*

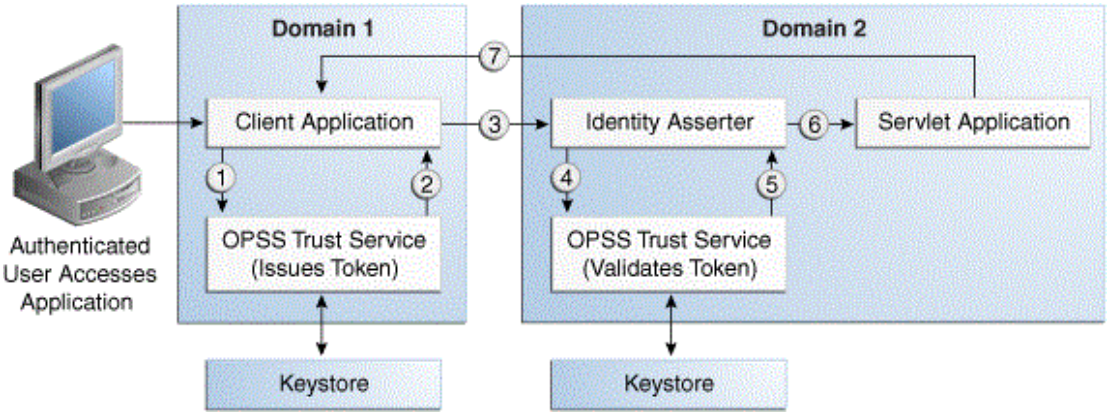
Propagating Identities over HTTP

> **Note:**
>
> Oracle recommends that you implement identity propagation using the OPSS trust service as explained in Propagating Identities with the OPSS Trust Service.

Figure 16-2 illustrates the typical flow of identity propagation using HTTP:

1. A client application in Domain1 requests a token for an authenticated user from Domain1's OPSS trust instance.
2. The service accesses Domain1's keystore and generates a token for the client application.
3. The client application encodes the token in an HTML header and dispatches an HTTP request to a Java servlet 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 instance.
5. The 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 Java servlet with the asserted identity.
7. The Java servlet sends an HTTP response to the client application request.

Figure 16-2 Identity Propagation over HTTP



The ready-to-use configuration sets the key alias based on the server name.

Propagating Identities with the OPSS Trust Service

To propagate identities across multiple domains or across containers in a single domain, Oracle recommends that you use the OPSS trust service as explained in the following sections:

- Propagating Identities Across Multiple WebLogic Server Domains
- Propagating Identities Across Containers in a Single WebLogic Server Domain
- Trust Provider Properties
- Propagating Identities Across Multiple WebLogic Server Domains
- Propagating Identities Across Containers in a Single WebLogic Server Domain
- Trust Provider Properties

Propagating Identities Across Multiple WebLogic Server Domains

In this use case there are two different WebLogic Server domains: Domain1 and Domain2. The client application is running in Domain1, and the Java servlet is running in Domain2. The client application uses Domain1's service for token generation, and the Java servlet uses Domain2's service for token validation.

The following sections include examples and configurations to implement this use case:

- Token Generation on the Client-Side Domain
- Server-Side or Token Validation Domain
- Token Generation on the Client-Side Domain
- Server-Side or Token Validation Domain

Token Generation on the Client-Side Domain

On the client side (Domain1) where the token is generated:

- Develop the Client Application
- Configure the Truststore
- Add a TrustServiceAccessPermission Grant
- Configure the Provider

Develop the Client Application

The client application can be a Java SE or Java EE application. The following example illustrates the 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());
```

Configure the Truststore

The truststore should be provisioned with a client certificate and a private key. The certificate is exported from the keystore and imported into the truststore. Both the keystore and the truststore, are managed by the keystore service (KSS) keystore in the client domain.

> **Note:**
>
> When you set the `trust.keystoreType` property to `KSS`, it is recommended *not* to use passwords for the keystore or the truststore. In this case, by default, the keystore and truststore are protected by codesource permissions and therefore 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="", algorithm="RSA")
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()
```

Add a TrustServiceAccessPermission Grant

The following grant (in the application `jazn-data.xml` file) illustrates a codesource permission that allows clients to use trust methods:

```
<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>
```

Add this grant to the security store with the `grantPermission` WLST command, or if the application is a Java EE application and the `jazn-data.xml` file is packed with the application, then have the grant migrated to the security store at application deployment.

> **See also:**
>
> grantPermission in *WLST Command Reference for Infrastructure Security*.
>
> Configuring the Filter and the Interceptor

Configure the Provider

The following example illustrates trust provider properties:

```
<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>
```

Server-Side or Token Validation Domain

On the server side (Domain2) where the token is validated:

- Develop the Server Application
- Configure web.xml
- Configure the Asserter
- Provision the Keystore
- Add Permissions
- Configure the Provider

Develop the Server Application

The Java servlet code first obtains the asserted user:

```
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();
}
```

Configure web.xml

Set the login configuration method to `CLIENT-CERT` in the `web.xml` file:

```
<web-app id="WebApp_ID"
…
 <login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>Identity Assertion </realm-name>
 </login-config>
…
</web-app>
```

Configure the Asserter

Configure the asserter in one of the following ways:

- Log in to Oracle WebLogic Server Administration Console and go to **Security Realms**, then to the **Providers** tab, then to **Authentication**, and then choose **TrustServiceIdentityAsserter**.

  This asserter calls trust methods to decode and validate the token from the incoming request, and passes the user name to WebLogic Server to establish the asserted subject.

- Use a script like the following:

  ```
  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()
```

Provision the Keystore

Export the client certificate you provisioned in the Domain1's keystore and import it into the Domain2's truststore, as illustrated in the following example.

> **Note:**
>
> Import the certificate in the keystore with an alias that matches the client name. In case of multiple domains, 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.

```
# 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 KSS
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()
```

Add Permissions

Use the `grantPermission` WLST command to add codesource grants to the application's `jazn-data.xml` file.

For example, the following codesource grant is required for the `MyApp` application to use trust methods:

```
<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>
```

> **See also:**
>
> grantPermission in *WLST Command Reference for Infrastructure Security*

Configure the Provider

The following example illustrates trust provider properties:

```
<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>
```

If the provider is used only for token validation, then the `aliasName` and `issuerName` attributes are not used for token validation and are therefore optional. In this case to validate a token, the provider looks for the certificate using the name included in the token.

Propagating Identities Across Containers in a Single WebLogic Server Domain

In this use case the client and server applications run in the same domain, 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.

Trust Provider Properties

To configure the trust provider, use the following properties:

- `trust.keyStoreType`
- `trust.keyStoreName`
- `trust.trustStoreName`
- `trust.aliasName`

- `trust.issuerName`
- `trust.provider.className`
- `trust.clockSkew`
- `trust.token.validityPeriod`
- `trust.token.includeCertificate`

The following example illustrates a trust provider configuration:

```
<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 modify the properties of the trust provider, use the `updateTrustServiceConfig` WLST command. For information about this command, see updateTrustServiceConfig in *WLST Command Reference for Infrastructure Security*.

Implementing a Custom Graphical User Interface

This section illustrates some of the operations needed when you implement, for example, a custom graphic UI to manage policies. The examples use the OPSS APIs and demonstrate how to:

- Queue users in the identity store.
- Queue application roles in the security store.
- Queue 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).
- Create, read, update, and delete the mapping of users and groups to application roles.

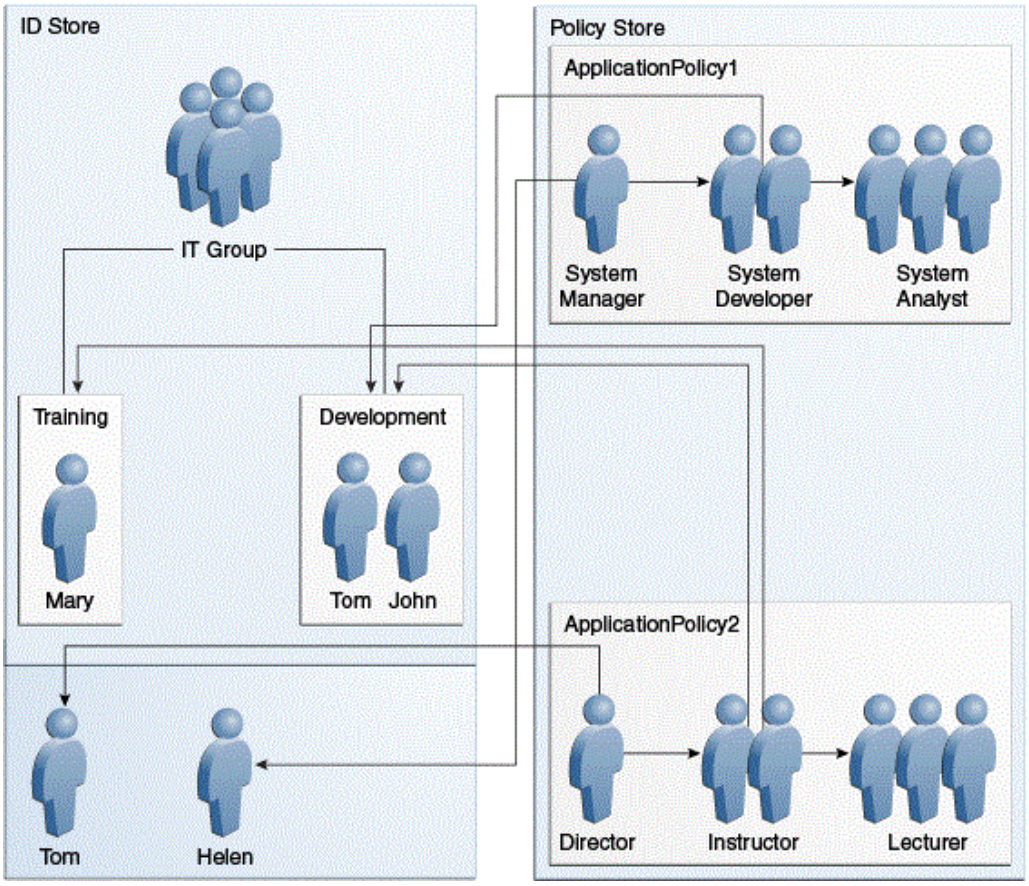
This use case assumes that:

- The identity store is an LDAP store.
- The security store is an LDAP store.
- The identity store contains the following hierarchy of users and groups:
  - The Mary, John, Tom, and Helen users.
  - The IT, Training, and Development groups.
  - The Training and Development groups, members of the IT group.
  - The Mary user, member of the Training group.

  - The Tom and John users, members of the Development development.
- The security store contains the following application policies and hierarchy of application roles:
  - The ApplicationPolicy1 and ApplicationPolicy2 application policies.
  - The System Manager, System Developer, and System Analyst roles, 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 Director, Instructor, and Lecturer roles are application roles referenced in the ApplicationPolicy2 policy. The Director role is a member of the Instructor role. The Instructor role is a member of the Lecturer role.
- Application roles are mapped to users and groups:
  - The System Manager role is mapped to the Helen user.
  - The System Developer role is mapped to the Development group.
  - The Director role is mapped to the Tom user.
  - The Instructor role is mapped to the Training and Development groups.

Figure 16-3 illustrates this hierarchy of application roles, users and groups, and the mapping of application roles to users and groups.

Figure 16-3 Mapping of Application Roles to Users and Groups



Note that this role hierarchy implies that a user in the System Manager role is also in the System Developer role, and similarly with the other roles. The role membership for each of the four users is next summarized:

- 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.

For the examples details, see the following sections:

- Imports Assumed
- Query Identity Store Example
- Create Role Example
- Query Roles Example
- Map Roles Example
- Get Roles that Contain a User Example

Imports Assumed

The examples 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;
```

Query Identity Store Example

The following example 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();
    }
```

Create Role Example

The following example 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();
    }
```

Query Roles Example

The following example 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();
    }
```

Map Roles Example

The following example 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();
        }
```

Get Roles that Contain a User Example

The following example illustrates how to get all the roles that contain a specified user:

```
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.
        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());
            }
        }
    }
```

Delete Role Mapping Example

The following example 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();
    }
```

Securing Oracle ADF Applications

The following sections explain the typical tasks performed in Oracle ADF applications developed with Oracle JDeveloper:

- Development Phase
- Deployment Phase
- Administration Phase

The participants are the application product manager, developers, and security administrators.

> **See also:**
>
> Summary of Tasks per Participant per Phase

- Development Phase
- Deployment Phase
- Administration Phase

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 JDeveloper, the built-in Oracle ADF framework, and Oracle WebLogic Server, all of which ensure a consistent approach to security throughout the application's life span.

You use ADF Security Wizard (an authorization editor) and the expression language editor, all within JDeveloper. Additionally and optionally, you 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 runtime environment.

You also define a number of application entitlements and roles required to secure the application, and this data is kept in a source control system together with the application source code.

Deployment Phase

After you have developed the application, you test it in a staging environment before deploying it to a production environment. In a production environment, the application and runtime services are integrated with other security components, such as single sign-on, user provisioning, and audit.

The type of security services usually change with the phase: for example, during development you keep credentials in a file or Oracle wallet, but in a production environment you store them in an LDAP server.

In the deployment phase, you migrate the policies to the production security store, and map application roles to enterprise groups according to application policies.

Administration Phase

The administration phase starts after you have deployed the application to a production environment. In this phase, you 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.

Summary of Tasks per Participant per Phase

The following tables summarize the major responsibilities per participant in each of the security life cycle phases and Figure 16-4 illustrates the basic flow.

Figure 16-4 Application Life Cycle Phases

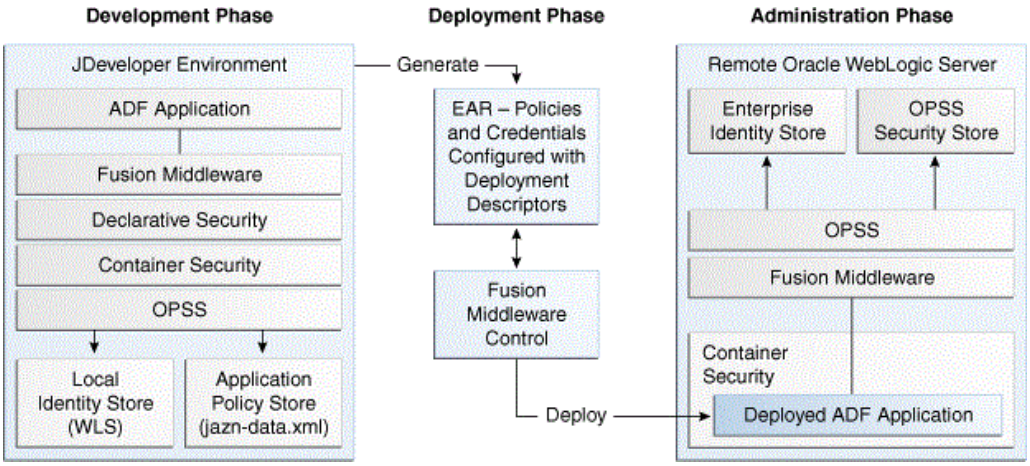


Table 16-1 Security Tasks for the Application Architect

Phase	Task
Development	Define high-level application roles based on functional security and data security requirements. Populate the initial security store.
Deployment	Define real-world customer scenarios for the QA (quality assurance) team to test.
Administration	Understand and identify the requirements to customize application policies. Consider defining templates for vertical industries.

Table 16-2 Security Tasks for the Application Developer

Phase	Task
Development	Use tools and processes, specifically JDeveloper, to build the application and to create security data, such as application roles and permissions. Use grants to specify data-level security. Test the application using a local security store with users and roles.
Deployment	Assist the QA team to troubleshoot and resolve runtime issues.

Table 16-3 Security Tasks for the Application Security Administrator

Phase	Task
Deployment	Use deployment services to migrate data in `jazn-data.xml` to the production security store. Map application roles to enterprise groups so that security policies can be enforced.
Administration	Apply patches and upgrades software, as necessary. Manage users and roles, as enterprise users and the application role hierarchy changes overtime. Manage policies packed with the application and creates new ones. Integrate with and manage the identity infrastructure.

Code and Configuration Examples

The following sections list code and configuration examples found elsewhere in this guide.

- Programming Examples
- Configuration Examples
- Programming Examples
- Configuration Examples

Programming Examples

The following topics include examples of typical security-related programming tasks:

- Querying the Identity Store Programmatically
- Implementing a Custom Graphical User Interface.
- Programmatic Authorization
- Managing Policies
- Checking Policies Programmatically
- The Class ResourcePermission
- Using the Identity Store Login Module for Authentication
- Using the Identity Store Login Module for Assertion

Configuration Examples

The following topics include examples of typical security-related configuration tasks:

- The Class ResourcePermission
- Configuring the Filter and the Interceptor
- Migrating Policies with migrateSecurityStore
- Migrating Credentials with migrateSecurityStore
- Configuring Single and Multiple LDAPs
- Configuring the LDAP Identity Store in Java SE Applications
- Authorization in Java SE Applications

Propagating Identities with JKS

The ready-to-use configuration sets the token name and the key alias based on the server name. To change these default values, use the procedures explained in Update Trust Parameters.

The following sections explain the propagation of identities over HTTP using Java Keystore (JKS):

- Single Domain Scenario
- Multiple Domain Scenario

Single Domain Scenario

In this scenario, the both client and Java servlet use the same service to generate and validate tokens. The following sections explain the tasks necessary to implement identity propagation when the client and Java servlet run in the same domain:

Create the Client Application

The following example illustrates a client application. Note that the `jps-api.jar` file and the `osdt_ws_sx.jar`, `osdt_core.jar`, `osdt_xmlsec.jar`, and `osdt_saml2.jar` files must be included in the class path.

```
// 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());
```

Configure the Keystore

Assuming that the server name is `jrfServer_admin`, the following command illustrates the creation of the keystore, represented by the generated `default-keystore.jks` file:

```
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 `jps-config.xml` file points to the generated `default-keystore.jks`:

```
<!-- 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 >
```

Configure Maps and Keys

Create a map/key pair used to open the keystore and another map/key pair used to generate tokens. The following commands illustrate these operations with the `createCred` WLST command:

```
// 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 information about `createCred`, see Managing Credentials with WLST.

Configure a Grant

Add a system policy with a codesource grant, which allows the client application to use trust methods:

```
<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>
```

You must stop and restart WebLogic Server for the grant to take effect.

Create the Java Servlet

The following example illustrates how a Java servlet obtains 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();
}
```

Configure web.xml

Set the appropriate login method in the `web.xml` file:

```
<web-app id="WebApp_ID"
…
 <login-config>
   <auth-method>CLIENT-CERT</auth-method>
   <realm-name>Identity Assertion</realm-name>
 </login-config>
…
</web-app>
```

Configure the Asserter

To configure the asserter:

1. Copy the `jps-wls-trustprovider.jar` identity asserter file to the location `${domain.home}/lib/mbeantypes`:

   ```
   cp ${common.components.home}/modules/oracle.jps_12.2.1/jps-wls-trustprovider.jar ${domain.home}/lib/mbeantypes
   ```

2. Restart WebLogic Server.
3. Use WebLogic Server Administration Console to configure the asserter:
   a. Log in as an administrator and go to **Security Settings**, then to **Security Realms**, then to **Providers**, then to the **Authentication** tab, and then click **New.** The **Create a New Authentication Provider** dialog is displayed.
   b. In this dialog, enter `TrustServiceIdentityAsserter` in the name text field, and choose `TrustServiceIdentityAsserter` from the pull-down list. Then click **OK**.
4. Verify that a grant like the following is present in the security store. This grant is required so that the asserter can use OPSS trust methods:

   ```
   <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>
   ```

   Changes to the `jps-config.xml` file require that you restart the server.

Update Trust Parameters

This section explains how to modify the trust parameters in the `jps-config.xml` file by executing a script.

By default, the `trust.aliasName` and `trust.issuerName` parameters are set to the server name. To modify these values, adapt and use the following script:

```
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"
```

Multiple Domain Scenario

In this scenario there are two different domains: Domain1 and Domain2. The client application is running in Domain1, and the Java servlet is running in Domain2. It is assumed that these two domains have each a trust service properly configured as explained in the Single Domain Scenario. The client uses Domain1's trust service for token generation, and the Java servlet uses Domain2's trust service for token validation.

In Domain1, the client code and the following configurations are identical to those described in the Single Domain Scenario:

- The client application as illustrated in Create the Client Application.
- The configuration of the keystore as illustrated in Configure the Keystore.
- The Credential Store Framework configuration as illustrated in Configure Maps and Keys.
- The grant configuration as in Configure a Grant.

In Domain 2, the Java servlet code and `web.xml` configuration are identical to those described in the Single Domain Scenario:

- The Java servlet code as illustrated in Create the Java Servlet.
- The configuration of the `web.xml` file as illustrated in Configure web.xml.
- The client certificate used to sign the token in Domain1 must be present in Domain2's keystore. To comply:

  1. Export the certificate from Domain 1's keystore:

     ```
     JAVA_HOME/bin/keytool -export
     -orakey orakey.cer
     -keystore default-keystore.jks
     -storepass password
     ```

  2. Import the certificate into Domain 2's keystore. Note that the alias used to import the certificate must match the name of the on the client side:

     ```
     JAVA_HOME/bin/keytool -importcert
       -alias orakey
       -keypass welcome
       -keyalg RSA
       -keystore default-keystore.jks
       -storepass password
     ```

  3. Set the Domain2's keystore password in the (Domain2's) credential store with the `createCred` WLST command:

     ```
     createCred(map="oracle.wsm.security", key="keystore-csf-key", user="keystore",
     password="password")
     ```

     See createCred in *WLST Command Reference for Infrastructure Security*.

Domains Using Both Protocols

In this scenario, applications use either HTTP or SOAP, and not all applications in the domain use the same protocol. In such scenario, the keystore can be shared by HTTP and SOAP services.

The following sections explain the special configurations required in this case:

- Single Domain Scenario
- Multiple Domain Scenario
- Single Domain Scenario
- Multiple Domain Scenario

Single Domain Scenario

In this scenario, there is one keystore. The following example illustrates the configuration required for a certificate with `orakey` alias:

```
<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 add in the token used by the target
    trust service instance as an alias to pick up the corresponding certificate
    to validate the token signature -->
    <property name="trust.issuerName" value="orakey"/>
</propertySet>
```

Multiple Domain Scenario

In this scenario, there are two domains and two keystores. The following example illustrates the configuration required in the domain that is issuing tokens for a certificate with `orakey` alias:

```
<!-- issuer domain truststore 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>
```

On the client side, the value of `trust.issuerName` can be same as `trust.aliasName`. However the name value can be overridden by setting a different value for `trust.issuerName` (as shown in the example). This name will be set in the token generated on the client side.

On the server side, if the server is used only for token validation, then it is not mandatory to set `trust.aliasName` and `trust.issuerName`. The 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 name as the alias (domain1 in the example).

The following example illustrates the configuration required in the domain that is receiving tokens for a certificate with `orakey` alias:

```
<!- the 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 Security Model

This chapter describes the OPSS authorization and policy models, and compares them with the Java EE and Java Authorization and Authentication Services (JAAS) authorization models.
This chapter includes the following sections:

- About the OPSS Authorization and Policy Models
- Authorization Models
- The JAAS/OPSS Authorization Model

About the OPSS Authorization and Policy Models

For information about the OPSS authorization and policy models, see *Administering Oracle Entitlements Server*.

Authorization Models

A policy specifies the permissions granted to code loaded from a given location. The JAAS model extends policies by allowing an optional list of principals. These policies grant permissions to code from a specified location that is run by any of those principals.

The OPSS model is based on the JAAS model and, moreover, allows application policies and roles, and system policies. Application roles can be mapped to enterprise users and groups (such as administrative roles). A policy can grant permissions to any of these roles, groups, or principals.

A Java EE application can delegate authorization to the container where it runs, or it can implement its own authorization with calls to methods such as `checkPermission`, `checkBulkAuthorization`, or `getGrantedResources`.

The following sections describe the main points of the Java EE and JAAS authorization models:

- The Java EE Authorization Model
- The JAAS Authorization Model

The Java EE Authorization Model

The Java EE authorization model uses role membership to control access to Enterprise JavaBeans (EJB) and web resources that are referenced by URLs. Policies assign permissions to users and roles, and they are enforced by the container.

In the Java EE model, authorization is implemented in either of the following ways:

- Declaratively, where 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 policies are processed 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, then 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 specified at the URL level or at the EJB method level.
Programmatic	Specified in application code. Provides fine-grained authorization.	Not so easy to update, because it involves code changes and recompilation.

A container can provide authorization to applications running in it in two ways: declaratively and programmatically, as explained in the following sections:

- Declarative Authorization
- Programmatic Authorization
- Java EE Application Example

- Declarative Authorization
- Programmatic Authorization
- Java EE Application Example

Declarative Authorization

Declarative authorization allows you to control access to URL-based resources (such as Java servlets and pages) and EJB methods.

To configure declarative authorization:

1. Specify (in standard deployment descriptors) the resource to protect and a role that has access to that resource. Alternatively, use code annotations.
2. Map the role to an enterprise group (in proprietary deployment descriptors, such as the `web.xml` file).

Programmatic Authorization

Programmatic authorization provides a fine-grained authorization not available in declarative approach, and it requires that the application code call the `isUserInRole` method (for Java servlets) or the `isCallerInRole` method (for EJB), both available from standard Java APIs.

Although these methods still depend on role membership to determine authorization, they give finer control over authorization decisions because the controlling access is not limited to EJB or URL.

Java EE Application Example

The following example illustrates an application calling the `isUserInRole` method. The example assumes that the application Enterprise ARchive (EAR) file includes the `web.xml` and `weblogic-application.xml` files, and that these files include the following specifications:

```
<!-- security roles in web.xml -->
 <security-role>
   <role-name>sr_developer</role-name>
 </security-role>

<!-- maaping of user to role in weblogic.application.xml -->
<wls:security-role-assignment>
  <wls:role-name>sr_developer</wls:role-name>
  <wls:principal-name>weblogic</wls:principal-name>
</wls:security-role-assignment>
```

Code Calling 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>");
        }
    }
```

The JAAS Authorization Model

The JAAS authorization model introduces permissions but also uses roles. In this model, a policy binds permissions with a subject (role, group, or user) and, optionally, with code. You grant permission to a role by calling the `addPrincipalsToAppRole` method. Permissions are evaluated by calls to the static `AccessController.checkPermission` method. The model allows a high control of resources.

In a policy you specify the following data:

- Application roles and enterprise groups allowed the permission(s).
- Permissions (in application policies) and codesources (in system policies). Application polices define what a user or the member of a group is allowed to access. System policies define what actions the code is allowed to perform.

When you program with this model, you precede sensitive parts of the your application with checks that determine whether the current user or role has the appropriate permissions to the code, and the code is run if the user has the right permissions. For an example, see Using Supported Permission Classes.

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.

The following sections explain the OPSS authorization model:

- The Resource Catalog
- Managing Policies
- Checking Policies Programmatically
- The Class ResourcePermission

The Resource Catalog

OPSS supports the specification and runtime support of the resource catalog in security stores.

The resource catalog allows you to:

- Describe policies and secured artifacts in human-readable terms.
- Define and modify policies independently of the application source code.

- Browse and search policies, roles, and the role hierarchy.
- Group permissions in entitlements.

Managing Policies

Manage the resource catalog with the following interfaces, all sub-interfaces of `oracle.security.jps.service.policystore.EntityManager`:

- `GrantManager` - Use this interface 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` - Use this interface to create, modify, and query permission entitlements.
- `ResourceManager` - Use this interface to create, delete, and modify resource instances.
- `ResourceTypeManager` - Use this interface to create, delete, modify, and query resource types.

To create a resource type, a resource instance, actions, or a permission set, use code like the following:

```
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 example 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 example 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);
```

Checking Policies Programmatically

When you check policies programmatically, keep in mind the following points:

- By default, authorization failures are not visible in the console. To have authorization failures sent to the console, set the `jps.auth.debug` system variable: `-Djps.auth.debug=true`.

  In particular, to have `JpsAuth.checkPermission` failures sent to the console, you must set that variable.

- The policy provider must be explicitly set in Java SE applications:

```
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.

The following sections illustrate the use of several methods to check policies programmatically:

- Using checkPermission
- Using doAs and doAsPrivileged
- Using checkBulkAuthorization
- Using getGrantedResources

Using checkPermission

Oracle Fusion Middleware supports the `checkPermission` method in the `java.security.AccessController` and `oracle.security.jps.util.JpsAuth` classes.

Oracle recommends the use of `checkPermission` in the `JpsAuth` class because it provides better debugging support, better performance, and audit support.

The static `AccessController.checkPermission` method uses the default access control context (the context inherited when the thread was created). To check permissions on some other context, call the instance `checkPermission` method on a particular `AccessControlContext` instance.

The following table describes the behavior of `checkPermission` according to the value of the JAAS mode:

Table 17-2 checkPermission Behavior According to JAAS Mode

JAAS Mode	checkPermission Behavior
`off` or `undefined`	Enforces codesource security based on the security policy in effect, and there is no provision for subject-based security.
`doAs`	Enforces a combination of codesource and subject-based security using the access control context created in 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 codesource) when evaluating a permission.

> **Note:**
>
> If you call `checkPermission` inside a `doAs` block and the check permission call fails, then to display the failed protection domain you must set the `java.security.debug=access,failure` system property.

The following example illustrates an application checking a permission. It assumes that the application EAR file includes the configuration `jazn-data.xml` and `web.xml` files.

jazn-data.xml

```
<?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 -->
        <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 following example illustrates the `JpsFilter` filter configuration:

```
<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>...
```

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();
        }
    }
}
```

Using doAs and doAsPrivileged

Oracle Fusion Middleware supports the `doAs` and `doAsPrivileged` methods in the `javax.security.auth.Subject` and `oracle.security.jps.util.JpsSubject` classes.

Oracle recommends you use the `oracle.security.jps.util.JpsSubject` class because it renders better performance and provides audit.

> **Note:**
>
> If you call `checkPermission` inside a `doAs` block and the check permission call fails, then to display the failed protection domain you must set the `java.security.debug=access,failure` system property.

Using checkBulkAuthorization

The `checkBulkAuthorization` method determines whether a subject has access to one or more resource actions. This method returns the set of resource actions the subject is authorized on the resources. Grants using resources must include the resource type.

When you call this method, make sure that:

1. You have set the `java.security.policy` system property to the location of the Oracle WebLogic Server policy file.
2. Your application calls `checkBulkAuthorization` *after* `setPolicy`:

```
java.security.Policy.setPolicy(new
oracle.security.jps.internal.policystore.JavaPolicyProvider())
```

`checkBulkAuthorization` assumes that:

- The caller can provide a subject with user and enterprise role principals, and a list of resources including the stripe each resource belongs to.
- The application can access the application stripes configured in the domain where the application is running.

Using getGrantedResources

The `getGrantedResources` method 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. This method returns only permissions associated with resource types and is available only when for LDAP security stores.

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 gives you complete control over the actions, target matching, and 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 it is 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 `oracle.security.jps.ResourcePermission` class that you use as the permission class within any application grant to protect application or system resources. In this way, you no longer need to write custom permission classes and can readily use that class in permissions within application grants stored in any supported policy provider. Do not use this class in system policies, but use it only in application policies.

Configuring Resource Permissions

A permission that uses the `ResourcePermission` class is called a *resource permission*, and it specifies the resource type, the resource name, and an optional list of actions:

```
<permission>
  <class>oracle.security.jps.ResourcePermission</class>
  <name>resourceType=type,resourceName=name</name>
```

```
  <actions>character-separated-list-of-actions</actions>
</permission>
```

Even though the resource type information is not used at runtime, the resource type definition is required.

The following examples illustrate the specifications of resource permissions, which include the 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>
```

A resource type associated with a resource permission can have an empty list of actions. Note the following points about resource permissions:

- The name must conform to the following format:

  `resourceType=`*`aType`*`,resourceName=`*`aName`*

  You must define the resource type of a resource permission. To obtain the type of a resource, use the `ResourcePermission.getType` method.

- The character-separated list of actions is optional. If specified, then it must be a subset of the actions specified in the associated resource type. The method ResourcePermission.getActions returns this list.

  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 `ResourcePermission.getResourceName` method returns the display name of a resource used in a permission.

- Wildcard characters are not supported in resource permissions.

Managing and Checking Resource Permissions

The following lines illustrate how to create a resource permission and how to check it:

```
ResourcePermission rp =
   new ResourcePermission("oracle.bi.publisher.Reports","GLReps","develop");
JpsAuth.checkPermission(rp);
```

The permission check succeeds if the resource permission satisfies the following conditions:

- The permission is an instance of the `ResourcePermision` class.
- The resource type name matches (ignoring case) the name of a resource type.
- The resource name matches exactly the name of a resource instance.
- The list of actions is a comma-separated subset of the set of actions specified in the resource type.

About the Class for a Resource Type

When you create a resource type, optionally specify a class. If unspecified, then it defaults to the `oracle.security.jps.ResourcePermission` class.

If two or more resource types share a class, then that class must be one of the following:

- The `oracle.security.jps.ResourcePermission` class.
- A concrete class extending the `oracle.security.jps.AbstractTypedPermission` abstract class, as illustrated by `MyAbstractTypedPermission`:

  ```
  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 `oracle.security.jps.TypePermission` class and extending the `java.security.Permission` class.

18

Developing with the Credential Store Framework

This chapter explains how to use the Credential Store Framework in your applications and describes guidelines for the credential store configuration.
This chapter includes the following topics:

- About the Credential Store Framework API
- Guidelines for Using the Credential Store Framework API
- About Map and Key Names
- Provisioning Access Permissions
- Using the Credential Store Framework API
- Credential Store Framework API Examples

About the Credential Store Framework API

You use the Credential Store Framework API to access, retrieve, and manage credentials kept in the credential store. This APIs allow you to:

- Check whether a credential map or a map and key is stored in the credential store.
- Obtain credentials associated within a map or a map and key.
- Assign credentials to a a map or to a map and key.
- Delete credentials within a map or a map and key.

Operations on the credential store are secured by the `CredentialAccessPermission` class, a class implementing the fine-grained control used by the credential framework.

> **See also:**
>
> Managing Credentials

Guidelines for Using the Credential Store Framework API

When you develop applications that use the Credential Store Framework API, make sure that you:

- Provision security policies that enable applications access to credentials.
- Determine appropriate map and key names to use, specially in environments where multiple applications use the same credential store.
- Make sure that a credential store instance is defined and properly configured in the `jps-config.xml` file.

See also:

Provisioning Access Permissions

About Map and Key Names

Using the Credential Store Framework API

Credential Store Framework API Examples

About Map and Key Names

Each application must have a unique map name associated with it in the credential store. This guarantees that no conflicts will arise between the various map and key names in the store, and that the map name identifies the application unambiguously. Within a given map, an application can store multiple keys each of which also has a unique name, so that the pair map name/key name identifies a single key in the credential store.

Provisioning Access Permissions

The credential framework secures access to maps, all keys within a map, and to specific keys within a map. To use the Credential Store Framework API you must specify access permissions that allow your application to use the API. Moreover, any code calling this API also requires a codesource permission, but these permissions are typically restricted to specific jars only. It is not recommended that you define access permissions to *all* maps and keys.

The following sections illustrate access permissions:

- Permission to Access a Key Example
- Permission to Access a Map Example
- Permission to Access a Key Example
- Permission to Access a Map Example

Permission to Access a Key Example

The following example shows an access permission to a source code to perform any action on a specific key within a map:

```
<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=myKey</name>
              <actions>*</actions>
            </permission>
        </permissions>
    </grant>
</jazn-policy>
```

Permission to Access a Map Example

The following example shows an access permission to a source code to perform specific actions to a map and all keys in that map:

```
<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>
              <actions>read,write,update,delete</actions>
        </permission>
        </permissions>
    </grant>
</jazn-policy>
```

Using the Credential Store Framework API

The following sections explain how to use this framework in Java SE and Java EE applications:

- Using the Credential Store Framework API in Java SE Applications
- Using the Credential Store Framework API in Java EE Applications
- Using the Credential Store Framework API in Java SE Applications

- Using the Credential Store Framework API in Java EE Applications

Using the Credential Store Framework API in Java SE Applications

To use the Credential Store Framework API in Java SE applications:

1. Ensure that the `jps-manifest.jar` file is in your class path.
2. Provide permissions to access Credential Store Framework APIs.
3. Set Java Virtual Machine (JVM) options as appropriate. Options include the following:
   - `-Doracle.security.jps.config`

     specifies the full path to the `jps-config-jse.xml` file, if different from the default location (`$DOMAIN_HOME/config/fmwconfig/jps-config-jse.xml`).
   - `-Djava.security.policy`

     specifies the location of the `weblogic.policy` file, if different from the default location (`$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.

See also:

Provisioning Access Permissions

Using OPSS in Java SE Applications

Using the Credential Store Framework API in Java EE Applications

To use the Credential Store Framework API in a Java EE application, provide the access permissions necessary for your application to work before deploying it to Oracle WebLogic Server.

See also:

Provisioning Access Permissions

Credential Store Framework API Examples

The following examples illustrate how credential store operations use the required access permissions:

Credential Store Framework Operations Example

The following example illustrates Credential Store Framework API operations that are used in by other examples:

```
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 because 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 because 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. 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();
    }

    /*
     * because the following performs a privileged operation, only
     * jar containing this class needs CredentialAccessPermission
     */
    public void doPrivilegedCredOperation() {
        AccessController.doPrivileged(new PrivilegedAction<String>() {
                public String run() {
                    doOperation();
                    return "done";
                }
        };
    }
}
```

Java SE Application with File Credentials Example

The example in this section illustrates a Java SE application that uses a file credential store represented by the `$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` file.

In the example, the `projectsrc.home` system property points to the directory containing the Java SE application, and `clientApp.jar` is the application JAR file which is present in the `dist` directory.

The following grant illustrates access permissions:

```
<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>
```

Because no permission is granted to `mapName=pc_map2,keyName=pc_key2`, the call to `setCredential` for that map and key will fail.

The credential store used by the application is specified in `jps-config-jse.xml`:

```
    <serviceInstances>
        <serviceInstance name="credstore_file_instance"
                         provider="credstore_file_provider">
            <property name="location" value="store" />
```

```
        </serviceInstance>
    </serviceInstances>
```

Here is the Java SE code that calls the 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);
            // next call is in a doPrivileged block and should succeed
            csf.doPrivilegedCredOperation();

            // because next call is not in a doPrivileged block,
            // it fails if CredentialAccessPermission is not granted to this
class
            csf.doCredOperation();
        } catch (JpsException e) {
            e.printStackTrace();
        }
   }
}
```

Java EE Application with File Credentials Example

This example shows a Java EE application using file credentials that calls the Credential Store Framework API. The `jazn-data.xml` file defines the appropriate access permissions, the codesource permissions, the permissions required for different combinations of map and key.

The following grant illustrates access permissions:

```
<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>
```

The credential store used by the application is specified in the `jps-config.xml` file:

```
    <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>
```

The `location` property specifies the location of the `cwallet.sso` file.

Here is the example using these configurations:

```
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>");

            //get hold of app-level CSF service store
            //Outside app context, it returns the domain CSF store
            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 create operation is implemented inside a privileged block. Note that in a Java SE environment, the following two calls are equivalent:

```
CredentialStore store =
JpsServiceLocator.getServiceLocator().lookup(CredentialStore.class);

CredentialStore store =
JpsContextFactory.getContextFactory().getContext().getServiceInstance(CredentialS
tore.class);
```

Java EE Application with LDAP Store Example

The following example uses the same application used in Java EE Application with File Credentials Example, but the credential store is now LDAP instead of a file.

Here is an example of an LDAP store configuration:

```
    <serviceProviders>
        <serviceProvider name="ldap.credentialstore.provider"
 class="oracle.security.jps.internal.credstore.ldap.LdapCredentialStoreProvider">
            <description>Prototype LDAP 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>
```

Java EE Application with DB Store Example

The following example uses the same application used in Java EE Application with File Credentials Example, but the credential store is now a database instead of a file.

Here is a example of a DB store configuration:

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

19

Developing with the User and Role API

This chapter explains how to use the User and Role API to access, search, and modify entries in the identity store, and how to configure Secure Sockets Layer (SSL) with LDAP providers.
It includes the following sections:

- About the User and Role API
- Working with Service Providers
- Searching the Identity Store
- Creating and Modifying Entries in the Identity Store
- User and Role API Examples
- Configuring SSL for LDAP Providers

About the User and Role API

> **Note:**
>
> The User and Role API is deprecated. Oracle recommends that you use instead the Identity Governance Framework and migrate usage to this framework. For information about this migration, see Migrating to Identity Directory API in *Developing Applications with Identity Governance Framework*.

The User and Role API allows applications to access identity information in a uniform and portable way regardless of the particular underlying identity repository. This repository can be an LDAP server, a database, a file, or some custom repository.

The User and Role API provides programmatic access to any repository, ensures portability, and helps you simplify application. For example, using this API, your application can access several repositories without requiring any changes to the application code.

This API includes methods to create, update, and delete users and roles, and search them for attributes; it allows you, for example, to obtain the email addresses of all users in a certain role.

To use the User and Role API from a Java container, the identity store must be LDAP and the Administration Server must be up and running. In addition, application role members must use the `weblogic.security.principal.WLSUserImpl` class.

Oracle recommends that you authenticate users with an authentication provider and that do not use User and Role API for that purpose, and that you do not use concurrently the User and Role API and other APIs accessing entries in the same LDAP server.

> **See also:**
>
> Authentication Providers and the User and Role API
>
> *Developing Security Providers for Oracle WebLogic Server*
>
> *Java API Reference for Oracle Platform Security Services User and Role*

Authentication Providers and the User and Role API

The User and Role API uses, by default, the first authentication provider configured in the domain. If your application requires using any other configured authentication provider, then configure this special use as explained in Working with Service Providers.

When more than one provider is configured in the environment, you specify providers in an ordered list and set a control flag in each of them. Using this order and the control flags, the server determines which provider to use. After one is chosen, the remaining providers are ignored.

Working with Service Providers

To implement a provider, choose the provider class appropriate to the underlying repository, configure that provider, and then configure the provider runtime, as explained in the following sections:

> **See also:**
>
> Identity Assertion Providers in *Developing Security Providers for Oracle WebLogic Server*

Setting Up the Environment

The User and Role API interacts with the identity repository through an identity provider, which carries out the actual communication with the underlying repository. This offers flexibility because the same code can be used with different repositories by modifying the provider's connection information.

To configure your environment to use the User and Role API:

- Ensure that the provider JAR file, which implements the underlying particular identity repository, and component JARs required by your provider are available in your environment.
- Specify the object classes that the search method use:

```
     <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>
```

- In case of an LDAP provider, configure the provider user so that it has permissions to read the `cn=common,cn=products,cn=oraclecontext` nodes.

> **See also:**
>
> Configuring Security Providers with Fusion Middleware Control

Choosing the Provider Repository

OPSS supports a number of user repositories for an identity service provider. For systems and versions, see Oracle Fusion Middleware Supported System Configurations.

The choice of repository determines the provider class to use with the provider, as described in the following table:

Table 19-1 Repository and Provider Classes

Repository	Provider Class
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
Embedded LDAP server	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

The provider class must implement the interface specified by the User and Role API framework. For information about this API, see *Java API Reference for Oracle Platform Security Services*.

Creating the Provider Instance

To create a provider instance after having identified the provider's class:

1. Use the `IdentityStoreFactoryBuilder.getIdentityStoreFactory` method to create a factory instance:

   ```
   IdentityStoreFactoryBuilder builder =new IdentityStoreFactoryBuilder ();
   ```

2. Use the `IdentityStoreFactory.getIdentityStoreInstance` method to create a store instance:

   ```
   IdentityStoreFactory oidFactory = builder.getIdentityStoreFactory(
      "oracle.security.idm.providers.oid.OIDIdentityStoreFactory", factEnv);
   ```

3. Obtain the handle to the identity store:

   ```
   oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
   ```

Configuring the Provider Start-Time and Runtime Properties

You can set a number of properties for the factory instance and the store instance, such as the URL, the repository port number, and the user and password to access the repository.

The following sections explain how to set these properties:

- Configuring Start-Time and Runtime Properties
- Enabling Execution Context ID

Configuring Start-Time and Runtime Properties

To configure the provider properties at start-time (when you create the provider) or at runtime, use properties including:

- Start-time properties, with names prefixed with ST_.
- Runtime properties, with names prefixed with RT_.

Start-time Configuration Properties

You configure start-time provider properties once because generally these values persist for the duration of the provider's lifetime.

Specify the property ST_SUBSCRIBER_NAME when you create the store instance, and all other start-time properties when you create the provider factory instance.

The following table describes the start-time identity provider properties.

Table 19-2 Start-Time Identity Provider Properties

Property Name	Description
ST_BINARY_ATTRIBUTES	The names of binary attributes stored in the LDAP server. The provider treats these attributes as binary data while sending and receiving data from the LDAP server.
ST_CONNECTION_POOL	The external connection pool, an instance of the `oracle.idm.connection.ConnectionPool` class. 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 provider user's name in the identity store.
ST_GROUP_NAME_ATTR	The provider user's role name in the identity store.
ST_USER_LOGIN_ATTR	The login ID of the provider's user in the identity store.
ST_SECURITY_PRINCIPAL	The user (principal).
ST_SECURITY_CREDENTIALS	The credentials to log in to the identity store.
ST_LDAP_URL	The URL of the identity store.
ST_MAX_SEARCHFILTER_LENGTH	The maximum length of the search filter allowed by the LDAP server.

Table 19-2 (Cont.) Start-Time Identity Provider Properties

Property Name	Description
ST_LOGGER	The logger that the API uses.
ST_SUBSCRIBER_NAME	The base distinguished name in the LDAP server. This property is specified when you create the IdentityStore instance and is used to determine default values for the remaining properties.
ST_CONNECTION_POOL_CLASS	The fully-qualified connection pool class name.
ST_INITIAL_CONTEXT_FACTORY	The fully-qualified class name of the factory that creates the initial context.

Runtime Configuration Properties

Runtime properties control the behavior of the provider's `IdentityStore` instance. You configure these properties by the specifying `StoreConfiguration` object that is obtained from the `IdentityStore` instance. All runtime properties have default values.

The following table describes the runtime identity provider properties.

Table 19-3 Runtime Identity Provider Properties

Property Name	Description
RT_USER_OBJECT_CLASSES	An array of classes required to create a user.
RT_USER_MANDATORY_ATTRS	Required attributes of a new user.
RT_USER_CREATE_BASES	Base DNs where a new user can be created.
RT_USER_SEARCH_BASES	Base DNs that can be searched for users.
RT_USER_FILTER_OBJECT_CLASSES	An array of classes used to search users.
RT_GROUP_OBJECT_CLASSES	An array of classes required to create a role.
RT_GROUP_MANDATORY_ATTRS	Required attributes of a new role.
RT_GROUP_CREATE_BASES	Base DNs where a new role can be created.
RT_GROUP_SEARCH_BASES	Base DNs that can be searched for roles.
RT_GROUP_MEMBER_ATTRS	An array of attributes in a role. All members of a role have a value for each of these attributes.
RT_GROUP_FILTER_OBJECT_CLASSES	An array of classes used to search roles.
RT_USER_SELECTED_CREATE_BASE	The distinguished name where a user is created with `createUser`. If null and the ST_SUBSCRIBER_NAME is unspecified, then the first supplied value of the RT_USER_CREATE_BASE is used. If null and the ST_SUBSCRIBER_NAME is specified, then the subscriber name of the identity store is used.
RT_GROUP_SELECTED_CREATE_BASE	The distinguished name where a role is created with `createRole`. If null and the ST_SUBSCRIBER_NAME is unspecified, then the first supplied value of the RT_GROUP_CREATE_BASE is used. If null and the ST_SUBSCRIBER_NAME is specified, then the subscriber name of the identity store is used.

Table 19-3 (Cont.) Runtime Identity Provider Properties

Property Name	Description
RT_GROUP_GENERIC_SEARCH_BASE	The distinguished name where to search roles for a given identity. In LDAP providers, it is set, by default, to the subscriber name or else to the first group search base.
RT_SEARCH_TYPE	The type of search: SIMPLE, PAGED, or VIRTUAL_LIST_VIEW.

Enabling Execution Context ID

ECID enabling is required only for LDAP identity stores. By default, ECID support is disabled in the User and Role API. To enable it, set the `ST_ECID_ENABLED` property to `true` when initializing the API:

```
factEnv.put(OVDIdentityStoreFactory.ST_ECID_ENABLED, "true");
```

Configuring the Provider when Creating a Factory Instance

Configuration at this stage affects the entire factory object as well as objects created using the specific factory instance. Many start-time properties are set at this time, including `ST_LDAP_URL`, `ST_SECURITY_PRINCIPAL`, and `ST_SECURITY_CREDENTIAL`.

The following sections illustrate how to configure provider properties when you create a factory instance:

- Configuring Common Properties
- Configuring Constants, Number of Connections, and Pool Connection
- Configuring Common Properties
- Configuring Constants, Number of Connections, and Pool Connection

Configuring Common Properties

The following example illustrates a provider configuration at the time the LDAP factory is created, including the specification of a log location:

```
IdentityStoreFactoryBuilder builder = new IdentityStoreFactoryBuilder();
IdentityStoreFactory oidFactory = null;
Hashtable factEnv = new Hashtable();
Logger mylogr = Logger.getLogger("mylogger.abc.com");
FileHandler fh = new FileHandler("userroleapi.log");
mylogr.addHandler(fh);
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/");
factEnv.put(OIDIdentityStoreFactory.ST_LOGGER_NAME, "mylogger.abc.com");
oidFactory = builder.getIdentityStoreFactory(
   "oracle.security.idm.providers.oid.OIDIdentityStoreFactory", factEnv);
```

Configuring Constants, Number of Connections, and Pool Connection

To overwrite constants and default property values, supply a map in the `ST_PROPERTY_ATTRIBUTE_MAPPING` property during factory creation.

The following example illustrates how to set the maximum and minimum number of connections, the custom connection pool class, and the map of the `RoleProfile.OWNER` user to the `myowner` attribute:

```
factEnv.put(LDIdentityStoreFactory.ST_CONNECTION_POOL_MIN_CONNECTIONS,  "3");
factEnv.put(LDIdentityStoreFactory.ST_CONNECTION_POOL_MAX_CONNECTIONS, "16");

factEnv.put(OIDIdentityStoreFactory.ST_CONNECTION_POOL_CLASS,
"oracle.security.idm.providers.stdldap.JNDIPool");

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);
```

All owner-related operations, such as `getOwners`, and `getOwnedRoles` are performed using the `myowner` attribute. The maximum minus the minimum connection values must be greater than 10.

Configuring the Provider when Creating a Store Instance

The identity store configuration applies to the store and all objects created with the store instance, and you specify the runtime property you want when you create the identity store instance.

The following example illustrates how to create a store instance and how to get a handle for it:

```
IdentityStore oidStore = null;
Hashtable storeEnv = new Hashtable();
storeEnv.put(OIDIdentityStoreFactory.ST_SUBSCRIBER_NAME,"dc=us,dc=oracle,dc=com")
;
oidStore = oidFactory.getIdentityStoreInstance(storeEnv);
```

Configuring the Provider at Runtime

To facilitate adding and modifying properties at runtime, the User and Role API provides the `getStoreConfiguration` method. For example, to set `RT_USER_SEARCH_BASES` at runtime:

```
StoreConfiguration conf = oidStore.getStoreConfiguration();
conf.setProperty("RT_USER_SEARCH_BASES", "dc=us,dc=oracle,dc=com");
```

Programming Guidelines

The following sections contain recommendations to consider when you work with providers and provider artifacts:

- Switching Providers
- Using Identity Store Objects

Switching Providers

To ensure that your application works after switching providers, consider the following guidelines:

1. Use the constants specified in `oracle.security.idm.UserProfile` only to refer to user properties. Avoid using native constants, which are not portable. For example, if you must obtain a user's login name, then fetch it using the `UserProfile.USER_NAME` constant:

   ```
   Property prop = usrprofile.getProperty(UserProfile.USER_NAME);
   ```

2. Use code like the following to obtain all the properties of a user,

   ```
   UserProfile upf = null;
   List proplst = store.getUserPropertyNames();
   String[] proparr = (String[]) proplst.toArray(new String[proplst.size()]);
   PropertySet pset = upf.getProperties(proparr);
   ```

3. Do not use native wildcard characters directly in your search filter string when you create search filters. Use instead the `SimpleSearchFilter.getWildCardChar` method that fetches the correct wildcard character for the provider:

   ```
   SmpleSearchFilter sf = m_identityStore.getSimpleSearchFilter(
      attrName, SimpleSearchFilter.TYPE_EQUAL, null);
   sf.setValue(filterStringWithoutWildcard+sf.getWildCardChar());
   ```

   For example, the method returns `%` for a database provider and `*` for an LDAP provider.

4. Apply the following conversion on the filter while generating the User and Role API filter, if your application accepts user-supplied filter strings with a predefined wildcard character:

   ```
   //User supplied filter 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);
   ```

Using Identity Store Objects

When implementing your application, consider the following points:

- The identity store implementations are not thread-safe. The User and Role API assumes that the store instances are not shared among threads. If the store instance is shared among threads, then the application code must take care to handle any required thread safety issues.

- Some applications allow a user session to change the create and search bases and various other runtime properties, which you define as runtime properties, and the identity store changes its runtime behavior according to these settings. The framework allows only one identity store instance per session.

The Provider's Lifetime

A provider persists in the environment until you close the factory instance. When the provider instance ends, all the objects that were created with that instance become invalid and subsequent calls on them throw an exception. When you close a factory instance, it is recommended that you explicitly close server connections and delete no longer valid instances. Factory instances are thread-safe, but identity store instances are not.

Similar considerations apply to closing the identity store instance.

Searching the Identity Store

Queries to the identity store can return a single identity or multiple ones. The following sections explain how to use the User and Role API to query the identity store:

- Searching for a Specific Identity
- Searching for Multiple Identities
- Using Search Filters

Searching for a Specific Identity

To query for a specific identity or role use any the following methods:

```
IdentityStore.searchUser(String name);
IdentityStore.searchUser(Principal principal);
IdentityStore.searchUser(int searchType, String name);
IdentityStore.searchRole(int searchType, String value);
```

Use these methods when you must get the object reference to a known identity kept in the store. They raise an exception if multiple entities with the same value are found in the store.

Searching for Multiple Identities

To query for multiple identities or roles use any of the following methods:

```
IdentityStore.search(SearchParams params);
IdentityStore.searchUsers(SearchParams params);
IdentityStore.searchRoles(int searchType, SearchParams params);
IdentityStore.searchProfiles(SearchParams params);
```

Using Search Filters

The User and Role API includes a number of search filters that facilitate a variety of search operations. The following sections explain the use of search filters:

- Filter Operators
- Filter for Logged-In User and Role
- Filters Examples

Filter Operators

The User and Role API supports the operators "=", "<",">", "<=" and ">=" which are used to create simple search filters, and the operators "&" and "|" which are used to combine two or more search strings to make a complex search filter.

Use the `NOT` operator in a simple or a complex search filter. The negated filter returns the entities which were rejected by the filter.

Filter for Logged-In User and Role

Applications commonly must retrieve the identity of the logged-in user and the user's group name. The `user.login.attr` and `groupname.attr` attributes are set when a user logs in, and you use the `UserProfile.getUserName` and `RoleProfile.getProperty(RoleProfile.NAME)` methods to get their values.

The following example shows how to query a logged in user:

```
SimpleSearchFilter sf = oidStore.getSimpleSearchFilter(
      UserProfile.USER_NAME,  SimpleSearchFilter.TYPE_EQUAL, "sampleUserName");
   SearchParameters ssp = new SearchParameters(sf, SearchParameters.SEARCH_USERS_ONLY);

   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 );
   }
```

The following example shows how to query the logged-in user's role:

```
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 );
    }
```

Filters Examples

The implementation of a simple search filter depends on the underlying store. In the following example, the filter allows all entries with value not null for the `NAME` field:

```
SimpleSearchFilter sf =
   oidStore.getSimpleSearchFilter(UserProfile.NAME,
   SimpleSearchFilter.TYPE_EQUAL, null);
sf.setValue(sf.getWildCardChar());
```

The following example retrieves users whose preferred language is not English:

```
SimpleSearchFilter sf =
   oidStore.getSimpleSearchFilter(
      UserProfile.PREFERRED_LANGUAGE,
      SimpleSearchFilter.TYPE_EQUAL,
      "english");
sf.negate();
```

The following example 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);
```

To use of nested filters to search for users whose name starts with a letter between "a" and "j" but not with the letter "i":

```
[continues from previous example]
SimpleSearchFilter sf3 =

oidStore.getSimpleSearchFilter(UserProfile.NAME,SimpleSearchFilter.TYPE_EQUAL,nul
l);
sf3.setValue("i"+sf3.getWildCardChar());
sf3.negate();

SearchFilter sfArray2[] = new SearchFilter[] {cf1, sf3};
ComplexSearchFilter cf2 =
   store.getComplexSearchFilter(sfArray2, ComplexSearchFilter.TYPE_AND);
```

The following example filters names starting with the letter A and prints the return values:

```
// search filter (cn=a*)
SimpleSearchFilter sf =

oidStore.getSimpleSearchFilter(UserProfile.NAME,SimpleSearchFilter.TYPE_EQUAL,nul
l);
```

```
sf.setValue("a"+sf.getWildCardChar());

SearchParameters params = new SearchParameters();
params.setFilter(sf);

// Search 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());
}
```

Creating and Modifying Entries in the Identity Store

The User and Role API allows you to add and modify identities with the use of the `UserManager` and `RoleManager` classes. To obtain `UserManager` and `RoleManager` instances from a store instance, use the following calls:

```
UserManager um = storeInst.getUserManager();
RoleManager rm = storeInst.getRoleManager();
```

The following sections provide examples that illustrate:

- Creating Identities and Roles
- Modifying an Identity
- Deleting an Identity

- Creating Identities and Roles
- Modifying an Identity
- Deleting an Identity

Creating Identities and Roles

To create a new identity, use one of the following methods in the `UserManager` class:

```
createUser(java.lang.String name, char[] password)
createUser(java.lang.String name, char[] password, PropertySet suppliedProps)
```

The second method sets user properties with the passed values. Required attribute values not supplied are given default values.

To create a new role, use one of the following methods in the `RoleManager` class:

```
createRole(String roleName);
createRole(String roleName, int roleScope);
```

Roles are created, by default, with the enterprise scope.

Modifying an Identity

To modify an identity, first obtain a reference to it. Then use the `setProperty` method in the `User`, `UserProfile`, `Role`, or `RoleProfile` classes to modify the identity.

To replace a display name:

```
UserProfile usrprofile = usr.getUserProfile();
ModProperty mprop = new ModProperty(UserProfile.DISPLAY_NAME,
   "modified display name",
   ModProperty.REPLACE);
usrprofile.setProperty(mprop);
```

Valid operators are: `ADD`, `REMOVE`, and `REPLACE`.

To modify a value in an attribute with multiple values, first remove the value you want to modify, and then add the modified value.

Deleting an Identity

To delete an identity, first obtain the user and role references, and then use the `dropUser` and `dropRole` methods:

```
User usr;
Role role;
usrmanager.dropUser(usr);
rolemanager.dropRole(role);
```

User and Role API Examples

The following examples illustrate the use of the User and Role API to query and manage users:

- Searching Users Example
- Managing Users Example
- Searching Users Example
- Managing Users Example

Searching Users Example

The following example illustrates how to set up a search filter to query users. To query groups, use the `RoleProfile` class. To query users, use the `UserProfile` class.:

```
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);

         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();
    }
  }
}
```

Managing Users Example

To create, modify, and delete an identity in a Microsoft Active Directory store:

```
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","password");

        // create the factory
        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);

        // create the store
        storeEnv.put(ADIdentityStoreFactory.ST_SUBSCRIBER_NAME,
                     "dc=upad,dc=us,dc=oracle,dc=com");
        adStore = adFactory.getIdentityStoreInstance(storeEnv);

        //get UserManager and create user
        UserManager usrmanager = adStore.getUserManager();
        String usrname = "amyd";
        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() );

        // modify user
        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("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 succeded");
      else
        System.out.println("password verification failed");

      SimpleSearchFilter sf = adStore.getSimpleSearchFilter(
                   UserProfile.NAME, SimpleSearchFilter.TYPE_EQUAL, usrname);

      SearchParameters params = new SearchParameters();
      params.setFilter(sf);

      // Search 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());
      }

      // delete user
      System.out.println("deleting user "+usrname);
      usrmanager.dropUser(usr);
      System.out.println("user dropped");
    }catch (Exception e)
    {
      e.printStackTrace();
    }
  }
}
```

Configuring SSL for LDAP Providers

The following sections describe the User and Role API support for SSL connections to providers:

- Setting Up SSL to Providers
- Customizing SSL to Providers
- Setting Up SSL to Providers
- Customizing SSL to Providers

Setting Up SSL to Providers

LDAP providers for the User and Role API rely on the Java Secure Sockets Extension (JSSE) to provide SSL with LDAP identity stores.

To set up the ready-to-use SSL:

1. Specify values for the keystore and key password and set system properties:

```
String keystore = "<keystore location>";
String keypasswd = "<keystore password>";
System.setProperty("javax.net.ssl.trustStore",keystore);
System.setProperty("javax.net.ssl.trustStorePassword", keypasswd);
```

2. Set the SSL URL of the LDAP server:

```
factEnv.put(ADIdentityStoreFactory.ST_LDAP_URL,
 "ldaps://ldaphost:sslport/");
```

3. Set the security protocol to SSL:

```
factEnv.put("java.naming.security.protocol","SSL");
```

> **See also:**
>
> Configuring SSL for the Identity Store

Customizing SSL to Providers

To customize SSL:

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:

```
factEnv.put(ADIdentityStoreFactory.ST_LDAP_URL,
 "ldaps://ldaphost:sslport/");
```

3. Set the security protocol to SSL:

```
factEnv.put("java.naming.security.protocol","SSL");
```

20

Developing with the Identity Governance Framework

This chapter explains how to access and maintain identities with the Identity Directory API provided with the Identity Governance Framework.
This chapter includes the following sections:

- About the Identity Governance Framework
- About the Identity Directory API Configuration
- Using the Identity Directory API
- Configuring SSL Using the Identity Directory API

About the Identity Governance Framework

The Identity Governance Framework allows applications to access identity data uniformly regardless of the particular underlying identity repository. This framework includes the Identity Directory API, a flexible, fully configurable collection of interfaces that allows access to artifacts in the identity store.

To use the Identity Directory API, add the `igf-manifest.jar` file to the application classpath.

> **See also:**
>
> *Java API Reference for Identity Governance Framework Identity Directory*

Identity Directory API Overview

The Identity Directory API allows Java EE and SE applications to access and manage identity data. This API is part of the Identity Governance Framework and offers all the framework's benefits for identity control.

The Identity Directory API allows you to:

- Operate on users and groups.
- Change passwords.

- Force password changes.
- Maintain attributes with multiple values, and static and dynamic groups.

About the Identity Directory API Configuration

The Identify Directory API provides an interface to access and modify users and group information from different identity stores. The configuration is specified in the `DOMAIN_HOME/config/fmwconfig/ids-config.xml` and `ovd/ids/adapters.os.xml` files, and the OPSS configuration file.

> **See also:**
>
> Configuring Security Providers with Fusion Middleware Control
>
> LDAP Identity Properties
>
> Identity Directory API Configuration in *Developing Applications with Identity Governance Framework*
>
> *Java API Reference for Identity Governance Framework Identity Directory*

Using the Identity Directory API

The following sections include examples that illustrate the use of the Identity Directory API to manage users and groups:

- Initializing and Obtaining the Identity Directory Handle
- Creating and Deleting a User
- Obtaining and Modifying a User
- Simple and Complex User Search
- Creating and Deleting a Group
- Obtaining a Group
- Group Search Filter
- Adding and Deleting a Member to a Group
- Initializing and Obtaining the Identity Directory Handle
- Creating and Deleting a User
- Obtaining and Modifying a User
- Simple and Complex User Search
- Creating and Deleting a Group
- Obtaining a Group
- Group Search Filter
- Adding and Deleting a Member to a Group

Initializing and Obtaining the Identity Directory Handle

The following example illustrates how to obtain the Identity Directory handle and a directory instance:

```
JpsContextFactory ctxFactory = JpsContextFactory.getContextFactory();
JpsContext ctx = ctxFactory.getContext();

//find the service instance
IdentityStoreService idstoreService =
ctx.getServiceInstance(IdentityStoreService.class)
 to
//get instance
oracle.igf.ids.IdentityDirectory ids = idstoreService.getIdentityStore();
```

The following example initializes the service with the configuration present in the IDS location. All user and group operations are performed with 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;
    public Ids1Test() throws IDSException {
        // Set Operational Config
        OperationalConfig opConfig = new OperationalConfig();

        // 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.getIdentityDirectory("userrole", opConfig);

        // Get UserManager and GroupManager handles
        uMgr = ids.getUserManager();
        gMgr = ids.getGroupManager();
    }
```

Creating and Deleting a User

The following examples illustrate how to create and delete a user.

```
public Principal createUser() {
        Principal principal = null;
        List<Attribute> attrs = new ArrayList<Attribute>();
        attrs.add(new Attribute("commonname", "test1_user1"));
        attrs.add(new Attribute("password", "password23".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"));
        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;
    }

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();
        }
    }
```

Obtaining and Modifying a User

The following examples illustrates how to obtain a handle to a user and modify it.

```
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;
   }

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();
       }
   }
```

Simple and Complex User Search

The following examples illustrate a simple and complex 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();
       }
   }

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")));

        // Request 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();
              System.out.println(user.getSubjectName());
              System.out.println("    " + user.getAttributeValue("commonname"));
           }
        } catch (Exception e) {
           System.out.println(e.getMessage());
           e.printStackTrace();
        }
     }
```

Creating and Deleting a Group

The following example illustrates how to create and delete 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;
   }
```

Obtaining a Group

The following example illustrates how to obtain a handle to a 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;
    }
```

Group Search Filter

The following example illustrates a search filter that returns multiple 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();
        }
    }
```

Adding and Deleting a Member to a Group

The following examples illustrate how to add and delete a user from 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();
        }
    }

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();
        }
    }
```

Configuring SSL Using the Identity Directory API

For information about Secure Sockets Layer (SSL) configuration when using the Identity Directory API, see Configuring SSL for the Identity Store.

21
Developing with the Keystore Service

This chapter explains how to use the Keystore Service API in your applications and includes examples of common keystore operations to access and retrieve keys at runtime.
This chapter includes the following topics:

- About the Keystore Service API
- Setting Policy Permissions
- Using the Keystore Service API in Java EE Applications
- Using the Keystore Service API in Java SE Applications
- Keystore Service API Examples

About the Keystore Service API

A keystore is used to store keys and certificates securely, and you use the Keystore Service API to access, retrieve, and maintain data in the keystore repository. This repository type can be a file, an LDAP, or a DB.

Operations include creating, reading, updating, and deleting keys, and get handles to keystore artifacts the keystore configured properties. These operations are secured by the`KeyStoreAccessPermission` class that implements the fine-grained access control model used by the service.

Before using the Keystore Service API in your application:

- Identify the application stripe and the keystore to use.
- Provision the required policies that enable your application to access and use the keystore. For information about provisioning policies, see Setting Policy Permissions.
- Make sure that the keystore instance is specified in the `jps-config.xml` file.

> **See also:**
>
> Managing Keystores with Fusion Middleware Control
>
> Managing Keystores with WLST
>
> Permission for a Keystore Example

Setting Policy Permissions

The keystore provider configured in the domain is instantiated when the server starts. A file keystore provider specifies its content in the `system-jazn-data.xml` file.

You can secure keys at the application stripe level, at the keystore level, or at a single key level. To access the keystore with the Keystore Service APIs, your application must have a codesource permission that specifies the operations the application is allowed.

Oracle recommends that these permissions be restricted to specific applications and keys.

The following sections illustrate several permission examples:

Permission for a Keystore Example

The following example illustrates a policy that grants a codesource any action to a keystore within a stripe:

```
<jazn-policy>
    <grant>
        <grantee>
            <principals>...</principals>
            <!-- This is the location of the JAR as loaded with the runtime -->
            <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 to which you want to grant permissions.
- `keystoreName` is the name of the keystore.

- `alias` indicates the key alias within the keystore. The wild card indicates the application is granted permission for all aliases.

> **See also:**
>
> Structure of the Keystore Service

Permission for a Map Example

The following example illustrates a policy that grants a codesource read, write, and delete permissions to all keystores within a stripe:

```
<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>
```

Permission for a Key Alias Example

The following example illustrates a policy that grants a codesource read permission to a specific keystore and alias within a stripe:

```
<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>
```

Using the Keystore Service API in Java EE Applications

To use the Keystore Service API in Java EE applications:

1. Set up the permission required for your application to use the API.
2. Start Oracle WebLogic Server.
3. Deploy the application.

> **See also:**
>
> Setting Policy Permissions
>
> Configuring Java EE Applications to Use OPSS

Using the Keystore Service API in Java SE Applications

To use the Keystore Service API in Java SE applications:

1. Ensure that the `jps-manifest.jar` file is in your classpath.
2. Set up the permission required for your applicant to use the API.
3. Set Java Virtual Machine (JVM), as appropriate:

   `-Doracle.security.jps.config`

   to specify the full path to the `jps-config-jse.xml` configuration file.

   `-Djava.security.policy`

   to specify the location of the `weblogic.policy` policy file.

   `-Dcommon.components.home`

   to specify the location of the `oracle_common` directory under middleware home.

   `-Dopss.version`

   to specify the version used in the environment.

   `-Djava.security.debug=all`

   to specify a debug level.
4. Run the application.

See also:

Setting Policy Permissions

About Java SE Application Security

Using OPSS in Java SE Applications

Keystore Service API Examples

The following sections show examples of use of the API:

- Keystore Service Management Example
- Reading Keys at Runtime Example

Keystore Service Management Example

The following example illustrates common keystore 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();
    }
     /*
     * because 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);
            // this call is in a doPrivileged block
            doPrivilegedKeyStoreOperation();


        } catch (JpsException je) {
            je.printStackTrace();
        }
    }
}
```

See also:

Java API Reference for Oracle Platform Security Services

Reading Keys at Runtime Example

The following examples illustrate several ways to access keystore artifacts at runtime.

- Getting a Handle to the Keystore
- Accessing Keystore Artifacts - Method 1
- Accessing Keystore Artifacts - Method 2
- Getting a Handle to the Keystore
- Accessing Keystore Artifacts - Method 1
- Accessing Keystore Artifacts - Method 2

Getting a Handle to the Keystore

To access a keystore, your application must first get the keystore handle by either specifying the application stripe and keystore name as separate parameters, or by specifying the application stripe and keystore name as one parameter.

Both methods work exactly in the same way for fetching keys and certificates, the only difference is the way the keystore is loaded. The first method 1 uses OPSS APIs. The second one uses Java SE Development Kit (JDK) APIs.

Accessing Keystore Artifacts - Method 1

The following example illustrates how to get a handle to the keystore, in which you specify the application stripe and keystore as 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.
// Method 1: Explicitly specify application stripe and keystore name as
// separate parameters. In this example, the last parameter is null because
// we are demonstrating a password-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.
Key key = keyStore.getKey("key-alias", null);

// Get the certificate associated with this alias
Certificate cert = keyStore.getCertificate("cert-alias");
```

> **Note:**
>
> Oracle recommends that you place calls to the methods `load`, `getKey`, `getCert` inside a `doPrivileged` block.

Accessing Keystore Artifacts - Method 2

The following example illustrates how to get a handle to the keystore, in which you specify the application stripe and keystore in one parameter using the JDK keystore URI.

```
// Create an object with parameters indicating the keystore to load
FarmKeyStoreLoadStoreParameter param = new FarmKeyStoreLoadStoreParameter();
param.setAppStripeName("keystoreapp");
param.setKeyStoreName("ks1");
// The password is set to null in this example because 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. because 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 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:

Oracle recommends you place calls to the methods `load`, `getKey`, `getCert` inside a `doPrivileged` block.

22

Developing with Oracle Fusion Middleware Audit Framework

This chapter explains how to use Oracle Fusion Middleware Audit Framework to audit application events. Using this framework, you create event definitions, register the application with the service at deployment, modify audit configuration at runtime, and generate reports. This chapter includes the following topics:

- Integrating Applications with the Oracle Fusion Middleware Audit Framework
- Creating Audit Definition Files
- Registering the Application with the Audit Service
- Managing Audit Policies Programmatically
- Logging Audit Events Programmatically
- Updating and Maintaining Audit Definitions

Integrating Applications with the Oracle Fusion Middleware Audit Framework

To integrate your application with the Audit Framework, perform the tasks explained in the following sections:

Creating Audit Definition Files

The following sections explain how to create the `component-event.xml` and translation files:

> **See also:**
>
> About the component_events.xml File

- The component-events.xml File
- Translation Files

The component-events.xml File

When you create the `component-events.xml` audit definition file, keep in mind the following points:

- Any change to an audit event definition requires that the version ID be modified by changing the minor or major numbers.
- There must be a 1-to-1 correspondence between tables for your application attribute definitions and database columns. Each custom attribute must have a corresponding order number in its definition.

> **See also:**
>
> What Are Version Numbers?
>
> About Custom Attribute to Database Column Mappings

Translation Files

The following procedure explains how to generate the XLIFF (XML Localization Interchange File Format) translations files and pack them in the `component_events_xlf.jar` file. At deployment and during registration, this information is stored in the audit store along with the component event definition.

1. Run a command like the following to generate XLIFF files:

```
java -cp $MW_HOME/oracle_common/modules/oracle.jps_12.2.1/jpsaudit.jar:
$MW_HOME/oracle_common/modules/oracle.jps_12.2.1/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. This `xlf` file contains translation units as well as help texts for all categories, events, and attributes. The prefixes for these are Category_, Event_ and Attribute_.
3. Package the translated files in a JAR file.

Registering the Application with the Audit Service

The following sections explain the various mechanisms by which applications can register with audit:

- Performing Declarative Audit Registration
- Programmatic Registration
- Registering the Application with Audit Using WLST
- Using Domain Extension Templates for Audit Artifacts

> **Note:**
>
> The use of domain extension templates is the preferred approach to registration. For information about templates, see Using Domain Extension Templates for Audit Artifacts.

- Performing Declarative Audit Registration
- Programmatic Registration
- Registering the Application with Audit Using WLST
- Using Domain Extension Templates for Audit Artifacts

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
- 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 store if the application is not yet registered.
- Redeployment - Upgrades the component event definition if the component is already registered.
- Undeployment - Removes the application's audit event definition from the audit store.

A simple database view to query audit records is created in the `IAU` schema at registration. This occurs for all except the component, for which Oracle Fusion Middleware Repository Creation Utility creates the view.

Custom Audit Registration

The parameters to customize audit registration are set in the `weblogic-application.xml` WebLogic Server deployment descriptor. This file is packaged in the `META-INF` directory of the application Enterprise ARchive (EAR) file. For information about packaging requirements, see Packaging a Java EE Application Manually.

A simple database view to query audit records is created in the `IAU` schema by audit registration, unless view creation is explicitly disabled.

Table 22-1 shows the parameters with their options:

Table 22-1 Parameters for Audit Registration

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 version support.
	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)
opss.audit.iauview	SIMPLE INDEXABLE DISABLE	• Creates a simple `IAU` view (default behavior) • Creates an indexable `IAU` view if indexes are specified in the `component_events.xml`application file. • Disables view creation.

The following example illustrates the use of registration and deregistration 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>
```

Programmatic Registration

The following example shoes how to register an application with audit programmatically:

```
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;
            }
        });
```

Calling the `auditService.register` method requires that the application be granted the `AuditStoreAccessPermission` permission with create, upgrade actions. For information about specifying grants, see Setting System Grants.

Here is a programmatic registration example:

```
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 API to register audit event definition
AccessController.doPrivileged(new PrivilegedExceptionAction<Object>() {
      public Object run() throws AuditException, IOException {
      auditService.register(auditReg);
      return null;
   }
});
```

Registration creates a database view to query audit records from the database. To add logic to manage view creation or to disable this feature, implement the interface `oracle.security.jps.service.audit.AuditRegistrationExt`, an interface that extends `AuditRegistration`:

```
SampleRegistration.java
class SampleRegistration implements AuditRegistrationExt {
  /*
    methods from Audit Registration go here
   */
  public AuditRegistrationExt.TYPE getIAUViewSupportType() {
     // add app specific logic here to determine the type
    return AuditRegistrationExt.TYPE.<type>;  // where type is SIMPLE, INDEXABLE,
DISABLE
  }
}
```

Registering the Application with Audit Using WLST

To register your application with audit, use the `registerAudit` WebLogic Scripting Tool (WLST) command. This command also allows you to create an audit view with the registration:

```
registerAudit(xmlFile, [xlfFile], componentType, [mode=OVERWRITE|UPGRADE],
[createView=SIMPLE|INDEXABLE|DISABLE])
```

where `SIMPLE` is the default if no value is provided for `createView`.

See registerAudit in the *WLST Command Reference for Infrastructure Security*.

Using Domain Extension Templates for Audit Artifacts

When a domain is created or extended, applications can specify the seeding of application-specific security data in the security store. For information about artifact seeding, see How Security Artifacts Are Seeded.

Java EE applications can register with audit by means of domain extension templates containing the `component_events.xml` and `component_events_xlf.jar` files. Audit registration processes them automatically when you deploy the application. For information about the files included in the template, see Layered Component Security Artifacts.

Managing Audit Policies Programmatically

The Audit Framework allows you to query audit data, and get and set audit policies programmatically. First use the following method to obtain an `AuditService` instance:

```
AuditService auditService =
JpsContextFactory.getContextFactory().getContext().getServiceInstance(AuditServic
e.class);
    AuditAdminService auditAdminService = auditService.getAdminService();
```

The following sections explain how to use this instance to query and view audit data:

- Querying Audit Data
- Viewing and Setting Audit Policies
- Querying Audit Data
- Viewing and Setting Audit Policies

Querying Audit Data

To query audit data, use any of the following methods:

```
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 query audit data:

```
 //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();

            }
        }
    }
```

Viewing and Setting Audit Policies

To retrieve and set an audit policy, use the following methods:

```
AuditPolicy getAuditPolicy(String componentType) throws AuditException;
void setAuditPolicy(String componentType, AuditPolicy auditPolicy) throws
AuditException;
```

The following example shows how to use these methods to obtain and 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;
        }
    });
```

Logging Audit Events Programmatically

The following sections illustrate how applications can use the Oracle Fusion Middleware Audit Framework to access audit programmatically and generate their own audit events:

- Oracle Fusion Middleware Audit Framework Interfaces
- Setting System Grants
- Obtaining the Auditor Instance

Oracle Fusion Middleware Audit Framework Interfaces

The Audit Framework provides the following class and interfaces:

```
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)
}
```

To set permissions and to obtain an auditor instance, see Setting System Grants, and Obtaining the Auditor Instance.

Setting System Grants

Applications must have the `AuditStoreAccessPermission` permission to use methods provided by the Audit Framework. In this example, the grant allows the `MyApp`

application to call the `auditService.getAuditor` method in an `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,action3</actions>
  </permission>
 </permissions>
</grant>
```

where the <actions> is the comma-separated list of actions that the application can carry out. The following table lists the available actions:

Action	Authorizes Client to
create	Call `AuditService.register` to register a new audit component.
upgrade	Call `AuditService.register` to upgrade a component event definition.
delete	Call `AuditService.unregister` to deregister an audit component.
read	Call `AuditService.getAuditor` to get an instance of component auditor.
modify	Call `AuditAdminService.setAuditPolicy` to modify audit policy.

Obtaining the Auditor Instance

After your application registers to audit, it can get its auditor instance programmatically:

```
//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);
}
```

Updating and Maintaining Audit Definitions

As the application's audit requirements evolve, you typically update audit definitions to reflect those changes. Specifically, you:

1. Update the audit definition file. For information about versions, see About Mapping and Version Rules.
2. Redeploy the application EAR file with the updated event definition file. Alternatively, notify audit registration about the newer version.
3. Verify your changes.

> **Note:**
>
> When you create or extend a WebLogic Server domain, you can specify audit artifacts in the domain template to facilitate audit definition upgrades. For information about files in component templates, see Layered Component Security Artifacts.

23

Configuring Java EE Applications to Use OPSS

This chapter describes the configuration and packaging recommended for Java EE applications that use OPSS but do not use Oracle Application Development Framework (Oracle ADF) security.
This chapter is includes the following sections:

- About Authentication in Java EE Applications
- Developing Authentication in Java EE Applications
- Configuring the Filter and the Interceptor
- Choosing the Appropriate Class for Enterprise Groups and Users
- Packaging a Java EE Application Manually
- Configuring Java EE Applications to Use OPSS

The following security-related files are relevant in a Java EE application during development, deployment, and runtime:

- `DOMAIN_HOME/config/fmwconfig/jps-config.xml`
- `DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml`
- `jazn-data.xml` (packaged in the application Enterprise ARchive (EAR) file)
- `cwallet.sso` (packaged in the application EAR file)
- `web.xml` (packaged in the application EAR file)
- `weblogic-application.xml` (packaged in the application EAR file)

> **See also:**
>
> Configuring Java SE Applications to Use OPSS
>
> Developing Oracle ADF Applications
>
> Securing Oracle ADF Applications
>
> OPSS API References

- About Authentication in Java EE Applications
- Developing Authentication in Java EE Applications
- Configuring the Filter and the Interceptor
- Choosing the Appropriate Class for Enterprise Groups and Users
- Packaging a Java EE Application Manually

- Configuring Java EE Applications to Use OPSS

About Authentication in Java EE Applications

The following topics contain information about developing authentication in Java EE applications:

- Authentication in *Understanding Security for Oracle WebLogic Server*.
- *Developing Applications with the WebLogic Security Service*:
  - Securing Web Applications
  - Using JAAS Authentication in Java Clients
  - Using SSL Authentication in Java Clients
- *Developing Security Providers for Oracle WebLogic Server*:
  - Authentication Providers
  - Login Modules
  - How to Develop a Custom Authentication Provider
  - Identity Assertion Providers
  - Servlet Authentication Filters

Developing Authentication in Java EE Applications

The OPSS login modules support programmatic user authentication and assertion in Java EE and SE applications, and the API you use to call these login modules is common to both Java EE and SE applications. Oracle WebLogic Server delegates user authentication and assertion to WebLogic Server security providers.

See also:

Authentication Providers in *Developing Security Providers for Oracle WebLogic Server*

Using Login Modules in Java Applications

Configuring the Filter and the Interceptor

The configurations explained in this section are required only if you are packaging or configuring your Java EE application (integrated with OPSS) outside Oracle JDeveloper.

OPSS provides the `JpsFilter` filter for Java servlets and the JpsInterceptor interceptor for Enterprise JavaBeans (EJB). The filter is configured in the `web.xml` file packed in a web application archive (WAR) file. The interceptor is configured in the `ejb-jar.xml` file packed in a JAR file.

OPSS also provides a way to configure the stripe that application MBeans access, in the `web.xml` file.

Ready-to-use, the filter is set with the default parameter values so you need not configure it in the deployment descriptor. However, you must configure it manually if a value different from the default value is required. The interceptor must be manually configured.

The parameters you use to configure the filter and the interceptor are explained in the following sections:

- Setting the Application Stripe
- Setting Application Role Support
- Setting the Anonymous User and Role
- Setting Authenticated Role Support
- Setting JAAS Mode

> **See also:**
>
> About the Anonymous User and Role
>
> About the Authenticated Role
>
> Interceptor Configuration Requirements
>
> Summary of Filter and Interceptor Parameters

- Setting the Application Stripe
- Setting Application Role Support
- Setting the Anonymous User and Role
- Setting Authenticated Role Support
- Setting JAAS Mode
- Interceptor Configuration Requirements
- Summary of Filter and Interceptor Parameters

Setting the Application Stripe

The application stripe is optionally specified in the application `web.xml` file, and it is used at runtime to determine which set of policies are applicable. If unspecified, it defaults to the application ID (which includes the application name).

To set the application stripe, use the `application.name` parameter. This setting is optional and case-sensitive. If unspecified, it defaults to the name of the application.

To specify the application stripe, configure either the `filter` or the `context-param` elements in `web.xml`:

```
<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>

<context-param>
 <description>JPS custom stripe id</description>
 <param-name>application.name</param-name>
 <param-value>stripeid</param-value>
</context-param>
```

If your application contains application MBeans that access the security store and your the name of your application is different from the name of the stripe they access, then you must use the `context-param` configuration like the previous one to specify the stripe name.

Additional Examples

The following example illustrates the configuration of the `MyServlet1` and `MyServlet2` servlets in `web.xml`. Note that servlets in the same WAR file 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 example illustrates the configuration of the `MyAppName` application stripe and the `JpsInterceptor` interceptor in the `ejb-jar.xml` file, and the binding of that interceptor to `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>.
```

Setting Application Role Support

To add application roles to a subject, set the `add.application.roles` parameter to `true`. Not to add them, set it to `false`. The default value is `true`. The principal class for an application role is the `oracle.security.jps.service.policystore.ApplicationRole` class.

Setting the Anonymous User and Role

To enable the use of the anonymous user, set `enable.anonymous` to `true`. To disable it, set it to `false`. The default value is `true`.

To remove the anonymous role from a subject, set `remove.anonymous.role` to `true`. To retain it, set it to `false`. The default value is `false`.

The names and the principal classes for the anonymous user and role are:

```
anonymous
oracle.security.jps.internal.core.principals.JpsAnonymousUserImpl
anonymous-role
oracle.security.jps.internal.core.principals.JpsAnonymousRoleImpl
```

The following example illustrates the setting of the anonymous user and role in the `web.xml` file, and the use of the `JpsFilter` filter by the `MyServlet` Java servlet:

```
<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 example illustrates the setting of `remove.anonymous.role` in the `ejb-jar.xml` file, and the use of the interceptor by `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>
```

Setting Authenticated Role Support

To add the authenticated role to a subject, set the `add.authenticated.role` property to `true`. Not to add it, set it to `false`. The default value is `true`.

The name and principal class for the authenticated role are:

```
authenticated-role
oracle.security.jps.internal.core.principals.JpsAuthenticatedRoleImpl
```

The following example illustrates the configuration of the `add.authenticated.role` property and the use of the `JpsFilter` filter by the `MyServlet` Java servlet, in the `web.xml` file:

```
<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>
```

Setting JAAS Mode

To set JAAS mode use the `oracle.security.jps.jaas.mode` parameter with one of the following values:

```
doAs
doAsPrivileged
off
undefined
subjectOnly
```

The default value is `doAsPrivileged`.

The following example illustrates the use of the `JpsFilter` filter by the `MyServlet` Java servlet:

```
<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 example illustrates a `doAs` setting and the use of the `JpsInterceptor` interceptor:

```
<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>
```

Interceptor Configuration Requirements

The following requirements apply to all `JpsInterceptor` parameter specifications:

- Must specify the parameter type in the `env-entry-type` element.
- Must specify the `injection-target` element, which identifies the same class as that of the interceptor (specified in the `injection-target-class` element). In the `injection-target-name` element, the name must be rewritten as a string where the dots are replaced by underscores.
- Must specify the binding of the interceptor to EJB by referring to the interceptor's class.

Summary of Filter and Interceptor Parameters

The following table lists the filter and the interceptor parameters:

Table 23-1 JpsFilter and JpsInterceptor Parameters

Parameter Name	Values	Default	Function	Notes
application.name	Any valid string. The value is case-sensitive.	The name of the application.	Specifies the subset of policies that the Java servlet or EJB use.	Must specify if several Java servlets or EJB access the same subset of policies.

Table 23-1 (Cont.) JpsFilter and JpsInterceptor Parameters

Parameter Name	Values	Default	Function	Notes
add.application.roles	`true` or `false`	`true`	Adds application roles to the subject.	Set to `false` not to add application roles to a subject.
enable.anonymous	`true` or `false`	`true`	Enables the anonymous user in the subject.	If `true`, then it includes the anonymous user and role in a subject.
remove.anonymous.role	`true` or `false`	`false`	Removes the anonymous role from the subject after authentication.	Available for Java servlets only. For EJB, the anonymous role is removed from a subject. If `false`, then the subject retains the anonymous role after authentication. If `true`, then it is removed after authentication.
add.authenticated.role	`true` or `false`	`true`	Allows adding the authenticated role to the subject.	Set to `false` not to include the authenticated role to a subject.
oracle.security.jps.jaas.mode	doAsPrivileged doAs off undefined subjectOnly	doAsPrivileged	Sets the JAAS mode.	Not Applicable (N/A)

Choosing the Appropriate Class for Enterprise Groups and Users

The classes you specify in members of an application role must be either the application role class or one of the following two:

```
weblogic.security.principal.WLSUserImpl
weblogic.security.principal.WLSGroupImpl
```

The following example illustrates the use of the `ApplicationRole` and `WLSGroupImpl` classes in the specification of a role:

```
<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>
```

Application role names are case-insensitive. Enterprise user and group names are case-sensitive.

> **See also:**
>
> Failure to Get Permissions - Case Mismatch

Packaging a Java EE Application Manually

This section explains the packaging requirements for Java servlets and EJB that use custom policies and credentials.

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

Java servlets are packaged in a WAR file that contains the `web.xml` configuration file. The EJB is packaged in a WAR file that contains the `ejb-jar.xml` configuration file. The WAR file must include the configuration of the filter (for Java servlets) or the interceptor (for EJB) in the corresponding configuration file.

The following packaging requirements are stated to Java servlets and the configuration of the filter in the `web.xml` file, but they also apply to EJB and the configuration of the interceptor in the `ejb-jar.xml` file:

- The application must be packaged in a single EAR file.
- The EAR file must contain exactly one `META-INF/jazn-data.xml` file that specifies application policies and roles.
- 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` file that configures the `JpsFilter` filter and such configurations in all EAR files must be identical.
- Component credentials in `cwallet.sso` files can be packaged in the EAR file.

If a component requires a filter configuration different from that of other components, then it must be packaged in a separate EAR file and deployed separately.

> **See also:**
>
> Configuring the Filter and the Interceptor

- Packaging Policies with the Application

- Packaging Credentials with the Application

Packaging Policies with the Application

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

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. No other policy package combination is supported, and policy files other than `jazn-data.xml` are disregarded.

Packaging Credentials with the Application

Application credentials are defined in the `cwallet.sso` file. The only supported way to include this file with an application is to package it in the `META-INF` directory of the application EAR file.

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. No other credential package combination is supported, and credential files other than the `cwallet.sso` file are disregarded.

Configuring Java EE Applications to Use OPSS

The following sections describe how to control the migration of policies and credentials packaged with a Java EE application at application deployment:

- Controlling Policy Migration
- Configuring Policy Migration According to Behavior
- Controlling Credential Migration
- Configuring Credential Migration According to Behavior
- Using File Credential Stores
- Using Supported Permission Classes
- Specifying Bootstrap Credentials Manually

> **See also:**
>
> Migrating Identities with migrateSecurityStore

- Controlling Policy Migration
- Configuring Policy Migration According to Behavior
- Using File Credential Stores
- Controlling Credential Migration
- Configuring Credential Migration According to Behavior

- Using Supported Permission Classes
- Specifying Bootstrap Credentials Manually

Controlling Policy Migration

You control the migration of application policies at deployment with parameters specified in the `META-INF/weblogic-application.xml` file. If you do not use Fusion Middleware Control to manage your application, then you must enter these configurations manually.

Set the`jps.deployment.handler.disabled` system property to `true` to disable the migration of policies and credentials at deployment for *all* applications regardless of particular settings in the `weblogic-application.xml` files.

When you deploy an application that incudes policies and credentials to a Managed Server running in a computer different from where the Administration Server is running, do not use the life cycle listener to control migration at deployment, because the data maintained by the Managed Server and the Administration Server would end up not synchronized. Instead, use the `migrateSecurityStore` command to migrate those policies and credentials to the security store.

The parameters that configure the migration and removal of policies 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

> **See also:**
>
> Deploying Secure Applications with Fusion Middleware Control

- jps.policystore.migration
- jps.policystore.applicationid
- jps.apppolicy.idstoreartifact.migration
- jps.policystore.removal
- jps.policystore.migration.validate.principal
- JpsApplicationLifecycleListener

jps.policystore.migration

This parameter, configured in the following example, specifies whether to migrate and merge or overwrite policies in the security store:

```
<wls:application-param>
  <wls:param-name>jps.policystore.migration</wls:param-name>
  <wls:param-value>Option</wls:param-value>
</wls:application-param>
```

where `Option` stands for MERGE, OVERWRITE, or OFF.

Set to OFF to prevent migration. Set to MERGE to migrate merging policies. Set to OVERWRITE to migrate overwriting policies in the target store. The default value is MERGE.

jps.policystore.applicationid

This parameter, configured in the following example, specifies the target stripe into which policies are migrated.

```
<wls:application-param>
  <wls:param-name>jps.policystore.applicationid</wls:param-name>
  <wls:param-value>myApplicationStripe</wls:param-value>
</wls:application-param>
```

The value can be any valid string. If unspecified, then WebLogic Server creates a stripe name based on the application name and version using the `application_name#version` format.

The value of this parameter must match the value of the `application.name` parameter specified for the `JpsFilter` filter (in `web.xml`) or for the `JpsInterceptor` interceptor (in `ejb-jar.xml`).

The value read from the `weblogic-application.xml` file is used at deployment. The value read from the `web.xml` or the `ejb-jar.xml` file is used at runtime.

jps.apppolicy.idstoreartifact.migration

This parameter, configured in the following example, specifies whether to exclude migrating references to enterprise users or groups, such as the mapping of application roles to enterprise users or groups. Thus it allows you to the migration of *just* application policies and it ignores the mapping of application roles to enterprise users or groups.

```
<wls:application-param>
  <wls:param-name>jps.apppolicy.idstoreartifact.migration</wls:param-name>
  <wls:param-value>Option</wls:param-value>
</wls:application-param>
```

where `Option` stands for `true` or `false`. Set to `false` to exclude the migration of artifacts referencing enterprise users or groups. Otherwise, set it to `true`. If unspecified, then it defaults to `true`.

> **Note:**
>
> After you deploy the application with this parameter set to `false` and before the application is used in the domain, map application roles to enterprise groups or users with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) or Oracle WebLogic Server Administration Console.

The following examples illustrate a `jazn-data.xml` file, packaged in the application EAR file, that describes the policy. The `system-jazn-data.xml` file represents the file security store into which application policies are migrated. It is assumed that the `jps.apppolicy.idstoreartifact.migration` parameter is `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, configured in the following example, specifies whether to remove policies when you undeploy the application:

```
<wls:application-param>
  <wls:param-name>jps.policystore.removal</wls:param-name>
  <wls:param-value>OFF</wls:param-value>
</wls:application-param>
```

Set to OFF to prevent removing policies. If not set, then policies are removed. By default, policies are not removed when you undeploy the application.

Consider using this setting when multiple applications share the same application stripe and one of them is undeployed, but note that setting `jps.plicystore.ramoval` to OFF for a given application requires knowing, at application deployment, whether it uses an application stripe shared with other applications.

jps.policystore.migration.validate.principal

This parameter, configured in the following example, specifies whether to check for principals in system and application policies at deployment or redeployment:

```
<wls:application-param>
  <wls:param-name>jps.policystore.migration.validate.principal</wls:param-name>
  <wls:param-value>option</wls:param-value>
</wls:application-param>
```

where `option` stands for `true` or `false`.

Set to `true` to validate 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, then log a warning and continue. Set to `false`, to skip the check. If unspecified, it defaults to `false`. Validation warnings are logged in the server log.

JpsApplicationLifecycleListener

This parameter, configured in the following example, specifies the listener class:

```
<wls:listener>
  <wls:listener-class>
    oracle.security.jps.wls.listeners.JpsApplicationLifecycleListener
  </wls:listener-class>
</wls:listener>
```

Configuring Policy Migration According to Behavior

The following sections describe recommendations and the settings required in typical scenarios:

- Recommendations
- Skipping Migrating Policies
- Migrating Merging Policies
- Migrating Overwriting Policies
- Removing or Not Removing Policies
- Migrating Policies in a Static Deployment

Recommendations

The following recommendations apply to both policies and credential migration.

A value setting not described in the following sections is not recommended.

If you want to deploy your application to multiple Managed Servers, then choose to migrate when you deploy it to just one of those servers only. The rest of the deployments should choose *not* to migrate policies and credentials. This ensures that the policies and credentials are migrated only once from the application store to the store. All the deployments must use the same application ID.

An alternative to migrating security data at deployment is to migrate policies and credentials with the `migrateSecurityStore` WLST command. For information about this command, see Migrating the Security Store with migrateSecurityStore.

If your application packages policies and credentials and you deploy it to several servers, then you must include the Administration Server so that the process updates the`$DOMAIN_HOME/config/fmwconfig/system-jazn-data.xml` policy file.

Skipping Migrating Policies

The following table shows the settings to skip migration:

Table 23-2 Settings to Skip Policy Migration

Settings	Valid at Deployment or Redeployment
JpsApplicationLifecycleListener	Set
jps.policystore.migration	OFF

Skip migrating policies when you want to keep domain policies as they are, but you would migrate policies when deploying the application for the first time.

Migrating Merging Policies

The following table shows the setting to migrate only policies that are not in the target store:

Table 23-3 Settings to Migrate Merging Policies

Settings	Valid at Deployment or Redeployment
JpsApplicationLifecycleListener	Set
jps.policystore.migration	MERGE
jps.policystore.applicationid	Optional. Defaults to the Java servlet or EJB name.
jps.apppolicy.idstoreartifact.migration	Optional. Set to `false` to exclude migrating policies that reference enterprise artifacts. Otherwise set to `true`. Defaults to `true`.
jps.policystore.migration.validate.principal	Optional. Set to `true` to validate enterprise users and roles in application and system policies. Set to `false`, otherwise. If unspecified, it defaults to `false`.

Choose to migrate policies with merging at redeployment if a policy has changed (since last undeployment) or you want to add new policies.

Migrating Overwriting Policies

The following table shows the setting to migrate all policies overwriting matching target policies:

Table 23-4 Settings to Migrate Overwriting Policies

Settings	Valid at Deployment or Redeployment
JpsApplicationLifecycleListener	Set
jps.policystore.migration	OVERWRITE
jps.policystore.migration.validate.principal	Optional. Set to `true` to validate enterprise users and roles in application and system policies. Set to `false`, otherwise. If unspecified, it defaults to `false`.

Choose to migrate policies with overwriting at redeployment if you want to replace all policies with new ones.

Removing or Not Removing Policies

Only application policies can be removed at undeployment. System policies are not removed. The following table shows the setting to remove application policies at undeployment:

Table 23-5 Settings to Remove Policies

Settings	Valid at Undeployment
JpsApplicationLifecycleListener	Set
jps.policystore.removal	Not set (default)

The stripe from which policies are removed at undeployment depends on the stripe you specified at deployment or redeployment. If you redeploy an application with a stripe specification different than the original one, then policies in that original stripe are not removed.

The following table shows the settings to disable removing policies at application undeployment:

Table 23-6 Settings to Disable Removing Policies

Settings	Valid at Undeployment
JpsApplicationLifecycleListener	Set
jps.policystore.removal	OFF

What Gets Removed and What Remains

Consider the `myApp` application that has been configured for automatic migration and removal of policies. The following example (in the application's `jazn-data.xml` file) illustrates the policies that are migrated at application deployment and policies that are not removed at application undeployment:

```
<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 security 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 codesource grant is migrated to the element
     jazn-policy in domain system-jazn-data.xml; when myApp is undeployed
     with FMC, it is not removed from security 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>
```

Migrating Policies in a Static Deployment

The following table shows the settings to migrate 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 security store.

Table 23-7 Settings to Migrate Policies with Static Deployments

Settings	Valid at Static Deployments
JpsApplicationLifecycleListener	Set
jps.policystore.migration	MERGE or OVERWRITE

The following table shows the settings to skip migrating application policies when the application is statically deployed.

Table 23-8 Settings Not to Migrate Policies with Static Deployments

Settings	Valid at Static Deployments
JpsApplicationLifecycleListener	Set

Table 23-8 (Cont.) Settings Not to Migrate Policies with Static Deployments

Settings	Valid at Static Deployments
jps.policystore.migration	OFF

Using File Credential Stores

A file credential store is the `cwallet.sso` file. The location of this file is specified in the `jps-config.xml` file with the `<serviceInstance>` element:

```
<serviceInstance name="credstore" provider="credstoressp">
   <property name="location"  value="myCredStorePath"/>
</serviceInstance>
```

> **See also:**
>
> *Administering Oracle Fusion Middleware*
>
> - Common Wallet Operations
> - Managing Keystores, Wallets, and Certificates

Controlling Credential Migration

You control the migration of application credentials at deployment with parameters specified in the `META-INF/weblogic-application.xml` file. If you do not use Fusion Middleware Control to manage your application, then you must enter these configurations manually.

Set the `jps.deployment.handler.disabled` system property to `true` to disable the migration of policies and credentials at deployment for *all* applications regardless of settings in `weblogic-application.xml` application files.

When you deploy an application that packages policies and credentials to a Managed Server running in a computer different from where the Administration Server is running, do not use the life cycle listener to control migration at deployment because the data maintained by the Managed Server and the Administration Server would end up not synchronized. Instead, use the `migrateSecurityStore` command to migrate those policies and credentials to the security store. For information about this command, see Migrating the Security Store with migrateSecurityStore.

The parameters that configure the migration and removal of credentials are:

- jps.credstore.migration
- JpsApplicationLifecycleListener
- jps.credstore.migration

jps.credstore.migration

This parameter, configured in the following example, specifies whether to migrate and merge or overwrite credentials in the security store:

```
<wls:application-param>
  <wls:param-name>jps.credstore.migration</wls:param-name>
  <wls:param-value>Option</wls:param-value>
</wls:application-param>
```

where `Option` stands for MERGE, OVERWRITE, or OFF.

Set to OFF to prevent migration. Set to MERGE to migrate merging credentials. Set to OVERWRITE to migrate overwriting credentials. The default value is MERGE.

Configuring Credential Migration According to Behavior

The following sections describe recommendations and the settings required in typical scenarios:

- Recommendations
- Skipping Migrating Credentials
- Migrating Merging Credentials
- Migrating Overwriting Credentials

Note that application credentials are not removed when you undeploy the application.

- Skipping Migrating Credentials
- Migrating Merging Credentials
- Migrating Overwriting Credentials

Skipping Migrating Credentials

The following table shows the setting to prevent the migration:

Table 23-9 Settings to Skip Credential Migration

Settings	Valid at deployment or redeployment
jps.credstore.migration	OFF

Migrating Merging Credentials

The following table shows the settings required to migrate only credentials that are not in the target store:

Table 23-10 Settings to Migrate Merging Credentials

Settings	Valid at deployment or redeployment
JpsApplicationLifecycleListener	Set
jps.credstore.migration	MERGE

Migrating Overwriting Credentials

The following table shows the setting to migrate all credentials overwriting target credentials:

Table 23-11 Settings to Migrate Overwriting Credentials

Settings	Valid at deployment or redeployment
JpsApplicationLifecycleListener	Set
jps.credstore.migration	OVERWRITE
jps.app.credential.overwrite.allowed	Must be `true`

Using Supported Permission Classes

The following sections describe the values you use in the `<class>`, `<name>`, and `<actions>` elements within a `<permission>` element in the `system-jazn-data.xml` file:

- Security Store Permission Class
- Credential Store Permission Class
- Generic Permission Class

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.

- Security Store Permission Class
- Credential Store Permission Class
- Generic Permission Class

Security Store Permission Class

The security store permission class is:

```
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
```

If the permission applies to a particular application, then use the following pattern for the corresponding `name` element, where `appStripeName` specifies the particular application:

```
context=APPLICATION,name=appStripeName
```

When the permission applies to all applications, use the following name pattern:

```
context=APPLICATION,name=*
```

If the permission applies to all applications and system policies, then use the following name pattern:

```
context=SYTEM
```

The values allowed in the `actions` element are the following (* stands for any action):

```
*
createPolicy
getConfiguredApplications
getSystemPolicy
getApplicationPolicy
createApplicationPolicy
deleteApplicationPolicy
grant
revoke
```

```
createAppRole
alterAppRole
removeAppRole
addPrincipalToAppRole
removePrincipalFromAppRole
hasPermission
containsAppRole
```

The following examples illustrate grants to create policies and to do runtime checks:

```
grant {
   permission
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
 "context=APPLICATION,name=*", "*";
   permission
oracle.security.jps.service.policystore.PolicyStoreAccessPermission
 "context=SYSTEM,adminresource=ApplicationPolicy,instancename=*", "*";
oracle.security.jps.JpsPermission "AppSecurityContext.setApplicationID.*";
 };

grant codebase "file:${opss.lib.location}/-" { permission
java.security.AllPermission;
 };
```

Credential Store Permission Class

The credential store permission class is the `oracle.security.jps.service.credstore.CredentialAccessPermission` class.

If the permission applies to a map and a particular key in that map, then use the following pattern for the corresponding `name` element, where `myMap` and `myKey` specify the particular credential:

```
context=SYSTEM,mapName=myMap,keyName=myKey
```

If the permission applies to a map and all keys in it, then use the following pattern, where `myMap` specifies the particular map:

```
context=SYSTEM,mapName=myMap,keyName=*
```

The values allowed in the `actions` element are the following (* stands for any action):

```
*
read
write
update
delete
```

Generic Permission Class

The generic permission class is the `oracle.security.jps.JpsPermission` class.

If the permission applies to an assertion performed by the `oracle.security.jps.callback.IdentityCallback` callback instance, then use the following pattern for the corresponding `<name>` element:

```
IdentityAssertion
```

The only value allowed in the `actions` element is `execute`.

Specifying Bootstrap Credentials Manually

You specify the credentials needed to connect and access the domain repository in the `cwallet.sso` file, which is configured in the `jps-config.xml` file. These credentials are called bootstrap credentials and are always kept in a file.

Bootstrap credentials are configured in a context named `bootstrap_credstore_context`:

```
<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>
```

where the `cwallet.sso` file is assumed present in the `bootstrap` directory.

The `bootstrap.security.principal.key` and `bootstrap.security.principal.map` properties specify the bootstrap map and key:

```
<serviceInstance provider="ldap.policystore.provider" name="policystore.ldap">
  ...
  <property value="bootstrapKey" name="bootstrap.security.principal.key"/>
  ...
</serviceInstance>
```

If the `bootstrap.security.principal.map` property is unspecified, then it defaults to `BOOTSTRAP_JPS`.

To modify or add bootstrap credentials, see modifyBootStrapCredential and addBootStrapCredential in *WLST Command Reference for Infrastructure Security*.

24

Configuring Java SE Applications to Use OPSS

This chapter describes how to use OPSS to implement authentication, authorization, and audit in Java SE applications.
This chapter includes in the following topics:

> **See also:**
>
> Configuring Java EE Applications to Use OPSS

- Using OPSS in Java SE Applications
- Implementing Security Services in Java SE Applications
- Authentication in Java SE Applications
- Authorization in Java SE Applications
- Audit in Java SE Applications

Using OPSS in Java SE Applications

To use OPSS in a Java SE application:

- Place the `jps-manifest.jar` file in the class path.
- Call the `JpsStartup.start` method at initialization.
- Set the following properties:
  - `oracle.security.jps.config`, the file path to the `jps-config-jse.xml` file.
  - `common.components.home`, the path to the `oracle_common` directory.
  - `java.security.policy`, the path to the file `java.policy`.
  - `opss.audit.logDirectory`, the path to a writable directory where audit writes records for the application.
- Set `-Djava.security.policy` to the location of the Oracle WebLogic Server `weblogic.policy` policy file (located in the `wl_home/server/lib` directory).

- Use the OPSS `AppSecurityContext` class and note the required associated grant:

```
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 that you include in your `jps-config-jse.xml` 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>
```

See also:

The JpsStartup Class

Configuring the Security Store

OPSS System Properties

- The JpsStartup Class

The JpsStartup Class

The following sections describe the states, runtime options, and examples of use of the `JpsStartup` class:

- JpsStartup.start States
- JpsStartup Constructor
- JpsStartup runtime Options

- OPSS Starting Examples

> **See also:**
>
> *Java API Reference for Oracle Platform Security Services*

- JpsStartup.start States
- JpsStartup Constructor
- JpsStartup runtime Options
- OPSS Starting Examples

JpsStartup.start States

The transition states of the `JpsStartup.start` method are defined by the following constants:

- `UNINITIALIZED` - The state before a call to `JpsStartup.start`.
- `INITIALIZING` - The state to which it transitions after calling `JpsStartup.start`.
- `FAILURE` - The state to which it transitions if the `INITIALIZING` state run into any errors.
- `ACTIVE` - The state to which it transitions if the `INITIALIZING` state completed successfully.
- `INACTIVE` - The state to which it transitions after calling the `JpsStatup.stop` method.

The `JpsStartup.getState` method returns the current OPSS state.

JpsStartup Constructor

The class includes the following constructor:

```
JpsStatup(java.lang.String platformType,
          java.util.Map<java.lang.String,ContextConfiguration>  ctxtCfgs,
          java.util.Map<java.lang.String,?> options)
```

Call this constructor when you want to start OPSS with additional context details and options.

The `ctxCfgs` argument holds the details of the context specified in the `jps-config-jse.xml` file. To start contexts other than the default context, pass multiple context information. To start a context other than the default context, pass the context name. If unspecified, then it uses the default context.

The `options` argument groups specific OPSS startup values. For information about runtime options, see Table 24-1.

> **See also:**
>
> OPSS Starting Examples

JpsStartup runtime Options

The following table describes the options you use with the `jpsStartup` method:

Table 24-1 JpsStartup Runtime Options

Option	Values	Default	Description
`ACTIVE_CONTEXT`	The name of an active context.	default	Use the specified context. Otherwise, use the default context.
`ENABLE_JAVA_POLICY_PROVIDER`	`true` or `false`	`true`	Set to `true` to use the policy provider. Otherwise, set to `false`.
`ENABLE_AUDIT_SERVICE_EXT`	`true` or `false`	`true`	Set to `true` to start audit monitoring and the audit loader when the service is started. Otherwise, set to `false`.
`ENABLE_POLICY_STORE_SERVICE_EXT`	`true` or `false`	`true`	Set to `true` to start the Policy Distribution Point (PDP) with a policy change. 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 following options set to `false`: • `ENABLE_JAVA_POLICY_PROVIDER` • `ENABLE_AUDIT_SERVICE_EXT` • `ENABLE_POLICY_STORE_SERVICE_EXT` • `ENABLE_DEPLOYMENT_HANDLING` Otherwise set to `DEFAULT`.
`JPS_CONFIG`	The path to `jps.config-jse.xml` specified by `oracle.security.jps.config`		Use the specified path. Otherwise, use the default path.

OPSS Starting Examples

This section illustrates the use of the `JpsStartup` class in typical scenarios.

Example 1

The following example illustrates how to start OPSS with no explicit parameters.

```
JpsStartup jpsStartUp = new JpsStartup();
jpsStartUp.start();
jpsStartUp.getState();
jpsStartUp.stop();
```

Example 2

The following example illustrates how to start OPSS with specific configuration values. Note that:

- `default1` is the default context passed as part of the `contextConfig` map.
- To disable audit, set `ENABLE_AUDIT_SERVICE_EXT` to `false`. By default, it is enabled.
- To disable runtime services set `ENABLE_POLICY_STORE_SERVICE_EXT` to `false`. By default, it 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 an additional context:

```
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 assumes that the `jps.config-jse.xml` file includes the 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 `ACTIVE_CONTEXT` parameter specifies the context to use as default. If `DEFAULT_CONTEXT_KEY` is not passed as part of map, then the default context is started *provided* there is no `ACTIVE_CONTEXT` key passed as part of `startupOption` map.

Example 5

The following example illustrates how to start OPSS in the `SCRIPT` mode. This mode is used when you do not want runtime services 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();
```

Implementing Security Services in Java SE Applications

The following sections describe the OPSS support in Java SE applications:

- Authentication in Java SE Applications
- Authorization in Java SE Applications
- Using the Credential Store Framework API in Java SE Applications

Authentication in Java SE Applications

The following sections explain the identity store support in Java SE applications:

- Configuring the LDAP Identity Store in Java SE Applications
- Using Login Modules in Java Applications
- Using the Login Modules in Java SE Applications

Configuring the LDAP Identity Store in Java SE Applications

A Java SE application can use an LDAP identity store configured in the `jps-config-jse.xml` file with the `serviceProvider`, `serviceInstance`, and `jpsContext` elements:

```
<serviceProviders>
  <serviceProvider type="IDENTITY_STORE" name="idstore.ldap.provider"
class="oracle.security.jps.internal.idstore.ldap.LdapIdentityStoreProvider">
    <description>Prototype LDAP 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) must match the instance referenced in `serviceInstanceRef`.
- The name of the `serviceProvider` (`idstore.ldap.provider` in the example) must match the provider specified in `serviceInstance`.

> **See also:**
>
> Configuring Services with Scripts
>
> LDAP Identity Properties

Using Login Modules in Java Applications

Unless otherwise stated the information in this section applies to both Java EE and SE applications, and the API you use to call login modules is common to both kinds of applications.

A login module is a component that authenticates users and populates a subject with principals. The login module authenticates a user after requesting a name and a password or some other data. If the authentication succeeds, then the module assigns the relevant principals to the subject.

In any of the supported login modules, set the following properties:

```
enable.anonymous (default: false)
remove.anonymous.role (default: true)
add.application.roles (default: true)
add.authenticated.role (default: true)
```

The supported login modules are:

- The User Authentication Login Module
- The User Assertion Login Module
- The Identity Store Login Module
- The Asserted User

In Java SE applications, the default identity store service supports the User Authentication and the User Assertion login modules.

- The User Authentication Login Module
- The User Assertion Login Module
- The Identity Store Login Module
- The Asserted User

The User Authentication Login Module

Use the User Authentication login module in both Java EE or SE applications to authenticate a user with a given user name and password. The configuration of this

login module is available ready-to-use, and it authenticates against the domain identity store.

The following example illustrates the use of 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();
```

The User Assertion Login Module

Use the User Authentication login module in both Java EE and SE applications to assert a user identity in the identity store. This requires provisioning an `oracle.security.jps.JpsPermission` permission and implementing the `oracle.security.jps.callback.IdentityCallback` callback, as described in the following sections:

- Provisioning the jpsPermission
- Implementing the CallbackHandler
- Implementing the Programmatic Assertion

Provisioning the jpsPermission

The following example illustrates a grant allowing the `jdoe` principal permission 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>
```

> **See also:**
>
> Using Supported Permission Classes

Implementing the CallbackHandler

The following example 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);
                }
            }
        }
    }
```

Implementing the Programmatic Assertion

The following example 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();
            }
        });
```

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, and they are specified in the `jps-config-jse.xml` file.

The sequence in which a context lists the login modules in a stack is significant because the authentication algorithm takes this order into account in its computation. Ready-to-use, the identity store service is specified in the `system-jazn-data.xml` file. The service can be reconfigured to use an LDAP store.

OPSS APIs includes the `oracle.security.jps.service.login.LoginService` interface that allows a Java SE application to call not just all the login modules in a stack, but a subset of them in a prescribed order. The context passed to `LoginContext` in this interface determines the stack of login modules that your application uses.

The class associated with Identity Store login module is the `oracle.security.jps.internal.jaas.module.idstore.IdStoreLoginModule` class.

Properties specific to this login module include `remove.anononymous.role` and `add.application.role`.

You configure an instance of this module in the `jps-config-jse.xml` file:

```
<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>
```

The following sections explain the use of this module in Java SE applications:

- Using the Identity Store Login Module for Authentication
- Using the Identity Store Login Module for Assertion

Using the Identity Store Login Module for Authentication

This section describes the use of the Identity Store login module for basic user name and password authentication.

The following example illustrates how to configure the `appName` context:

```
<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>
```

The following example illustrates a callback that handles the name and password:

```
import javax.security.auth.Subject;
import javax.security.auth.login.LoginContext;
import javax.security.auth.callback.*;
import java.io.IOException;
Subject sub = new Subject();
CallbackHandler cbh = new YourCallbackHandler();
LoginContext context = new LoginContext(appName, subject,  cbh);
context.login();

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);
 }
          }
      }
  }
}
```

Using the Identity Store Login Module for Assertion

To use the Identity Store login module for assertion, provide a permission to execute the protected `setIdentity` method, and implement the callback handler that uses the `oracle.security.jps.callback.IdentityCallback` class.

The following example illustrates a configuration that allows the `MyApp` application permission 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>t
  </permissions>
</grant>
```

The following example illustrates the configuration that allows the `jdoe` user permission 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>
```

The following example 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 example 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",
 "password".toCharArray()));
            System.out.println("asserted user subject = " +
                               loginModuleExample.assertUser("testUser"));
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

The Asserted User

Applications often must execute actions as run-as operations. OPSS allows you to use the subjects of an asserted user to execute application logic as if it were executed by that user.

The following sections explains how to use the OPSS `SubjectSecurity` class to implement run-as operations:

- The SubjectSecurity Class
- Programming Guidelines and Recommendations
- Using SubjectSecurity

The SubjectSecurity Class

Here are several scenarios where you use this class:

- UC1 - A scheduler allows submitting and executing jobs. These jobs are 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 context.

- UC2 - A scheduled job is executed by an entity bean that calls other entity beans to complete the job, and all the code to complete the job must execute in the user's context. Therefore this context must 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 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 must persist the anonymous user. Eventually, the job must be executed by the anonymous user.
- UC5 - An MBean running in an application is called from a UI via an MBean server. The MBean operation call a series of diagnostic tests that must be run as a particular user.
- UC6 - An identity has been asserted and the assertion has produced a user context that can be used to run operations as that user.

Programming Guidelines and Recommendations

The `SubjectSecurity` class allows you to execute code either when a precomputed subject is available, or when the user must first be asserted. If the application context has not been set, then before you use the `SubjectSecurity` class, set its ID:

```
AppSecurityContext.setApplicationID(applicationID)
```

If you have set the application context, then the call to `setApplicationID` is optional.

When programming with the `SubjectSecurity` class:

- Use `SubjectSecurity.getActionExecutor(subject)` if you have already obtained a subject and want to execute some operation with it.
  - If you obtained the subject with the `SubjectUtil.getCurrentSubject` method, then that subject works with the container security and OPSS, and application roles are considered.
  - If you obtained the subject programmatically using the `JpsLoginModuleFactory` class, then that subject works with the container security and OPSS.
- Use `SubjectSecurity.getActionExecutor(userName)` if just the user name is available and want to execute some operation as that user. To use this method, provide a codesource 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.

- Note that `ActionExecutor.execute(PrivilegedAction)` calls `Subject.doAs` with the 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 the `getActionExecutor` method, and executes the action.

For information about the `oracle.security.jps.runtime.SubjectSecurity` class and the `oracle.security.jps.runtime.ActionExecutor` interface, see *Java API Reference for Oracle Platform Security Services*.

Using SubjectSecurity

The following example illustrates how to use the `SubjectSecurity` class to run as the asserted `jdoe` user:

```
// 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
        }
```

Using the Login Modules in Java SE Applications

To call a login module in Java SE applications, use the `getLoginContext` method of the `oracle.security.jps.service.login.LoginService` interface.

Similar to the `LoginContext` method in the standard Java Authorization and Authentication Services (JAAS) API, the `getLoginContext` method returns a `LoginContext` object instance that you use to authenticate a user, but, more generally, it allows you to use any number of login modules and in any order. Authentication is performed on just those login modules and in the order you passed them.

The following example illustrates user authentication against a subset of login modules with the `getLoginContext` method:

```
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 these login modules in the order listed in the array. Each login module in the array is listed in the default context in the `jps-config-jse.xml` file.

The following example 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.JpsUserAuthenticat
ionLoginModule"/>
   <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>
```

Authorization in Java SE Applications

Use a file, LDAP, or database-based security store in your Java SE applications and configure all services in the `jps-config-jse.xml` file. In this file, you also set system properties appropriate to your applications.

The following sections describe the configuration of policy and credential stores:

- Configuring Policy and Credential File Stores
- Configuring Policy and Credential LDAP Stores
- Configuring Database-Based Security Stores
- File Store Unsupported Methods

> **See also:**
>
> OPSS System Properties

- Configuring Policy and Credential File Stores
- Configuring Policy and Credential LDAP Stores
- Configuring Database-Based Security Stores
- File Store Unsupported Methods

Configuring Policy and Credential File Stores

A file policy store is specified in the `system-jazn-data.xml` file. A file credential store is specified in the `cwallet.sso` file.

The following example illustrates policy and credential configurations:

```
<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 Credential Store Instance</description>
  </serviceInstance>

  <serviceInstance name="policystore.xml" provider="policystore.xml.provider"
location="./system-jazn-data.xml">
   <description>File 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 that setting the `oracle.security.jps.policy.principal.cache.key` property to `false` in the policy store instance is required.

> **See also:**
>
> Using File Credential Stores
>
> modifyBootStrapCredential and addBootStrapCredential in *WLST Command Reference for Infrastructure Security*

Configuring Policy and Credential LDAP Stores

The examples in this section assume that the domain uses an LDAP store.

The following example illustrate the configurations of providers, instances, and context 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>

<serviceInstance location="./bootstrap" provider="credstoressp"
name="bootstrap.cred">
  <property value="./bootstrap" name="location"/>
</serviceInstance>

<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 example illustrates how to obtain programmatically a reference to the security store and assumes that you have set the `oracle.security.jps.config` system property to the location of the `jps-config-jse.xml` file:

```
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 ...
```

> **See also:**
>
> Prerequisites to Using the LDAP Security Store

Configuring Database-Based Security Stores

This sections presents configuration examples of database-based policy, credential, and keystores in the `jps-config-jse.xml` file. Note the points about the following example:

- The value of the `jdbc.url` property should be identical to the name of the Java Database Connectivity (JDBC) data source entered when you created the data source.
- The values of the bootstrap credentials (map and key) must match those you passed to the `addBootStrapCredential` WebLogic Scripting Tool (WLST) command when you created the bootstrap credential.

```
<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>
```

See also:

Prerequisites to Using the Database Security Store

Properties Common to OPSS Services

Keystore Service Properties

File Store Unsupported Methods

The file store does not support methods that involve cascading deletions. It supports only simple queries.

The following example illustrates a simple query that returns of all permissions with a display name matching `MyDisplayName`:

```
PermissionSetSearchQuery query = new PermissionSetSearchQuery();
query.addQuery(PermissionSetSearchQuery.SEARCH_PROPERTY.DISPLAY_NAME,
               false,
               ComparatorType.EQUALITY,
               "MyDisplayName",
               BaseSearchQuery.MATCHER.EXACT);
getPermissionSets(query);
```

Cascading deletions relates to any method whose signature includes the Boolean `cascadeDelete` argument. The only value allowed in case of a file store is `false`.

Audit in Java SE Applications

The following sections explain how to use Oracle Platform Security Services Common Audit Framework to audit events in Java SE applications and some common audit usage scenarios:

> **See also:**
>
> Using OPSS in Java SE Applications

- About Audit in Java SE Applications
- Configuring the Audit Bus-Stop Directory
- Configuring Audit Loaders
- Common Audit Scenarios in Java SE Applications

About Audit in Java SE Applications

In Java SE applications, the audit configuration is specified in the `jps-config-jse.xml` file:

```
<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 lets your application capture specific runtime events that are recorded in bus-stop files. The audit loader periodically moves data from bus-stop files to the audit store.

See also:

Configuring the Audit Bus-Stop Directory

Configuring Audit Loaders

Common Audit Scenarios in Java SE Applications

Managing Bus-Stop Files

About Audit Logs and Bus-stop Files

Audit Service Properties

Configuring the Audit Bus-Stop Directory

The bus-stop file names follow the format `host_pid_audit_major_minor.log`, as in `sample.myhost.com_12345_audit_1_0.log`. Note that the host name, the process ID, and the version number are embedded in the file name.

Your Java SE application must use a writable directory to which the service writes audit bus-stop files. To specify this location, set the `audit.logDirectory` property with the `setAuditRepository` WLST command, or set the `opss.audit.logDirectory` system property. If the runtime process is unable to determine a writable directory, then audit may not work.

Java SE applications can share the bus-stop directory if you set it with the `audit.logDirectory` property and all the Java SE applications in the environment use the same `jps-config-jse.xml` file, or the same value is passed as a system property to all Java SE applications.

See also:

setAuditRepository in *WLST Command Reference for Infrastructure Security*

Configuring Audit Loaders

The audit loader is a process that moves audit data from bus-stop files to a database. The `audit.loader.repositoryType` system property must be set in Java EE and SE applications. By default, it is set to database.

Java SE applications can use two kinds of loaders:

- The audit loader, a thread started by runtime audit. Ready-to-use, the audit loader is disabled; to enable it, set `audit.loader.enable=true`.
- The standalone audit loader for Java SE applications, which Oracle recommends, especially if you are using a shared directory.

> **See also:**
>
> Configuring Standalone Audit Loader

Common Audit Scenarios in Java SE Applications

The following sections describe common scenarios that implement audit in Java SE applications. We consider environments with and without a collocated WebLogic Server. A collocated environment is a WebLogic Server domain where Java Required Files (JRF) are installed.

- Configuring Audit with a Collocated WebLogic Server
- Configuring Audit Without a Collocated WebLogic Server

Configuring Audit with a Collocated WebLogic Server

To set audit in your the domain:

1. Set up the audit loader repository with Fusion Middleware Control or use the `setAuditRepository` WLST command:

   ```
   # 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");
   ```

   The command updates the `jps-config.xml` and `jps-config-jse.xml` files. A standalone audit loader is not needed if the writable audit directory for Java SE applications is the same directory as the log directory for WebLogic Server.

2. If you choose to have a separate writable audit directory for Java SE applications, then start the standalone audit loader and use a system properties, such as `oracle.instance`, to set the value of the bus-stop directory:

   ```
   $JAVA_HOME/bin/java
   -classpath $COMMON_COMPONENTS_HOME/modules/oracle.jps_12.2.1/jps-manifest.jar:
   $COMMON_COMPONENTS_HOME/modules/oracle.jdbc_12.2.1/ojdbc6dms.jar:
   $COMMON_COMPONENTS_HOME/modules/oracle.iau_12.2.1./jps-manifest.jar
   -Doracle.instance=$ORACLE_INST
   -Doracle.security.jps.config=path_to_jps-config-jse.xml
   oracle.security.audit.ajl.loader.StandaloneAuditLoader
   ```

Configuring Audit Without a Collocated WebLogic Server

The following task assumes that a middleware home is available but no WebLogic Server is running, so neither Fusion Middleware Control nor the `setAuditRepository` WLST command is available.

To set up audit in this case:

1. Specify the bus-stop directory with the `opss.audit.logDirectory` or `oracle.instance` system properties. Ready-to-use, audit is enabled in Java SE applications if you provide a writable directory to the Java Virtual Machine (JVM).
2. Use the `addBootStrapCredential` WLST command to add the bootstrap credential to the bootstrap credential store:

   ```
   addBootStrapCredential(jpsConfigFile='../../../user_projects/domains/
   base_domain/config/fmwconfig/jps-config-jse.xml', map='AuditDbPrincipalMap',
   key='AuditDbPrincipalMap', username='TEST_IAU_APPEND', password='password')
   ```

3. Add the `audit.loader.jdbcString` property to the audit configuration in the `jps-config-jse.xml` file with a value such as `audit.loader.jdbcString=jdbc:oracle:thin:@host:1521:sid`
4. If several Java SE applications share an audit directory, then start the standalone audit loader to move records to the database.
5. If every Java SE application writes to its own audit directory, then enable the runtime service's audit loader by `specifying audit.loader.enable=true`.

   Otherwise, if the audit directory is shared, then start the standalone audit loader.

See also:

Running Standalone Audit Loader

Part V
Reference

This part contains the following appendixes:

- OPSS Configuration File Reference
- File Store References
- Oracle Fusion Middleware Audit Framework Reference
- User and Role API Reference
- Administration with Scripts and MBeans
- OPSS System and Configuration Properties
- OPSS API References
- Using an OpenLDAP Identity Store
- Configuring Adapters for Identity Virtualization
- Troubleshooting OPSS

A

OPSS Configuration File Reference

This appendix describes the hierarchy of elements and attributes in the `jps-config.xml` and `jps-config-jse.xml` configuration files, typically located in the `$DOMAIN_HOME/config/fmwconfig` directory.
This appendix includes the following sections:

> **See also:**
>
> Configuring Services with MBeans

- <values>

First and Second Hierarchy Levels

The specifications in the OPSS configuration file apply to an entire domain, that is, to all Managed Servers and applications deployed in the domain.

The top element in the `jps-config.xml` file is the `jpsConfig` element. 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.

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 `<propertySet>`, `<serviceProvider>`, and `<serviceInstance>`.
<propertySets> {0 or 1} <propertySet> {1 or more} <property> {1 or more}	Groups one or more `<propertySet>` elements so that they can referenced as a group.
<extendedProperty> {0 or more} <name> {1} <values> {1} <value> {1 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.
<extendedPropertySets> {0 or 1} <extendedPropertySet> {1 or more} <extendedProperty> {1 or more} <name> {1} <values> {1} <value> {1 or more}	Groups one or more `<extendedPropertySet>` elements so that they can referenced a group.

Table A-1 (Cont.) First- and Second-Level Elements in jps-config.xml

Elements	Description
<serviceProviders> {0 or 1} <serviceProvider> {1 or more} <description> {0 or 1} <property> {0 or more}	Groups one or more <serviceProvider> elements, each of which defines an implementation of an OPSS service, such as a provider or a login module.
<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}	Groups one or more <serviceInstance> elements, each of which configures and specifies property values for a service provider defined in a <serviceProvider> element.
<jpsContexts> {1} <jpsContext> {1 or more} <serviceInstanceRef> {1 or more}	Groups one or more <jpsContext> elements, each of which is a collection of service instances that an application can use.

Third and Lower Hierarchy Levels

This section describes, in alphabetical order, the complete set of elements that can occur under the elements described in the First and Second Hierarchy Levels.

- <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 illustrates a description:

```
<serviceProvider ... >
   <description>file IdStore Provider</description>
   ...
</serviceProvider>
```

<extendedProperty>

This element defines an extended property in the following locations:

- Directly under <jpsConfig>, it defines an extended property for general use. It can specify, for example, all the base DNs in LDAP authentication providers.
- Directly under <extendedPropertySet>, it defines an extended property set.
- Directly under <serviceInstance>, it defines an extended property for a particular service instance.

An extended property 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 you create 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. Each `<jspContext>` must have a unique name.

Attributes

Name	Description
`name`	The name for the OPSS context. Contexts must have a unique names. 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 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 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) The default context must configure all 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-2 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 property set with multiple values.
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 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 Java Authorization and Authentication Services (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>OAM 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, then 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 instance or login module instance.

Each provider instance specifies the name of the instance, used to refer to the provider within the configuration file 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, use the procedure explained in Configuring Services with Scripts

Set properties and extended properties of a service instance in the following ways:

- Set properties directly with <property> subelements.
- Set extended properties directly with <extendedProperty> subelements.
- Refer to previously defined sets of properties with <propertySetRef> subelements.
- Refer to previously defined sets of extended properties with <extendedPropertySetRef> subelements.

Attributes

Name	Description
`name`	Designates a name for this service instance. 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 identity store that uses the `location` property to specify the file location:

```
<serviceInstances>
   <serviceInstance name="idstore.xml" provider="idstore.xml.provider">
      <!-- Subscriber name must be defined for file Identity Store -->
      <property name="subscriber.name" value="jazn.com"/>
      <!-- This is the location of file Identity Store -->
      <property name="location" value="./system-jazn-data.xml"/>
   </serviceInstance>
...
</serviceInstances>
```

Example 2

The following example illustrates the configuration a credential store. 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 Instance</description>
      <property name="location"
                value="${oracle.instance}/config/JpsDataStore/JpsSystemStore" />
   </serviceInstance>
...
</serviceInstances>
```

Example 3

The following example illustrates the configuration of an LDAP identity store:

```
<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.JpsUserAuthenticationLo
ginModule"/>
    <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, authentication, policy, or login module, and the Java

class that implements to use 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 (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. Valid types are 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.
	Values: any valid type.
	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)
`class`	Specifies the fully qualified name of the Java class that implements this service provider instantiated when the service provider is created. 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:

```
<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 a 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 Store References

This appendix describes the hierarchy of elements and attributes in the `system-jazn-data.xml` and `jazn-data.xml` files that specify file identity stores. File identity stores are supported in Java SE applications only.
This appendix includes the following topics:

- <name>
- <owner>
- <owners>
- <permission>
- <permissions>
- <permission-set>
- <permission-sets>
- <policy-store>
- <principal>
- <principals>
- <provider-name>
- <realm>
- <resource>
- <resources>
- <resource-name>
- <resource-type>
- <resource-types>
- <role>
- <role-categories>
- <role-category>
- <role-name-ref>
- <roles>
- <type>
- <type-name-ref>
- <uniquename>
- <url>
- <user>
- <users>
- <value>
- <values>

File Store Hierarchy

This section describes the element hierarchy of the `system-jazn-data.xml` and `jazn-data.xml` files. The `<jazn-data>` element is the root element. The elements directly under this root element are:

- `<jazn-realm>`
- `<policy-store>`

- <jazn-policy>

The <jazn-principal-classes> and <jazn-permission-classes> elements and their subelements may appear in the system-jazn-data.xml file as subelements of the <policy-store> element, but they are for backward compatibility only.

Table B-1 Hierarchy of Elements in system-jazn-data.xml and jazn-data.xml

Element	Description
<jazn-data>	The top-level element in the system-jazn-data.xml file.
<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}	Specifies security realms, and the users and enterprise groups included in each realm.

Table B-1 (Cont.) Hierarchy of Elements in system-jazn-data.xml and jazn-data.xml

Element	Description
<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}	Configures application-level policies. Define roles at the application level, and then define members in the roles. Members of a role are users and other 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 and jazn-data.xml

Element	Description
| <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}	

Table B-1 (Cont.) Hierarchy of Elements in system-jazn-data.xml and jazn-data.xml

Element	Description
| | <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}	
<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.

File Store Elements and Attributes

The following sections describe the elements and attributes used in the `system-jazn-data.xml` and `jazn-data.xml` files:

- <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>

<actions>

This element specifies the operations permitted by the associated permission class. Values are case-sensitive and are specific to each permission implementation.

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 instantiated to represent a principal 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.

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. 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 referenced may be an application role, application role member, principal, enterprise group, or user, and it uniquely identifies the item. GUIDs are sometimes generated and used internally by OPSS.

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.

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 12.2.1 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 code sources) 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}
```

Examples of jazn-policy

This is the first example of 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>
```

This is the second example of 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-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 in each of them. It 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 for a resource type. Queries for resources of this type delegate to this 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 with the <class> element.

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.
- 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 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 specifies the name of a permission 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.

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, with a set of <permission> subelements.

The `system-jazn-data.xml` schema definition does not specify this as a required element, but OPSS runtime 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 example illustrates the configuration of a permission set:

```
<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 permission sets:

- The actions specified in a <member-resource> must match one or more of the actions specified for the resource type referenced in <resource-name-ref>.
- A <member-resources> can have multiple <member-resource> elements in it.
- Permission sets must have at least one resource.

- Permission sets can exist without being used in principals.

In addition, in a permission, the name, description, and display name are case-sensitive.

<permission-sets>

This element specifies a collection 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 policies. Under <applications> there is an <application> for each application. The policies are specified in a <jazn-policy> of each <application>.

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).

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 get these permissions, the subject should include all the 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 domain security tore. 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 in it.

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 example 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 that the name, description, and display names are case-sensitive.

<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 example 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 used when there are resources managed in a store other than the security store.

  When specified, the class in a <provider-name> element is used as a resource finder. Queries for resources of this type (with the `ResourceManager` method) delegate to this class instead of using the built-in resource finder.

- The class specification is required and case-sensitive.
- The description and display specifications are optional and case-insensitive.
- The action specification 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.

<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 Using checkPermission for an example.

<role-category>

This element specifies 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 Using 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 roles that belong to the 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 uniquely identifies the item. 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 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 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. Specify additional attributes for application-level roles with 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 describes how to use Oracle Fusion Middleware Audit Framework to create and develop reports from your audit data.
This appendix includes the following topics:

This appendix covers reports based on the report template model of Oracle Business Intelligence Publisher 10*g*. For information about a different approach based on the audit dynamic model, see Using Audit Analysis and Reporting .

- Audit Events
- The Audit Schema
- Audit Filter Expression Syntax
- Naming and Logging Audit Files

Audit Events

The following sections describe the components, the events, and the attributes that you use with audit:

- What Components Can Be Audited?
- System Categories and Events
- OPSS Event Attributes

What Components Can Be Audited?

The Audit Framework provides the foundation to audit Oracle Fusion Middleware components and applications, such as the following:

- OPSS
- Oracle Web Services Manager
- Oracle Directory Integration Platform
- Oracle HTTP Server

- Oracle Internet Directory

This appendix provides audit information for events generated by OPSS only. For information about audit in other components and applications, refer to the respective administration guides.

System Categories and Events

The Audit Framework allows you to audit events in several core platform security services, including:

- System events for OPSS services
- Core OPSS
- Identity Governance Services
- Identity virtualization

The following tables list specific events:

- Table C-1
- Table C-2
- Table C-3
- Table C-4

Table C-1 System Categories and Events

Category	Event	Description
UserSession	UserLogin UserLogins	In applications with multiple tiers, inner tiers often use some special user ID to log in to the next tier. These logins are considered in the separate Internal Logins category. The User Login/Logout events only records actions by regular users.
	UserLogout UserLogouts	An end user or administrator logs out.
	Authentication	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	An internal login between two tiers.
	InternalLogout	An internal logout between two tiers.
	QuerySession	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.
Authorization	CheckAuthorization	Set of authorization events.
Data Access	CreateDataItem Create a data item	
	DeleteDataItem	Delete a data item.

Table C-1 (Cont.) System Categories and Events

Category	Event	Description
	QueryDataItemAttributes	Query the attributes associated with a data item.
	ModifyDataItemAttributes	Modify the attributes associated with a data item, for example access.
AccountManagement	ChangePassword	
	CreateAccount	Create a user, group, or any principal account.
	DeleteAccount	Delete an account for a user, group, or other principal.
	EnableAccount	Enable an account for a user, group, or other principal
	DisableAccount	Disable an account for a user, group, or other principal.
	QueryAccount	Query the user's account.
	ModifyAccount	Modify the account attributes.
ServiceManagement	InstallService	Install or upgrade a service or an application.
	RemoveService	Uninstall 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	Call a service or an application.
	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	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
DataViaAssociate	NA	a communication channel between system components
	ReceiveDataViaAssoc	Receive data from an associated peer.
	SendDataViaAssoc	Send data to an associated peer.
DataItemContentAccess	CreateDataItemAssoc	Open a data item, for example a file.

Table C-1 (Cont.) System Categories and Events

Category	Event	Description
	TerminateDataItemAssoc	Close a data item, for example a file.
	QueryDataItemAssocContext	Query attributes of a data item, for example mode of access, size limits, access paths, and so on.
	ModifyDataItemAssocContext	Modify attributes of a data item.
	QueryDataItemContents	Read the data item.
	ModifyDataItemContent	Write or append to the data item.
Exceptional	StartSystem	Boot a system host.
	ShutdownSystem	Shut down the system.
	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	ConfigureAuditPolicy	Modify parameters that control audit, such as audit event filtering.
	ConfigureAuditRepository	Configure the audit storage type.

Table C-2 Core OPSS 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

Table C-2 (Cont.) Core OPSS Events

Event Category	Event Type	Attributes used by Event
	IsAccessAllowed	NA
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

Table C-2 (Cont.) Core OPSS Events

Event Category	Event Type	Attributes used by Event
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
	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

Table C-2 (Cont.) Core OPSS Events

Event Category	Event Type	Attributes used by Event
RolePolicyManagement	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
	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,

Table C-2 (Cont.) Core OPSS Events

Event Category	Event Type	Attributes used by Event
	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
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

Table C-2 (Cont.) Core OPSS Events

Event Category	Event Type	Attributes used by Event
	PermissionSetModification	InitiatorDN, GUID, CodeSource, Principals, Permission, PermissionClass, ManagedApplication, PermissionSetName, PolicyDomainName, ResourceActions, Cascade, ModifiedAttributeName, ModifiedAttributeValue, ModifiedAttributeValueOld, ResourceActionsOld

Table C-3 Identity Governance 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

> **See also:**
>
> Audit Events and Event Categories

OPSS Event Attributes

Table C-5 lists attributes of audit events.

Table C-5 Attributes of Audit Events

Namespace	Attribute Name	Description
common	ApplicationName	The Java EE application name.
	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 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 `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	DN 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 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.

Table C-5 (Cont.) Attributes of Audit Events

Namespace	Attribute Name	Description
	Resource	Identifies a resource being accessed, such as a web page, a file, a directory, a web service, or a document. The resource name combines the host name and the URI.
	RID	This is the relationship identifier. 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:

Audit Attribute Groups

The Audit Schema

Even though prebuilt reports use a subset of event attributes, the Audit Framework allows using the entire event attribute set in your custom reports.

Table C-6 and Table C-7 describe the audit schema. 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

Table C-6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
	IAU_ORGID	VARCHAR2(255 Bytes)	Yes	2
	IAU_COMPONENTID	VARCHAR2(255 Bytes)	Yes	3
	IAU_COMPONENTTYPE	VARCHAR2(255 Bytes)	Yes	4
	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

Table C-6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
	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
	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

Table C-6 (Cont.) The Audit Schema

Table Name	Column Name	Data Type	Nullable	Column ID
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
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 tables in the audit schema that 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)

Table C-7 (Cont.) Additional Audit Schema Tables

Table Name	Column Name	Data Type
	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
	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 - IAU_BOOLEAN_050	NUMBER
	IAU_INT_001 - IAU_INT_050	NUMBER
	IAU_LONG_001 - IAU_LONG_050	NUMBER
	IAU_FLOAT_001 - IAU_FLOAT_050	NUMBER
	IAU_DOUBLE_001 - IAU_DOUBLE_050	NUMBER
	IAU_STRING_001 - IAU_STRING_100	VARCHAR(2048)
	IAU_DATETIME_001 - IAU_DATETIME_050	TIMESTAMP
	IAU_LONGSTRING_001 - IAU_LONGSTRING_050	CLOB
	IAU_BINARY_001 - IAU_BINARY_050	BLOB
IAU_AuditService	IAU_ID	NUMBER
	IAU_TransactionId	VARCHAR(255)

Table C-7 (Cont.) Additional Audit Schema Tables

Table Name	Column Name	Data Type
IAU_USERSESSION	IAU_ID	NUMBER
	IAU_AuthenticationMethod	VARCHAR(255)

Audit Filter Expression Syntax

When you choose a custom audit policy, you have the option to specify a filter expression along with an event.

For example, use the following expression:

```
Host Id -eq "myhost123"
```

to enable the audit event for a particular host only. Enter this expression with the `setAuditPolicy` command.

An expression can be a Boolean 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 these 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"
```

The rules for literals are:

- Boolean literals are `true` or `false`.

- Date time literals must be enclosed in double quotes and can have different formats. For example, “June 25, 2016 2:00 pm", “06/25/2016 2:00 pm" are both valid.
- String literals are 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"
```

Naming and Logging Audit Files

In Java EE applications, the audit files names follow the pattern `audit*.log`. The current file name is audit.log.

When that file fills up (it reaches the configured maximum audit file size which is 100MB), it is renamed to `audit<n>.log` and records are written to a new `audit.log`. So the current logs are written to `audit.log` and old logs are found in `audit1.log`, `audit2.log`, and so on.

In Java SE applications and system components, the audit log files names follow the pattern `hostname_pid_audit*.log` and these files follow a cycle similar to that of log files in Java EE applications. The current log file name is `host_pid_audit.log`. Note that the process ID is embedded in log file names, as in `host_12345_audit.log`.

After you configure an audit store, the audit loader reads these files and transfers the records to the database. After transferring a log file (such as `audit2.log` or `host_11925_audit1.log`), it deletes the log file, but it never deletes the current log files `audit.log` or `host_pid_audit.log`.

For applications with audit definitions in the dynamic model, the file names follow the format `audit_major_minor.log`. Note that the file name has embedded the version number as in `audit_1_2.log`.

Log files follow the W3C extended logging format where:

- `#Fields` specifies all the fields in the rest of the file.
- `#Remark` specifies common attributes.
- Attributes are separated by spaces and missing attributes are indicated by a dash.

D

User and Role API Reference

This appendix describes the attributes and parameters you use to develop applications with the User and Role API for LDAP repositories.

> **Note:**
>
> The User and Role API is deprecated. Oracle recommends that you use instead the Identity Governance Framework and migrate usage to this framework. For information about this migration, see Migrating to Identity Directory API in *Developing Applications with Identity Governance Framework*.

This appendix includes the following sections:

Mapping User Attributes to LDAP Directories

Table D-1 lists user attributes in the `UserProfile.property` file and the attribute that corresponds in the directory servers supported. IBM Tivoli and OpenLDAP use the same set of parameters. Microsoft ADAM and Microsoft Active Directory use the same set of parameters.

Table D-1 User Attributes in Directory Servers

User Attribute	Oracle Internet Directory	Embedded LDAP Server	Microsoft Active Directory	ODS EE	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
BUSINESS_EMAIL	mail	mail	mail	mail	mail	mail

Table D-1 (Cont.) User Attributes in Directory Servers

User Attribute	Oracle Internet Directory	Embedded LDAP Server	Microsoft Active Directory	ODS EE	Novell eDirectory	OpenLDAP
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	generationqualifier	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 Directory Servers

User Attribute	Oracle Internet Directory	Embedded LDAP Server	Microsoft Active Directory	ODS EE	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	telephonenumber	telephonenumber	telephonenumber	telephonenumber	telephonenumber	telephonenumber
FIRST_NAME	givenname	givenname	givenname	givenname	givenname	givenname
TIME_ZONE	orcltimezone	-	-	-	-	-
MAIDEN_NAME	orclmaidenname	-	-	-	-	-
PASSWORD	userpassswo rd	userpassswo rd	userpassswo rd	userpassswo rd	userpassswo rd	userpassswo rd
DEFAULT_GROUP	orcldefaultpr ofilegroup	-	-	-	-	-
ORGANIZATION	o	o	o	o	o	o
HOME_PHONE	homephone	homephone	homephone	homephone	homephone	homephone
BUSINESS_MOBILE	mobile	mobile	mobile	mobile	mobile	mobile
UI_ACCESS_MODE	orcluiaccessi bilitymode	-	-	-	-	-
JPEG_PHOTO	jpegphoto	jpegphoto	jpegphoto	jpegphoto	jpegphoto	jpegphoto
MANAGER	manager	manager	manager	manager	manager	manager
TITLE	title	title	title	title	title	title
EMPLOYEE_NUMBER	employeenu mber	employeenu mber	employeenu mber	employeenu mber	employeenu mber	employeenu mber
LDUser.PASSWORD	userpasswor d	userpasswor d	userpasswor d	userpasswor d	userpasswor d	userpasswor d

Mapping Role Attributes to LDAP Directories

Table D-2 lists each role attribute in UserProfile.property and its corresponding attribute in different directory servers. IBM Tivoli and OpenLDAP use the same set of parameters. Microsoft ADAM and Microsoft Active Directory use the same set of parameters.

Table D-2 Role Attributes in Directory Servers

Role Attribute	Oracle Internet Directory	Embedded LDAP Server	Microsoft Active Directory	ODS EE	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

Default Configuration Parameters

This section lists default configuration parameter values and the source of the value in different directory servers.

Table D-3 lists parameter values for Oracle Internet Directory and Microsoft Active Directory. Note that Active Directory requires SSL when setting sensitive information like passwords.

Table D-3 Oracle Internet Directory and Microsoft Active Directory Parameters

Parameter	Oracle Internet Directory	Microsoft 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.) Oracle Internet Directory and Microsoft Active Directory Parameters

Parameter	Oracle Internet Directory	Microsoft 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	Binary Attribute	Binary Attribute + {objectguid, unicodepwd}
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Note:

The Binary Attributes include photo, personalsignature, audio, jpegphoto, javaSErializeddata, thumbnailphoto, thumbnaillogo, userpassword, usercertificate, cacertificate, authorityrevocationlist, certificaterevocationlist, crosscertificatepair, and x500UniqueIdentifier.

The config attribute is extracted from the meta information present in the directory. The schema attribute is extracted from the schema in the directory.

Table D-4 lists parameters for Oracle Directory Server Enterprise Edition and Novell eDirectory.

Table D-4 Directory Server Enterprise Edition and Novell eDirectory Parameters

Parameter	DS EE	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" }
RT_GROUP_MANDATORY_ATTRS	#schema	#schema

Table D-4 (Cont.) Directory Server Enterprise Edition and Novell eDirectory Parameters

Parameter	DS EE	Novell eDirectory
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	Binary Attribute	Binary Attribute + {objectguid, unicodepwd}
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Note:

The Binary Attributes include photo, personalsignature, audio, jpegphoto, javaSErializeddata, thumbnailphoto, thumbnaillogo, userpassword, usercertificate, cacertificate, authorityrevocationlist, certificaterevocationlist, crosscertificatepair, and x500UniqueIdentifier.

The config attribute is extracted from the meta information present in the directory. The schema attribute is extracted from the schema in the directory.

Table D-5 lists the parameters for OpenLDAP and Oracle Virtual Directory.

Table D-5 OpenLDAP and Oracle Virtual Directory Parameters

Parameter	OpenLDAP	Oracle Virtual Directory
RT_USER_OBJECT_CLASSES	{"inetorgperson", "person", "organizationalperson" }	{"inetorgperson"}
RT_USER_MANDATORY_ATTRS	#schema	#schema

Table D-5 (Cont.) OpenLDAP and Oracle Virtual Directory Parameters

Parameter	OpenLDAP	Oracle Virtual Directory
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	Binary Attribute	Binary Attribute + {objectguid, unicodepwd}
ST_LOGGER_NAME	oracle.idm.userrole	oracle.idm.userrole

Note:

The Binary Attributes include photo, personalsignature, audio, jpegphoto, javaSErializeddata, thumbnailphoto, thumbnaillogo, userpassword, usercertificate, cacertificate, authorityrevocationlist, certificaterevocationlist, crosscertificatepair, and x500UniqueIdentifier.

The config attribute is extracted from the meta information present in the directory. The schema attribute is extracted from the schema in the directory.

Table D-6 lists the embedded LDAP server parameters.

Table D-6 Embedded LDAP Parameters

Parameter	Default
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
ST_BINARY_ATTRIBUTES	*(BBA) See note below about BBAs.
ST_LOGGER_NAME	oracle.idm.userrole

E

Administration with Scripts and MBeans

This appendix describes the administrative tasks that you carry out with WebLogic Scripting Tool (WLST) commands and the OPSS MBean API.
It includes the following sections:

- Configuring Services with Scripts
- Configuring Services with MBeans
- Restricting Access to MBeans

Configuring Services with Scripts

If your application uses the User and Role API and must access a provider user attribute different from the default `cn` attribute, then you must configure the authentication provider to use the desired user attribute and initialize the provider properly.

Use the following procedure to create a script that changes the provider initialization, so that the User and Role API uses the specified user attribute to access data in the configured provider:

1. Create a 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. Replace `userName`, `userPass`, `localHost`, and `portNumber` by the appropriate values 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 called. Assume that the script is saved as `/tmp/updateServiceInstanceProperty.py`.

3. Change to the `$ORACLE_HOME/common/bin` directory and run `wlst.sh`:

```
>cd $ORACLE_HOME/common/bin
>wlst.sh /tmp/updateServiceInstanceProperty.py -si servInstName -key propKey
-value propValue
```

   where:

   - `servInstName` is the name of the service instance provider whose properties you want to modify.
   - `propKey` identifies the name of the property to insert or modify.
   - `propValue` is the name of the value to add or update.

   The command modifies the `$DOMAIN_HOME/config/fmwconfig/jps-config.xml` domain configuration file by adding or updating a property to the passed instance provider. If you pass a key that matches the name of a property, then that property is updated with the passed value.

4. Restart Oracle WebLogic Server.

Example E-1 Example of Use

Assume that the domain uses the `idstore.ldap` authentication provider. Then:

```
wlst.sh /tmp/updateServiceInstanceProperty.py -si idstore.ldap -key "myPropName"
-value "myValue"
```

adds (or updates) the specified property of that instance provider:

```
<serviceInstance provider="idstore.ldap.provider" name="idstore.ldap">
   ...
   <property name="myPropName" value="myValue"/>
   ...
</serviceInstance>
```

Configuring Services with MBeans

OPSS provides a set of JMX-compliant Java EE Beans that are used by Fusion Middleware Control and scripts to manage, configure, and monitor services. Use of MBeans is recommended in Java EE applications only.

The following sections explain how to use OPSS MBeans:

- Supported OPSS MBeans
- Using OPSS MBeans
- Programming with OPSS MBeans

> **See also:**
>
> OPSS API References

Supported OPSS MBeans

Table E-1 lists the MBeans that OPSS supports, 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 Object Name
`Jps Configuration`	Manages domain configuration in `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. Update or write operations do not require server restart to effect changes. All changes take place immediately. Access is restricted to security administrators only.	`com.oracle.jps:type=JpsCredentialStore`
`Global Policy Store`	Manages global policies in the security store configured in the default context. Update or write operations do not require server restart to effect changes. All changes take place immediately.	`com.oracle.jps:type=JpsGlobalPolicyStore`
`Application Policy Store`	Manages application policies in the security store configured in the default context. Update or write operations do not require server restart to effect changes. All changes take place 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`

Using OPSS MBeans

To call an OPSS MBean, write a script and run it using WLST, or write a Java program, or use the MBean browser in Fusion Middleware Control.

To call an OPSS MBean with Fusion Middleware Control:

1. In the appropriate domain, first choose **AdminServer**, and then **System MBean Browser.** The **System MBean Browser** page is displayed.
2. In that page, expand the nodes **Application Defined MBeans**, **com.oracle.jps** and **Domain**.
3. Choose an MBean and use the **Attributes**, **Operations**, and **Notifications** tabs in the right pane to inspect the MBean attribute values and the methods.

For example, to retrieve a credential with a given map and key, use scripting to call the MBean operation `JpsCredentialMXBean.getPortableCredential(map, key)`.

> **See also:**
>
> Programming with OPSS MBeans
>
> Navigating MBeans (WLST Online) in *Understanding the WebLogic Scripting Tool*

Programming with OPSS MBeans

The following example illustrates how to call the `JpsConfiguration` MBean. Note that:

- It assumes that the following JAR files are in the class path:
  - `$ORACLE_HOME/oracle_common/modules/oracle.jps/jps-api.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.jps/jps-mbeans.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.jmx/jmxframework.jar`
  - `$ORACLE_HOME/oracle_common/modules/oracle.idm/identitystore.jar`
  - `$WEBLOGIC_HOME/server/lib/wljmxclient.jar`
- The connection is established by the `init` method.
- 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 = "password";
        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.CustomXmlEnterpriseRole
Impl", "oc4j-app-administrators", PortablePrincipal.PrincipalType.CUSTOM);
        PortablePrincipal[] arrPrincs = {CUSTOM_JDOE, CUSTOM_APP_ADMINS};
        //codesource 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.CredentialAccessPermission",
"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();
        }
    }
}
```

Restricting Access to MBeans

The information in this section is not restricted to OPPS MBeans but applies, more generally, to Oracle Fusion Middleware MBeans.

A *logical role* is a role specified declaratively or programmatically by a Java EE application. It is defined in an application deployment descriptor and used in the application code. You can map logical roles to enterprise groups or users, but you cannot map these roles to application roles.

The security access to MBeans is based on logical roles rather than on security permissions. MBeans are annotated with role-based constraints that are enforced at runtime by the JMX Framework.

The following sections illustrate the use of annotations, list the particular access restrictions, and explain the mapping of logical roles to enterprise groups:

- Annotation Examples
- Mapping Logical Roles to Enterprise Groups
- Particular Access Restrictions

Annotation Examples

The following example illustrates the use of 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;
  …
}
```

where:

- `@AtrributeGetterRequiredGlobalSecurityRole` specifies that the user must belong to the role `Operator` to access the get `isActive` method.
- `@AtrributeSetterRequiredGlobalSecurityRole` specifies that the user must belong to the `Admin` role to access the `setActive` method.
- `@OperationRequiredGlobalSecurityRole` specifies that the user must belong to the `Admin` role to access the `setActiveVirtualScreenId` method MBean.

Note that all these three annotations apply just to a specific item in the interface.

The following example illustrates the use of an 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 { … }
```

`@MbeanRequiredGlobalSecurityRole` specifies that the user must belong to the `Admin` role to access any operation or attribute of the MBean, so its scope is the entire MBean. Annotations with method or attribute scope override annotations that apply to the entire MBean.

`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.

Mapping Logical Roles to Enterprise Groups

Table E-2 shows the mapping of logical roles to enterprise groups.

Table E-2 Mapping of Logical Roles to WebLogic Server Groups

Logical Role	Default Permissions	WebLogic Group
`Admin`	Read and write access to all MBeans	`Admin`
`Configurator`	Read and write access to configuration MBeans	`Admin`
`Operator`	Read access to configuration MBeans. Read and write access to runtime 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`

> **See also:**
>
> Users, Groups, and Security Roles in *Securing Resources Using Roles and Policies for Oracle WebLogic Server*

Particular Access Restrictions

By default, all MBean write and update operations require that the user be a member of the `Admin` or `Configurator` roles. In addition, operations annotated with the `@Impact(value=1)` tag require that the user be a member of the `Admin` role, and operations annotated with the tag `@Impact(value=0)` require that the user be a member of the `Admin` or `Operator` roles.

Table E-3 describes the roles required to access attributes and operations in MBeans:

Table E-3 Roles Required per Operation

Operations with Impact Value	MBean Type	Requires One 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
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 and configuration properties you set at the server startup.
It includes the following sections:

- OPSS System Properties
- OPSS Configuration Properties

All OPSS system and configuration changes require server restart to take effect.

> **See also:**
>
> Programming with OPSS MBeans

- OPSS System Properties
- OPSS Configuration Properties

OPSS System Properties

A system property that has been introduced or modified is not in effect until the server is restarted. To set a system property, edit `setDomainEnv.sh` and add the property to the `EXTRA_JAVA_PROPERTIES` variable.

Table F-1 lists the system properties available with OPSS.

Table F-1 OPSS System Properties

System Property Name	Specifies
`common.components.home`	The location of the common components home. Required for both Java EE and SE applications. No default value.
`java.security.debug`	The permission failure when `JpsAuth.checkPermission` is called inside a `Subject.doAs` block and the permission check fails. Setting `jps.auth.debug` or `jps.auth.debug.verbose` is not enough to get a failure notification in this case. Optional.
`java.security.policy`	The location of the Java security policy file.

Table F-1 (Cont.) OPSS System Properties

System Property Name	Specifies
`jps.app.permissioncollectionmap.size`	The number of permission collection map entries kept in memory. Each entry corresponds with a set of permissions. It requires that you set `jps.policystore.ref.useSoftHardMapForSelectedMaps` to `true`. Optional. Valid values: a positive integer. Default value: 512.
`jps.authz`	The delegation of calls to the `AccessController.checkPermission` Java SE Development Kit (JDK) method that reduces runtime and debugging overhead. Optional. Valid values: `NULL`, `SM`, `ACC`, and `DEBUG_NULL`. No default value.
`jps.auth.debug`	The server logging output. Default value: `false`. Optional.
`jps.auth.debug.verbose`	The server logging output. Default value: `false`. Optional.
`jps.combiner.optimize`	The caching of a subject's protection domain. Optional. Valid values: `true`, `false`. Default value: `false`.
`jps.combiner.optimize.lazyeval`	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`	The number of combiner map entries kept in memory. Each entry corresponds with a set of principals. It requires that you set `jps.policystore.ref.useSoftHardMapForSelectedMaps` to `true`. Optional. Valid values: a positive integer. Default value: `128`.
`jps.deployment.handler.disabled`	The migration of policies and credentials for applications deployed on a WebLogic Server. Valid only for WebLogic Server. Set to `true` to disable the migration of application policies and credentials for all applications deployed on the server regardless of application settings in the `weblogic-application.xml` file. Optional. Valid values: `true`, `false`. Default value: `false`.

Table F-1 (Cont.) OPSS System Properties

System Property Name	Specifies
`jps.policystore.hybrid.mode`	The hybrid mode. When it is enabled, the policy provider reads from the `java.policy`, `weblogic.policy` file, and the security store reads from the `jps-config.xml` file. Optional. Valid values: `true`, `false`. Default value: `true`.
`jps.policystore.ref.useSoftHardMapForSelectedMaps`	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 `false`, then the `SoftKeyHashMap` type is used. If `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 `jps.subjectmap.size`, `jps.combinermap.size`, and `jps.app.permissioncollectionmap.size` properties. Optional. Valid values: `true`, `false`. Default value: `false`.
`jps.subject.cache.ttl`	The number of milliseconds after which group membership changes are in effect. This value must be kept synchronized with the value `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`	The number of subject map entries kept in memory. Each entry corresponds with TTL information about a subject. For this setting to take effect, the `jps.policystore.ref.useSoftHardMapForSelectedMaps` property must be `true`. Optional. Valid values: a positive integer. Default value: 128.
`oracle.security.jps.config`	The path to the domain configuration `jps-config.xml` or `jps-config-jse.xml` files. 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`	The path to the directory of a codesource URL. Optional. No default value. For an example of use, see <url>.

Table F-1 (Cont.) OPSS System Properties

System Property Name	Specifies
`oracle.deployed.app.ext`	The extension of codesource URL. Optional. No default value. For an example of use, see <url>.
`oracle.security.jps.log.for.approle.substring`	The name of an application role that contains a specified substring. If the substring to match is unspecified, then it logs all application role names. Optional. No default value.
`oracle.security.jps.log.for.permeffect`	The grant that was granted or denied. If the value is unspecified, then it logs all grants (regardless whether they were granted or denied). Optional. No default value.
`oracle.security.jps.log.for.permclassname`	The name of the permission class that matches exactly a specified name. If the name to match is unspecified, then it logs all permission class names. Optional. No default value.
`oracle.security.jps.log.for.permtarget.substring`	The name of a permission target that contains a specified substring. If the substring to match is unspecified, then it logs all permission targets. Optional. No default value.
`oracle.security.jps.log.for.enterprise.principalname`	The name of the principal (enterprise user or enterprise role) that matches exactly a specified name. If the name to match is unspecified, then it logs all principal names. Optional. No default value.
`opss.audit.logDirectory`	The location of the audit log files for SE applications if it is not set in the `jps-config-jse.xml` configuration file. Optional. No default value. Valid values: any writeable directory.
`wlst.offline.log`	The location of the log file when running offline WLST. Optional. No default value. Valid values: <filename>, stdout, strerr, disable.
`wlst.offline.log.priority`	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.

Table F-1 (Cont.) OPSS System Properties

System Property Name	Specifies
`oracle.security.jps.policystore.resource.cache.size`	The number of resources kept in the resource cache for one application policy. Valid in Java EE and Java SE applications. Applies to Oracle Internet Directory and database stores. Optional. Default value: 1000.

See also:

System Properties

Configuring Services with Scripts

OPSS Configuration Properties

The following sections describe service properties:

- Properties Common to OPSS Services
- Policy Store Service Properties
- Credential Service Properties
- LDAP Identity Properties
- Properties Common to All LDAP Servers
- Trust Service Properties
- Audit Service Properties
- Keystore Service Properties
- Anonymous and Authenticated Roles Properties

- Properties Common to OPSS Services
- Policy Store Service Properties
- Credential Service Properties
- LDAP Identity Properties
- Properties Common to All LDAP Servers
- Trust Service Properties
- Audit Service Properties
- Keystore Service Properties
- Anonymous and Authenticated Roles Properties

Properties Common to OPSS Services

The following tables describe the OPSS properties common to all services except for the trust store service. For information about trust store service properties, see Trust Service Properties.

Table F-2 Common Properties — Properties valid in both Java EE and SE applications

Property Name	Specifies
`bootstrap.security.principal.key`	The key for the password credentials to access the LDAP store, stored in the `cwallet.sso` file. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Required. No default value. The ready-to-use value is `bootstrap`.
`bootstrap.security.principal.map`	The map for the password credentials to access the LDAP store, stored in the `cwallet.sso` file. Valid in Java EE and SE applications. Applies to LDAP and DB security 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 security stores. Required. No default value. Value example: `jdbc:oracle:thin:@xxx27.com:1345:asi102cn`
`ldap.url`	The URL of the LDAP security store, with the format `ldap://host:port`. Valid in Java EE and SE applications. Applies only to LDAP stores. Required. No default value.
`oracle.security.jps.farm.name`	The relative distinguished name format of the domain node in the LDAP store. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Required. No default value.
`oracle.security.jps.ldap.root.name`	The relative distinguished name format of the root node in the LDAP store. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Required. No default value.

Table F-2 (Cont.) Common Properties — Properties valid in both Java EE and SE applications

Property Name	Specifies
`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 SE applications. Applies to LDAP and DB security stores. Optional. Default value: 5000
`server.type`	The type of the security store. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Required. No default value. Values: `OID`, `DB_ORACLE`.

Table F-3 Common Properties — Properties valid in Java EE applications

Property Name	Specifies
`datasource.jndi.name`	The Java Naming and Directory Interface (JNDI) name of the Java Database Connectivity (JDBC) data source instance. Valid in Java EE applications only. Applies to only DB security 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 security 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 security stores. Optional. Default value: 15
`weblogic.dbuser.map` `weblogic.dbuser.key`	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 security stores. Optional. Default value: none.

Table F-3 (Cont.) Common Properties — Properties valid in Java EE applications

Property Name	Specifies
`oracle.security.jps.db.useWeblogicDBUserMapKey`	Where to find the map and key for the WebLogic DB user/password. This property is automatically set when reassociating to a DB security store. Valid in Java EE applications only. Applies to only DB security stores. Optional. Valid values: `true` or `false`. Default value: `false`. If `true`, then the `weblogic.dbuser.map` and `weblogic.dbuser.key` properties specify the credential's map and key for the WebLogic DB user/password. Otherwise, if `false` or unspecified, then the `bootstrap.security.principal.map` and `bootstrap.security.principal.key` properties specify the credential's map and key for the WebLogic DB user/password.

Table F-4 Common Properties — Properties valid in Java SE applications

Property Name	Specifies
`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 LDAP and DB security 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 LDAP and DB security stores. Optional. No default value.
`jdbc.driver`	The JDBC driver. Valid in Java SE applications only. Applies to only DB security stores. Required. No default value. Value example: `oracle.jdbc.driver.OracleDriver`

See also:

Configuring Services with Scripts

Policy Store Service Properties

The following sections describe the policy store service properties:

- Policy Store Service Configuration
- Runtime Policy Configuration

> **See also:**
>
> Configuring Services with Scripts

- Policy Store Service Configuration
- Runtime Policy Configuration

Policy Store Service Configuration

The policy store provider class to use with LDAP or DB security stores is the `oracle.security.jps.internal.policystore.ldap.LdapPolicyStoreProvider` class.

Table F-5 describes the properties specific to policy store. Additional properties are listed in Properties Common to OPSS Services.

Table F-5 Policy Properties

Property Name	Specifies
`oracle.security.jps.policystore.resourcetypeenforcementmode`	Throwing exceptions if any of the following checks fail: • Verify that if two resource types share the same permission class, then 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 checks fail, then the system throws an exception and the operation is stopped. If set to `Lenient`, when any of the checks fail, then 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 SE applications. Applies to LDAP and DB security stores. Optional. Default value: `Lenient` Valid values: `Strict`, `Lenient`.

Example 1

The following example illustrates the configuration of a 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 example illustrates the configuration of an LDAP 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>
```

Example 3

The following example illustrates the configuration of DB security stores 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 example illustrates the configuration of a DB policy store 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>
```

Runtime Policy Configuration

The runtime policy store provider class you use with LDAP or DB security stores is the `oracle.security.jps.az.internal.runtime.provider.PDPServiceProvider` class.

Table F-6 lists the runtime properties of policy store instances.

Table F-6 Runtime Policy Properties

Property Name	Specifies
`oracle.security.jps.policystore.rolemember.cache.type`	The type of the role member cache. Valid in Java EE and SE applications. Applies to LDAP and DB security 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 SE applications. Applies to LDAP and DB security 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 efficient when the policy footprint is very small. Default value: `FIFO`.

Table F-6 (Cont.) Runtime Policy Properties

Property Name	Specifies
`oracle.security.jps.policystore.rolemember.cache.size`	The number of the roles kept in the member cache. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Optional. Default value: 1000.
`oracle.security.jps.policystore.policy.lazy.load.enable`	The policy lazy load. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Optional. Valid values: `true`, `false`. Default value: `true`.
`oracle.security.jps.policystore.policy.cache.strategy`	The type of strategy used in the permission cache. Valid in Java EE and SE applications. Applies to LDAP and DB security 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 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 SE applications. Applies to LDAP and DB security stores. Optional. Default value: 1000.
`oracle.security.jps.policystore.refresh.enable`	The policy store refresh. If this property is set, then `oracle.security.jps.ldap.cache.enable` cannot be set. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Optional. Valid values: `true`, `false`. Default value: `true`.
`oracle.security.jps.ldap.cache.enable`	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 SE applications. Applies to LDAP and DB security stores. Optional. Valid values: `true`, `false`. Default value: `true`.

Table F-6 (Cont.) Runtime Policy Properties

Property Name	Specifies
`oracle.security.jps.policystore.refresh.purge.timeout`	The number of milliseconds after which the security store cache is purged. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Optional. Default value: 43200000 (12 hours).
`oracle.security.jps.ldap.policystore.refresh.interval`	The number of milliseconds at which the security store is polled for changes. Valid in Java EE and SE applications. Applies to LDAP and DB security stores. Optional. Default value: 600000 (10 minutes).
`oracle.security.jps.policystore.rolemember.cache.warmup.enable`	The way the ApplicationRole membership cache is created. If `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` when the number of application roles is very high. Valid in Java EE and SE applications. Applies to LDAP and DB security 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 SE applications. Applies to file, LDAP, and DB security stores. Optional. Default value: the system temporary folder.
`oracle.security.jps.pdp.AuthorizationDecisionCacheEnabled`	The authorization cache is enabled. Valid in Java EE and SE applications. Applies to file, LDAP, and DB security 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 SE applications. Applies to file, LDAP, and DB security stores. Optional. Default value: 10

Table F-6 (Cont.) Runtime Policy Properties

Property Name	Specifies
`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 it is needed. Valid in Java EE and SE applications. Applies to file, LDAP, and DB security stores. Optional. Default value: 500
`oracle.security.jps.pdp.AuthorizationDecisionCacheTTL`	The number of seconds during which session data is cached. Valid in Java EE and SE applications. Applies to file, LDAP, and DB security stores. Optional. Default value: 60
`oracle.security.jps.policystore.resourcetypeenforcementmode`	Throwing 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` and any of the checks fail, then the system throws an exception and the operation is terminated. If set to `Lenient` and any of the checks fail, then 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 SE applications. Applies to LDAP and DB security stores. Optional. Default value: `Lenient` Valid values: `Strict`, `Lenient`.

Credential Service Properties

Table F-7 lists the properties specific to credential store instances. Additional properties are listed in Properties Common to OPSS Services.

Table F-7 Credential Store Properties

Property Name	Specifies
`encrypt`	To encrypt credentials. Valid in Java EE and SE applications. Applies only to file and LDAP stores. Valid values: `true`, `false`. Optional. Default value: `false`.

The following example illustrates the configuration of a credential store 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="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>
```

> **See also:**
>
> Configuring Services with Scripts

LDAP Identity Properties

Table F-8 lists the properties of LDAP identity stores, and states extended properties are and User and Role API properties.

Table F-8 LDAP Identity Store Properties

Property Name	Specifies

`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` - Novelle Directory
	`OPEN_LDAP` - OpenLdap
	`LIBOVD` - Oracle Library OVD
	`CUSTOM` - Any other type
	If using a custom authentication provider, then 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
`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

Table F-8 (Cont.) LDAP Identity Store Properties

Property Name	Specifies
`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.WlsLdapIdStoreConfigProvider`

`group.create.bases`	The base DNs used to create groups. Extended property.
	Valid in Java EE and 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 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

`group.filter.object.classes`	The fully qualified names of object classes used to search groups. Extended property. Valid in Java EE and SE applications. Optional. Value example: `groupOfUniqueNames`. Corresponding User and Role API property: ROLE_FILTER_OBJECT_CLASSES

Table F-8 (Cont.) LDAP Identity Store Properties

Property Name	Specifies

`group.mandatory.attrs`	The attributes that must be specified when creating groups. Extended property.
	Valid in Java EE and 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 a group. Extended property.
	Valid in Java EE and 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 groups. Extended property.
	Valid in Java EE and 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

`group.selected.create.base`	The base DNs for creating groups. Valid in Java EE and SE applications. Optional. Value example: `cn=users,dc=us,dc=abc,dc=com` (single DN) Corresponding User and Role API property: ROLE_SELECTED_CREATEBASE

Table F-8 (Cont.) LDAP Identity Store Properties

Property Name	Specifies
`groupname.attr`	The attribute that uniquely identifies the name of the group. Valid in Java EE and SE applications. Optional. Value example: `cn` Corresponding User and Role API property: ROLE_NAME_ATTR
`group.selected.search.base`	The base DNs for searching groups. Valid in Java EE and 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 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 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 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 SE applications. Optional. Value example: `<property name="user.login.attr" value="mail"/>` Corresponding User and Role API property: USER_LOGIN_ATTR

Table F-8 (Cont.) LDAP Identity Store Properties

Property Name	Specifies

`user.mandatory.attrs`	The attributes that must be specified when you create a user. Extended property.
	Valid in Java EE and 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 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 SE applications. Optional. Corresponding User and Role API property: USER_NAME_ATTR Note that if you reset the attribute `username`, then you must also reset `username.attr`.
`ldap.host`	The name of the system hosting the identity store. Valid in Java EE and SE applications. Optional.
`subscriber.name`	The default realm for the identity store. Valid in Java EE and SE applications. Optional. Value example: `dc=us,dc=oracle,dc=com`. Corresponding User and Role API property: ADF_IM_SUBSCRIBER_NAME

Table F-8 (Cont.) LDAP Identity Store Properties

Property Name	Specifies
`virtualize`	Where search and modifications are performed. If `true`, then searching and modifying is available in all configured authentication providers. If `false`, then searching and modifying is available in only the first provider in the configured stack. Set to `true` to use the User and Role API to search or write information in *all* providers. Valid in Java EE and SE applications. Optional. Valid values: `true` or `false`. Default value: `false`. Value example: `<property name="virtualize" value="true"/>`

The following example illustrates the configuration of an LDAP 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>
```

See also:

Configuring Services with Scripts

Properties Common to All LDAP Servers

Table F-9 lists properties common to LDAP servers.

In case of an LDAP identity store and 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 instance *must* include the following property:

```
<property
name="INITIAL_CONTEXT_FACTORY" value="com.sun.jndi.ldap.LdapCtxFactory"/>
```

Table F-9 Generic LDAP Properties

Property Name	Specifies
`connection.pool.authentication`	The type of LDAP connection that the JNDI connection pool uses. Valid in Java EE and SE applications. Optional. Values: `none`, `simple`, and `DIGEST-MD5`. Default value: `simple`.
`connection.pool.max.size`	The maximum number of connections in the LDAP connection pool. Valid in Java EE and SE applications. Optional. Value example: 30
`connection.pool.min.size`	The minimum number of connections in the LDAP connection pool. Valid in Java EE and SE applications. Optional. Value example: 5
`connection.pool.protocol`	The protocol to use for the LDAP connection. Valid in Java EE and SE applications. Optional. Values: `plain`, `ssl`. Default value: `plain`.
`connection.pool.provider.type`	The connection pool to use. Valid in Java EE and SE applications. Optional. Values: `JNDI`, `IDM`. Default value: `JNDI`.
`connection.pool.timeout`	The number of milliseconds that an idle connection can remain in the pool. After time-out, the connection is closed and removed from the pool. Valid in Java EE and SE applications. Optional. Default value: 300000 (5 minutes)
`oracle.security.jps.ldap.max.retry`	The maximum number of retry attempts if there are problems with the LDAP connection. Valid in Java EE and SE applications. Optional. Value example: 5

The following example illustrates a configuration of several properties:

```
<!-- 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"/>
```

> **See also:**
>
> Configuring Services with Scripts

Trust Service Properties

Table F-10 lists the properties specific to the trust service.

Table F-10 Truststore Properties

Property Name	Specifies
`merge.jdkcacerts.with.trust`	Whether to return public CA certificates in the keystore `kss://system/publiccacerts` with a keystore query to `kss://system/trust`. Set to `true` to have all certificates in `publiccacerts` included in the keystore query return. Set to `false` not to have them included in the query. Valid in Java EE and SE applications. Values: `true` or `false`. Optional. Default: `false`.
`trust.keystoreType`	The type of the truststore: Java Keystore (JKS) or keystore service (KSS) keystore. Valid in Java EE and SE applications. Optional. Valid values: `JKS` of `KSS`. Default: none. If unspecified and KSS is provisioned, then the value is `KSS`. Otherwise it is `JKS`.
`trust.keyStoreName`	The store name with the format: `kss://<stripeName>/<keyStoreName>` Applies only when `trust.keystoreType` is `KSS`. Valid in Java EE and SE applications. Optional. Default: `kss://opss/trustservice_ks`.
`trust.trustStoreName`	The store URL with the format: `kss://<stripeName>/<keyStoreName>` Applies only when `trust.keystoreType` is `KSS`. Valid in Java EE and SE applications. Optional. Default: `kss://opss/trustservice_ts`.

Table F-10 (Cont.) Truststore Properties

Property Name	Specifies
`trust.aliasName`	The alias to use to get an X.509 certificate and private key from the keystore. Valid in Java EE and SE applications. Optional. Default: the name of the Oracle WebLogic Server domain.
`trust.issuerName`	The name (included in the token) that the target trust service uses to pick up and validate the token. Valid in Java EE and SE applications. Optional. Default: the name of the WebLogic Server domain.
`trust.provider.className`	The fully-qualified name of the trust provider class. Valid in Java EE and SE applications. Required. Value: `the only supported value is oracle.security.jps.internal.trust.provider.embedded.EmbeddedProviderImpl`.
`trust.clockSkew`	The number of seconds the time-gap allowed when verifying time conditions. Valid in Java EE and SE applications. Optional. Default: 0.
`trust.token.validityPeriod`	The number of seconds that a token remains valid after being issued. Valid in Java EE and SE applications. Required. Default: none.
`trust.csf.map`	The map of the credential to access the keystore. Valid in Java EE and SE applications. Optional. Default: the value of the keystore instance property `keystore.csf.map`.
`trust.csf.keystorePass`	Applies only when `trust.keystoreType` is `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 SE applications. Optional. Default: the value of the keystore instance property `keystore.pass.csf.key`.

Table F-10 (Cont.) Truststore Properties

Property Name	Specifies
`trust.csf.keyPass`	The key of the credential to access the keystore (the map is set by `trst.csf.map`). Applies only when `trust.keystoreType` is `JKS` Valid in Java EE and SE applications. Optional. Default: the value of the keystore instance property `keystore.sig.csf.key`.
`trust.token.includeCertificate`	The Security Assertion Markup Language (SAML) token includes a certificate. Valid in Java EE and SE applications. Required. Valid values: `true` or `false`. Default: `false`.

The following example 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>
```

See also:

Configuring Services with Scripts

Audit Service Properties

Table F-11 lists the properties specific to audit. Additional properties are listed in Properties Common to OPSS Services.

Table F-11 Audit Properties

Property Name	Specifies	Required?	Values	Default Value
`audit.filterPreset`	The audit level.	no	None, Low, Medium, or High	None

Table F-11 (Cont.) Audit Properties

Property Name	Specifies	Required?	Values	Default Value
`audit.customEvents`	The custom events that to audit. 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	NA	NA
`audit.specialUsers`	The list of users whose activity is always audited, even if the `filterPreset` property is `none`.	no	NA	NA
`audit.maxFileSize`	The size of a bus-stop file where audit events are written. Integer is in Bytes	no	NA	104857600
`audit.loader.interval`	The number of seconds with which audit loader uploads to database.	no		15 seconds
`audit.loader .repositoryType`	The store type for the audit events. If type is Database (DB), then also define audit.loader.jndi or JDBC property.	yes	File, DB	File
audit.loader.jndi	The JNDI name of the data source in application servers for uploading audit events into database.	no	NA	jdbc/ AuditAppend DataSource
`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 a Java SE application and the repository type is DB.	no	NA	NA
`audit.loader.jdbc.string`	The JDBC string for JDBC connection when running a Java SE application and repository type is DB.	no	NA	
`audit.logDirectory`	The base directory for bus-stop files.	required for JavaSE	NA	jse
`audit.timezone`	Recording events using a specific time zone.	no	UTC, local	UTC
`audit.change.scanning.interval`	The number of milliseconds after which, the service checks for any changes.	no	whole number greater than zero	60000 (60 seconds)

The following example illustrates the use of properties in a 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>
```

> **See also:**
>
> Configuring Services with Scripts

Keystore Service Properties

Table F-12 lists the properties specific to the keystore. Additional properties are listed in Properties Common to OPSS Services.

Table F-12 Keystore Service Properties

Property Name	Specifies	Required?	Values	Default
`keystore.file.path`	The location of the file keystores.xml when file provider is configured.	Yes, if a file keystore provider is configured.	-	./
`ca.key.alias`	The key alias of the third party CA used for the keystore service instance.	No	-	-
`location`	The absolute or relative path. location of the keystore.	Yes, if keystore.type is JKS. No, if keystore.type is PKCS11 or HSM (LunaSA)	Path to keystore	./default-keystore.jks
`keystore.type`	The type of keystore.	No	KSS, JKS, PKCS11, Luna	JKS
`keystore.csf.map`	The credential store map name that OWSM uses. Used by OWSM only.	No	Credential store map name	oracle.wsm.security
`keystore.pass.csf.key`	The credential store key that points to Keystore password. Used by OWSM only.	No	Credential store csf key name	keystore-csf-key
`keystore.sig.csf.key`	The 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 by OWSM only.	No	Credential store csf key name or, for HSM, the direct alias	sign-csf-key
`keystore.enc.csf.key`	The 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 by OWSM only.	No	Credential store csf key name or, for HSM, the direct alias	enc-csf-key

The following example illustrates a keystore 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 example illustrates a keystore configuration for an LDAP 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 example illustrates a keystore configuration for a DB 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>
```

See also:

Configuring Services with Scripts

Anonymous and Authenticated Roles Properties

Table F-13 lists the properties that can be used to configure anonymous users, anonymous roles, and authenticated roles.

Table F-13 Anonymous and Authenticated Roles Properties

Property Name	Specifies
`anonymous.role.description`	The description of the anonymous role. Valid in Java EE and SE applications. Optional. No default value.
`anonymous.role.name`	The name of the principal in the anonymous role. Valid in Java EE and SE applications. Optional. Default value: `anonymous-role`
`anonymous.role.uniquename`	The name of the anonymous role. Valid in Java EE and SE applications. Optional. Default value: `anonymous-role`
`anonymous.user.name`	The name of the principal in the anonymous user. Valid in Java EE and SE applications. Optional. Default value: `anonymous`
`authenticated.role.description`	The description of the authenticated role. Valid in Java EE and SE applications. Optional. No default value.
`authenticated.role.name`	The name of the principal in authenticated user roles. Valid in Java EE and SE applications. Optional. Default value: `authenticated-role`
`authenticated.role.uniquename`	The name of the authenticated role. Valid in Java EE and SE applications. Optional. Default value: `authenticated-role`
`remove.anonymous.role`	The anonymous role to remove from the subject after a user is authenticated. Valid in Java EE and SE applications. Optional. Valid values: `true`, `false`. Default value: `false`.

See also:

Configuring Services with Scripts

G

OPSS API References

This appendix lists the reference documentation available for the APIs used with OPSS.

- OPSS API

  Java API Reference for Oracle Platform Security Services
- OPSS MBean API

  Java API Reference for Oracle Platform Security Services MBeans
- User and Role API

  Java API Reference for Oracle Platform Security Services User and Role
- Identity Directory API

  Java API Reference for Identity Governance Framework Identity Directory

H

Using an OpenLDAP Identity Store

This appendix describes the setup required to use OpenLDAP 2.2 as the repository for the identity store.
It includes the following section:

Using an OpenLDAP Identity Store

To use OpenLDAP 2.2 for the identity store:

1. Use Oracle WebLogic Server Administration Console to create a new authentication provider:
   - Choose OpenLDAPAuthenticator from the list of providers.
   - Set the control flag of the OpenLDAPAuthenticator to `SUFFICIENT`.
   - Set the control flag of the WebLogic Default Authenticator to `SUFFICIENT`.
   - Change the order 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 object class in the Group From Name Filter to something other than group of names.
2. From the directory where OpenLDAP is installed:
   - Open `slapd.conf` for edit.
   - Insert the following line in the include section at the top:

     ```
     include ./schema/inetorgperson.schema
     ```

   - Save the file and restart the OpenLDAP.

This procedure adds the `inetorgperson` object to every new external role you create in the OpenLDAP. That class is required to map external roles to an application roles.

Configuring Adapters for Identity Virtualization

This appendix describes how to configure, implement, and log messages for adapters used in identity virtualization.
This appendix includes the following sections:

- About Split Profiles
- Configuring Split Profiles
- Implementing Split Profiles
- Logging Identity Virtualization Library

About Split Profiles

A *split profile* is an identity whose attributes are stored in two (or more) sources. Identity virtualization supports split profiles and querying multiple LDAP directories in a single query. So when an application must obtain attributes from more than one source directory for an identity, it uses identity virtualization with split profiles. The adapter configuration is stored in the `adapters.os_xml` file, but connection parameters, such as host, port, and credentials, are obtained from the OPSS configuration.

When configuring the LDAP connection parameters, the `user.create.bases` and `group.create.bases` properties must correspond to the primary adapter's namespace.

> **See also:**
>
> *Administering Oracle Virtual Directory*
>
> - Understanding Oracle Virtual Directory Adapters
> - Understanding the Join View Adapter
>
> Identity Store Parameters

Configuring Split Profiles

To configure split profiles:

1. Set the `virtualize` property to true to enable queries against multiple LDAPs. For information about configuring the identity store, see Configuring the Identity Store .

2. Use the `createJoinAdapter` WLST command to create a join adapter in the primary identity store:

```
createJoinAdapter(adapterName="Join Adapter Name",  root="Namespace",
primaryAdapter="Primary adapter Name")
```

3. Use the `addJoinRule` WLST command to add the join rule to each secondary store:

```
addJoinRule(adapterName="Join Adapter Name", secondary="Secondary Adapter
Name", condition="Join Condition")
```

4. Use the `modifyLDAPAdapter` WLST command to modify adapters in all stores:

```
modifyLDAPAdapter(adapterName="AuthenticatorName", attribute="Visible",
value="Internal")
```

> **See also:**
>
> *WebLogic Scripting Tool Command Reference for Identity and Access Management*:
>
> - createJoinAdapter
> - addJoinRule
> - modifyLDAPAdapter

Implementing Split Profiles

Assume that Microsoft Active Directory is the primary authentication provider with the `cn=users,dc=acme,dc=com` user base, and Oracle Internet Directory is the secondary provider with the `cn=users,dc=oid,dc=com` user base.

To implement split profile with these two adapters:

1. Create a join adapter on the primary authentication provider:

```
createJoinAdapter(adapterName="JoinAdapter1", root="dc=acme,dc=com",
primaryAdapter="AD")
```

2. Add the created join adapter to the secondary authentication provider:

```
addJoinRule(adapterName="JoinAdapter1", secondary="OID", condition="uid=cn")
```

   where `uid=cn` indicates that if for a user, the `uid` value matches the `cn` value in Microsoft Active Directory, then the attributes are combined.

   The attribute on the left side of the equal sign is the attribute in the secondary adapter and the attribute on the right side is the attribute in the primary adapter.

3. Change the visibility of all adapters:

```
modifyLDAPAdapter(adapterName="OID", attribute="Visible", value="Internal")
modifyLDAPAdapter(adapterName="AD", attribute="Visible", value="Internal")
```

4. Restart Oracle WebLogic Server.

See also:

WebLogic Scripting Tool Command Reference for Identity and Access Management:

- createJoinAdapter
- addJoinRule
- modifyLDAPAdapter

Logging Identity Virtualization Library

To enable identity virtualization library logging:

1. Remove any previously configured identity virtualization library loggers.
2. Create the new logger named `oracle.ods.virtualization.accesslog` with the `NOTIFICATION` level.
3. Create a handler to specify the file associated with the logger, where all messages are logged.
4. Add `auditLogPublisher` to `the DOMAIN_HOME/config/fmwconfig/ovd/default/provider.os_xml` file:

```
<providers>
   ...
   <auditLogPublisher>
      <provider name="FMWAuditLogPublisher"></provider>
      <provider name="AccessLogPublisher">
<configClass>oracle.ods.virtualization.config.AccessLogPublisherConfig</
configClass>
         <properties>
            <property name="enabled" value="true"/>
         </properties>
      </provider>
   </auditLogPublisher>
   ...
</providers>
```

5. Restart WebLogic Server.

J

Troubleshooting OPSS

This appendix describes common problems that you may encounter when you configure or use OPSS and explains how to solve them.
It includes the following sections:

- The OPSS Diagnostic Framework
- Diagnosing Security Errors
- Troubleshooting Reassociation and Migration
- Troubleshooting Server Startup
- Troubleshooting Permissions
- Troubleshooting Connections and Access
- Oracle Business Intelligence Publisher Time Zone
- Troubleshooting Searching
- Troubleshooting Versions
- Troubleshooting Other Errors
- Need Further Help?

The OPSS Diagnostic Framework

OPSS includes a framework that helps you reduce the resolution time of problems. This framework allows you to extract internal states of a domain and dump that information to a file. The framework provides a number of tests to generate dumps that record the characteristics of domain servers and services in that domain.

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`
- Keystore tests
  - `oracle.security.jps.diag.test.JpsKeyStoreTest`
  - `oracle.security.jps.diag.test.JpsKeyStoreConfigTest`

Running a Test

To call a test, use the `executeDump` command with the following syntax:

```
executeDump(name='opss.diagTest', outputFile='dumpLocation',
args={'testtname':'testName'})
```

where:

- The first argument, `name='opss.diagTest'`, is fixed.
- `outputFile` specifies the location where the dump is generated relative to where the command is run.
- All character strings in the third argument are fixed except for the `testName` string, which you set to one of the tests. If you use the wild card *, 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
```

Example of Use

The following sequence illustrates a situation where you use the diagnose framework:

1. You receive a customer problem.

2. From the logged failure, you determine that the problem has to do with a connection (for example).
3. You run one or more tests in the domain in question, such as `oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTest`:
   a. Connect to Oracle WebLogic Server:

      ```
      wls:/offline> connect('adminuser', 'adminpass', 't3://localhost:7001')
      ```

   b. Call a test, such as the following:

      ```
      >executeDump(name='opss.diagTest', outputFile='myDumpTest',
      args={'testname':
      'oracle.security.jps.diag.test.JpsContainerAuthenticationLdapConnectivityTest'}
      )
      ```

      The test generates a dump at the specified location. The dump includes the following lines:

      ```
      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) …
      ```

   c. Inspect the dump for errors and failures. In this example, the text in bold indicates that the connection to the LDAP provider could not be established.

Diagnosing Security Errors

In addition to the OPSS diagnostic framework, use loggers to diagnose and solve a variety of security issues as described in the following sections:

- About OPSS Loggers
- Loggers by Service
- System Properties
- Understanding Log Entries

> **See also:**
>
> The OPSS Diagnostic Framework

About OPSS Loggers

The following sections describe the log files and loggers that OPSS supports and explains how to configure, set logger levels, and view log files with Fusion Middleware Control and WebLogic Scripting Tool (WLST):

- About Diagnostic Log Files
- Offline WLST Loggers

About Diagnostic Log Files

Each server instance in a domain writes all OPSS exceptions raised by 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.

Servers write security-related errors to diagnostic files. Server-related security errors, such as exceptions raised by issues with a subject or principal, or errors that may occur while migrating or reassociating domain security data, are written in the Administration Server diagnostic log. Application-related security errors, such as exceptions raised by application-specific policies or credentials, are written in the diagnostic log of the Managed Server where the application is running.

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.

The domain logs duplicate some of the messages in server logs, and they help determine the server on which a fault has occurred in domains with a large number of servers.

The generation of a new log file is determined by file size: when a log file exceeds a specified size, the system generates a new one with a name whose integer suffix is increased by 1.

See also:

Server Log Files and Domain Log Files in *Configuring Log Files and Filtering Log Messages for Oracle WebLogic Server*.

What Is the WebLogic Diagnostics Framework? in *Configuring and Using the Diagnostics Framework for Oracle WebLogic Server*.

Adding WebLogic Logging Services to Applications Deployed on Oracle WebLogic Server.

Managing Log Files and Diagnostic Data in *Administering Oracle Fusion Middleware*.

Offline WLST Loggers

Logging for online WLST commands is automatic, but it is not for offline commands. When you use an offline command such as the `migrateSecurityStore` command, enable logging by starting the Java Virtual Machine (JVM) with the following system properties:

- `wlst.offline.log`, with one of the following values: `<filename>`, `stdout`, `sterr`, or `disable`. If unspecified, then the log files are located in the `$MW_HOME/logs` directory.
- `wlst.offline.log.priority`, with one of the following values: `OFF`, `SEVERE`, `WARNING`, `INFO`, `CONFIG`, `FINE`, `FINER`, `FINEST`, `ALL`, `debug`, `info`, `warn`, `error`, `fatal`.

Loggers by Service

The following sections list the loggers available per service:

- Logging Authorization
- Logging Audit
- Logging the User and Role API
- Logging Other Components

- Logging Authorization
- Logging Audit
- Logging the User and Role API
- Logging Other Components

Logging Authorization

You can enable and disable loggers without having to stop and restart servers. Typically, you set the level of loggers to `TRACE:32`.

OPSS provides several loggers that help troubleshooting runtime authorization failures:

- oracle.security.jps.util.JpsAuth - Logs the start and return of `checkPermission`.

- oracle.security.jps.trace.logger - Logs information about application roles, permissions, targets, principals, and granted and denied policies. This logger generates a large output. Use it to debug a single use case only.
- oracle.jps.authorization - Logs messages during runtime authorization.
- oracle.jps.common - Logs messages in OPSS common functional areas, such as service management and user management.
- oracle.security.jps.dbg.logger - Logs messages that are used to debug the `JpsFilter` filter permission checks.
- oracle.security.jps.az.internal.runtime.policy.AbstractPolicyImpl - Logs messages for the `oracle.security.jps.az.internal.runtime.policy.AbstractPolicyImpl` class.
- oracle.security.jps.internal.policystore.JavaPolicyProvider - Logs messages for the `oracle.security.jps.internal.policystore.JavaPolicyProvider` class.

Logging Audit

This section explains how to interpret audit log messages and how to use them to diagnose component failures. Log files are located at:

DomainName`/servers/$SERVER_NAME/logs/$SERVER_NAME-diagnostic.log`

Table J-1 lists the available diagnostic log files.

Table J-1 Log Files for Audit Diagnostics

Component	Log Location	Configuring Loggers
Java EE Components	DomainName/ servers/$SERVER_NAME/ logs/$SERVER_NAME- diagnostic.log	oracle.security.audit.logger (See instructions below)
OPMN Components	See Logging Audit.	See Logging Audit.
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	Oracle Fusion Middleware Repository Creation Utility log location (Default is $ORACLE_HOME/rcu/log/)RCU_LOG_LOCATION can be set to change this location	Repository Creation Utility log level. Default is ERROR.

Configuring Audit Loggers

Use Fusion Middleware Control to configure the `oracle.security.audit.logger` logger, and to view loggers. For information about log files, see Managing Log Files and Diagnostic Data in *Administering Oracle Fusion Middleware*.

Audit error messages are numbered `IAU-XXX`.

To configure audit logging for system components managed with OPMN, edit the `auditconfig.xml` file to specify the log directory location to which the component's audit logs should be written:

```
<LogsDirectory>
   <MaxFileSize></MaxFileSize>
   <Location>/tmp/audit/auditlogs</Location>
</LogsDirectory>
```

Logging the User and Role API

To trace OPSS User and Role API calls, set the `oracle.idm.userroleapi` logger to TRACE:32.

Logging Other Components

Additionally, OPSS provides the following loggers:

`oracle.jps.deployment` logs issues with OPSS artifacts packed with the application when you deploy the application.

`oracle.jps.openaz` logs issues with PEP API calls. Setting `oracle.jps.openaz.level` to `FINEST`, logs information about submitted requests - identity, resource, action, context - and authorization results.

`oracle.jps.attribute` logs issues with the OPSS Attribute service.

System Properties

Use the following properties to configure debugging at server startup:

- `jps.auth.debug`, logs permission checks that fail only. To disable permission check messages, set this property to `false`. By default, it is `true`.
- `jps.auth.debug.verbose`, logs permission checks that fail. Logs more information than jps.auth.debub. By default, it is `false`.
- `DebugOPSSPolicyLoading`, a flag that monitors the progress and setting of the policy provider.
- `java.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.

Edit the `setDomainEnv.sh` script and add the desired property to the `EXTRA_JAVA_PROPERTIES` system property. This change requires that you restart the server.

> **Caution:**
>
> Setting a high logging output may cause stuck threads, especially when file loading takes place. To avoid this situation, change the timeout value that WebLogic Server uses to mark a thread as stuck to a higher value.

Other System Properties

Additional system properties that may further help you debugging are the following:

- `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, then it logs all application role names.
- `oracle.security.jps.log.for.permeffect`, logs a grant that was granted or denied. If the value is unspecified, then it logs all grants (regardless whether they were granted or denied).
- `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, then it logs all permission class names.
- `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, then it logs all permission targets.
- `oracle.security.jps.log.for.enterprise.principalname`, logs the name of the principal (enterprise user or role) that matches exactly a specified name. If the name to match is unspecified, then it logs all principal names.

Examples of Use

The following examples illustrate typical settings of system properties.

- To log all application role names that contain the `myAppRole` string:

  ```
  -Doracle.security.jps.log.for.approle.substring=myAppRole
  ```

- To log all denied permission checks:

  ```
  -Doracle.security.jps.log.for.permeffect=deny
  ```

- To log all granted permission checks:

  ```
  -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 `java.util.PropertyPermission`:

  ```
  -Doracle.security.jps.log.for.permclassname=java.util.PropertyPermission
  ```

- To log all target names that contain the `p.mon` string:

  ```
  -Doracle.security.jps.log.for.permtarget.substring=p.mon
  ```

- To log all authorizations involving the `manager` principal name:

  ```
  -Doracle.security.jps.log.for.enterprise.principalname=manager
  ```

- To log application role names that match a substring or principal names that match a string, use both `oracle.security.jps.log.for.approle.substring` and `oracle.security.jps.log.for.enterprise.principalname`.

Understanding Log Entries

Your understanding of log errors is crucial to isolate and solve an error. This section explains how to interpret the contents of a diagnostic log file, such as the following:

```
[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:

- [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 *Error Messages*.

- [oracle.jps.admin]

  Identifies the logger category. The subcategories of `oracle.jps` (such as `admin`) 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 (Java SE applications only)
  - `idmgmt` - identity store errors
  - credstore - credential store errors
  - authorization - policy store errors at runtime
  - 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. Correlates and traces sequence of events. The execution context ID (ECID) provides information about the flow across processes, such as, from a request, to 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.

Troubleshooting Reassociation and Migration

The following sections describe issues with data security operations:

- Reassociation Failure
- Unsupported Schema
- Missing Policies in Reassociated Security Store
- Migration Failure

Reassociation Failure

This section explains three reasons why reassociating from a file to an LDAP store may fail.

Symptom 1- Error Code 32

Reassociation fails and an error like the following is logged in the `MyServerName.diagnostic.log` file:

```
[LDAP: error code 32 - No Such Object]
Authentication to LDAP server ldap://myServer.com:3060 is unsuccessful.
```

Diagnosis 1

This error means that the specified node does not exist in the LDAP server.

It is required that the root node you specified exist in the LDAP *before* you start reassociating the store.

Solution 1

Verify that the data you enter in the **JPS Root DN** text field matches the name of a node in the target LDAP directory, and then rerun the reassociation.

Symptom 2- Error Code 68

Reassociation fails and an error like the following is logged in the *serverName*`.diagnostic.log` file:

```
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

This error indicates that the name specified in the **WebLogic Domain Name** text field is a grandchild of the **JPS Root DN** node in the target LDAP directory.

It is required that the `cn` not be a descendant of the root node.

Solution 2

Verify that the name you enter in the **WebLogic Domain Name** text field 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 *serverName*`.diagnostic.log` file:

```
[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

This error points to some problem with the `test-role` application role.

Ensure that when you enter 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 LDAP server. This test catches any problems with those values before reassociation begins.

Solution 3

In our example, a quick inspection of `system-jazn-data.xml` reveals that the `test-role` role is used by a policy, but it is not defined. The following example illustrates 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 stores with the fixed file, and (c) rerun the reassociation.

Symptom 4 - Audit Store is Not Reassociated

This note applies to release 11.1.1.6.0. Reassociation with Oracle Enterprise Manager Fusion Middleware Control (Fusion Middleware Control) completes successfully, but you do not see the audit store in the target store. This is because in that release the audit store could be reassociated only with the `reassociateSecurityStore` WLST command.

Solution 4

In the original environment, run the `reassociateSecurityStore` command with a different `jpsroot` node. This effects an LDAP-to-LDAP directory reassociation and data (including audit data) gets migrated to the new node.

Unsupported Schema

This section explains a reason why reassociation to an LDAP server may fail.

Symptom

Reassociating the security store to an LDAP repository fails and the Administration Server 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 indicates that the schema of the target LDAP repository is not supported.

Solution

Update the target LDAP repository to one supported and then try reassociating again. The version of Oracle Internet Directory must be 10.1.4.3 or later. For a list of supported versions, see Using an LDAP Security Store.

Missing Policies in Reassociated Security Store

Symptom

You successfully reassociated a file store to an LDAP store, but codesource policies are missing in the target store.

Diagnosis

At runtime, the server reports a stack trace like 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.checkPolicyStoreAccessPerm
ission(AbstractPolicyStore.java:246)
        at
oracle.security.jps.internal.policystore.ldap.LdapPolicyStore.getApplicationPolicy(Ldap
PolicyStore.java:281)
```

```
        at
oracle.security.jps.internal.policystore.PolicyUtil.getGrantedAppRoles(PolicyUtil
.java:898)
        at
oracle.security.jps.internal.policystore.PolicyUtil.getJpsAppRoles(PolicyUtil.jav
a:1354)
        at
oracle.security.jps.wls.JpsWlsSubjectResolver$1.run(JpsWlsSubjectResolver.java:27
3)
        at
oracle.security.jps.wls.JpsWlsSubjectResolver$1.run(JpsWlsSubjectResolver.java:27
0)
        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 codesource, and the authorization is not allowed because it evaluates to `false`.

Solution

Check the Administration Server logs for messages like the following, which suggests that the schema was never seeded during the reassociation:

```
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
```

To ensure that the schema is seeded during reassociation:

1. Remove the `cn=OPSS` container under the `cn=OracleSchemaVersion` container in LDAP.
2. Start with a clean working instance of a file policy store.
3. Reassociate the file store to a target LDAP store.

Check the Administration Server logs to confirm that the OPSS schema was seeded in the LDAP server by looking for messages like the following:

```
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 reassociating to a Release 11*g* Oracle Internet Directory server, then the schema version should read: 11.1.1.1.0

If reassociating to a Release 10.1.4.3 Oracle Internet Directory server, then the schema version should read: 11.1.1.0.0

The policy store version is set in LDAP under:

```
cn=PolicyStore,cn=OPSS,cn=OracleSchemaVersion
```

The credential store version is set in LDAP under:

```
cn=CredentialStore,cn=OPSS,cn=OracleSchemaVersion
```

Migration Failure

This section describes a reason why policy migration fails when you deploy the application. Note that an application deployment may succeed even though the migration has failed. For information about version issues, see also Incompatible Versions of Security Stores.

Symptom

You configured your application to migrate policies at deployment. The application deployment succeeds, but the server diagnostic file has 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)...
```

In this example, the `JpsJdev` application was deployed to the `server_soa` Managed Server. The key phrase to look for to locate such error is highlighted in the output example. This error also describes the artifact that raised the exception, the `test_role` application role.

Diagnosis

A look at `jazn-data.xml` reveals that the `test_role` role is referenced in a grantee:

```
<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 is 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 you have defined all application roles referenced in application policies. If a referenced role name cannot be matched, then the migration fails.

Troubleshooting Server Startup

This section explains several reasons why Oracle WebLogic Server may fail to start in the following sections:

- Missing Required LDAP Authentication Provider
- Missing Administrator Account
- Missing Permission
- Server Fails to Start
- Other Server Start Issues
- Permission Failure Before Server Starts

Missing Required LDAP Authentication Provider

This section explains a reason why WebLogic Server may fail to start after modifying the list of authentication providers in a domain.

Symptom

After modifying the list of providers in a domain, WebLogic Server fails to start, and the error messages include the following:

```
java.lang.IllegalArgumentException: null KeyStore name
```

Diagnosis

One cause of this problem is that the list in your domain does not include an LDAP provider. An LDAP provider is *required* in this list in any domain using OPSS.

Solution

Add a provider:

1. Open `DOMAIN_NAME/config/config.xml`.
2. Edit `to` include within the element `<realm>` an LDAP provider:

```
<realm>
 ...
 <sec:authentication-provider xsi:type="wls:default-authenticatorType">
 </sec:authentication-provider>
 ...
</realm>
```

3. Restart the server.

After the server is up and running, modify the list of providers to include the provider of your choice with Oracle WebLogic Server Administration Console, and ensure that at least one of them is an LDAP authentication provider.

Use WebLogic Server Administration Console to:

1. Go to the page **Create a new Authentication Provider**.
2. Enter a name and choose a type:
   - ActiveDirectoryAuthenticator
   - WebLogic Default Authenticator (this is the one inserted manually in the example)
   - LDAPAuthenticator
   - LDAPX509IdentityAsserter
   - OpenLDAPAuthenticator
   - OracleInternetDirectoryAuthenticator
   - OracleVirtualDirectoryAuthenticator

Missing Administrator Account

This section explains a reason why WebLogic Server may fail to start.

Symptom

After removing the configured authentication provider and adding the Oracle Internet Directory authentication provider, the server fails to start.

Diagnosis

Most likely, you have forgotten to enter an account member of the Administrators group in the provider you added. The server requires that such an account be present in one authentication provider. This account is always present in the default one.

Solution

Add the deleted LDAP provider manually:

1. Open `DOMAIN_NAME/config/config.xml`.
2. Edit `to` include within the `<realm>` element the default provider:

```
<realm>
 ...
 <sec:authentication-provider xsi:type="wls:default-authenticatorType">
 </sec:authentication-provider>
 ...
</realm>
```

3. Restart the server.

After the server is up and running:

1. Create an account for a member of the Administrators group.
2. Set the flag to SUFFICIENT.
3. Restart the server, which it should start without problems, because it is using the account in the Administrators group provided in the default provider.
4. Reset the flag to REQUIRED and remove the default provider. The server should now start using the account in the Administrators group that you created.

Missing Permission

This section explains a reason why WebLogic Server may fail to start.

Symptom

The server fails to start when you try to start it with the security manager enabled (with the `-Djava.security.manager` system property).

Diagnosis

The grant that allow access to public key methods in `oraclepki.jar` when the security manager is enabled at server startup is not found.

Solution

Ensure that a grant like the following is present in the `weblogic.policy` file, or add it if it is not present:

```
grant codeBase "file:${oracle.home}/modules/oracle.pki_${opss.version}/*" {
 permission java.security.AllPermission;
};
```

This grant is provided by default. Note that when security manager is enabled, the access to all system resources requires codesource permission grants.

See Using the Java Security Manager to Protect WebLogic Resources in *Developing Applications with the WebLogic Security Service*.

Server Fails to Start

This section explains two reasons why WebLogic Server fails to start.

Symptom 1

The `${domain.home}/config/fmwconfig` domain directory is on an NFS-mounted partition, and an error message like the following is logged when you start the server:

```
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 you start the server again, the following message is logged:

```
java.io.IOException: No locks available.
```

Diagnosis 1

Because OPSS requires file locking to manage security data in file stores, the `No locks available` error message 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 credential operation run into an exception. Furthermore, when `orapki` debugging is turned on and you start the server again, a file permission error is logged.

Diagnosis 2

Credential operations create and make use of temporary files in the `/tmp` directory. 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*` not owned by the user starting the server, and restart the server.

Symptom 3

This symptom is same as symptom 2, but it has a different resolution. The server fails to start because a credential operation run into an exception. Furthermore, when `orapki` debugging is turned on and you start the server 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 fails 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.

Other Server Start Issues

This section explains several reasons why WebLogic Server may fail to start.

Symptom

When attempting to load and set the policy provider, WebLogic Server fails to start and logs an exception similar to the following:

```
<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 `jps-config.xml` configuration file. If this task is not completed successfully, then WebLogic Server fails to start. 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 server startup failure, check the server's log file and inspect the logged stack trace. For information 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 codesource 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

The following steps describe a solution to each of these reasons:

1. Ensure that the correct path is specified by the `oracle.security.jps.config` system parameter:

   ```
   -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 with the full path specified 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. The following examples illustrate incorrect and corrected codesource URLs:

```
Example 1 - Incorrect URL
<grantee>
  <codesource>
     <url>${my.oracle.home}/jlib/webcacheua.jar<</url>
   </codesource>
</grantee>

Example 1 - Corrected URL (in bold)
<grantee>
  <codesource>
     <url>file:/${my.oracle.home}/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

Example 2 - Incorrect URL
<grantee>
  <codesource>
     <url>c:/myfolder/jlib/webcacheua.jar</url>
   </codesource>
</grantee>

Example 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 information about the syntax of URL specifications in a codesource, see <url>.

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 started up, but the server diagnostic file contains 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 service required to check that permission was not fully initialized at the time of the request.

Solution

To workaround this issue:

1. Edit the `weblogic.policy` file and add the appropriate grant(s).
2. Start WebLogic Server with the following system properties:
   - `java.security.policy` set to the location of the `weblogic.policy` file.
   - `jps.policystore.hybrid.mode` set to `true`.

Troubleshooting Permissions

This section describes the following issues:

- Troubleshooting System Policy Failures
- Failure to Get Permissions - Case Mismatch
- Authorization Check Failure
- User Gets Unexpected Permissions
- Granting Permissions in Java SE Applications
- Application Policies Not Seen in 12c High Availability (HA) Domain

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 must perform a management operation on a policy, a credential, a key, or audit.A runtime authorization failure on any of these operations throws the `java.security.AccessControlException` exception. This section describes the process that you follow to diagnose the issue.

Symptom

A runtime authorization failure occurs:

```
java.security.AccessControlException: access denied
(oracle.security.jps.service.credstore.CredentialAccessPermission
```

```
context=SYSTEM,mapName=oracle.patching,keyName=FUSION_APPS_PATCH_WLS_ADMIN-KEY read)
```

The first step is to get information about the failure by 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 output, look for the string “Failed ProtectionDomain." The following output example 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

This failure indicates a mismatch between the provisioned codesource grant and the runtime 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 runtime evaluates to. To this end:

1. Inspect the provisioned codesource grant with either Fusion Middleware Control as explained in Managing Policies with Fusion Middleware Control or the `listCodeSourcePermissions` WLST command. See listCodeSourcePermissions in *WLST Command Reference for Infrastructure Security*.
2. If the provisioned codesource grant does not match the data in the runtime error, then modify the grant with Fusion Middleware Control or the `grantPermission` WLST command. See grantPermission in *WLST Command Reference for Infrastructure Security*.

3. Ensure that domain and other variables expand to correctly specified the grant. Note that, in loggers, codesources are written using the absolute path to the JAR file, but URLs are written using environment variables.
4. If you specified the grant in the `system-jazn-data.xml` file, then review the grant in that file so that the fix will take effect when you deploy the application to a new environment.
5. If your code is required to run in a privileged block, then make sure that it is doing so.

Failure to Get Permissions - Case Mismatch

This section explains some reasons why an enterprise user or role (group) fails to get permissions.

Symptom

An enterprise user or group, properly entered in an authentication provider, is not granted the permissions defined by a grant.

Diagnosis

This problem is likely to occur when there is a case mismatch between the stored name and the supplied name. For example, this mismatch would occur when the stored user name is *jDoe* and the supplied user name is *jdoe*.

Solution 1

The first solution involves setting the appropriate property in the authentication provider 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 an authentication provider, even when the corresponding characters differ in case.

To set your provider property:

1. Use WebLogic Server Administration Console to go to the page where your provider is configured. For example, if you are using the default provider, then go to the Default Authenticator page at **Realms.myrealm.Providers.Default Authenticator.**
2. Select the tab **Provider Specific**.
3. Set the property **userRetrievedUserNameAsPrincipal** to true.
4. Restart the server.

Solution 2

The second solution considers the case where one must produce a principal from a user name.

When you create a user or role principal, use:

```
Principal userPrincipal = new WLSUserImpl(user.getUserProfile()getName());
Principal rolePrincipal = new WLSGroupImpl(role.getRoleProfile().getName());
```

instead of:

```
Principal userPrincipal = new WLSUserImpl(user.getName());
Principal rolePrincipal = new WLSGroupImpl(role.getName());
```

To obtain the correct user or group name, either pass the name *exactly* as it is stored in the authentication provider or use the sequence of calls:

```
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);
```

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 this an authorization failure is a mismatch between the context and the stripe that you application is using.

On the one hand, the application stripe that an application uses is specified in the `web.xml` file with the `application.name` parameter in the configuration of the `JpsFilter` filter or the `JpsInterceptor` interceptor. If the application stripe is unspecified, then the system picks an application stripe based on the application name.

On the other hand, the runtime policies that your application uses are specified in the `system-jazn-data.xml` file with the `<application.name>` element.

If those two names do not match or you have not explicitly specified the stripe to use, then, most likely, your application is accessing the wrong policy stripe and is not able to authorized your application's 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 match. However, in cases where several applications share the same policy stripe, they may differ.

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 security store has not been properly updated before you remove users or change user data.

Solution

Before deleting a user, revoke all the permissions, application roles, and enterprise groups that had been granted to that user. If you fail to remove all security data referencing the user, they are left dangling and, potentially, inherited if another user with the same name is created at a later time.

Similar considerations apply to when a user name 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.

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 lines like the following:

```
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 little inspection indicates that the security store includes the following attribute:

```
orcljaznjavaclass=oracle.security.jps.internal.policystore.UnresolvedPrincipal+cn=enterpriseRoleName2
```

Solution

The code should be replaced by the following lines:

```
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 `PrincipalEntry` array instead of the `Principal` array and the `PermissionEntry` array instead of the `Permission` array.

Application Policies Not Seen in 12c High Availability (HA) Domain

This section describes a sequence that results in application policies not taking effect in the 12c high availability (HA) 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 domain, either to the Administration Server or to a Managed Server (but not to both).
2. Undeploy the application.
3. Redeploy the application to a server *different* from the server in step 1.

Step 3 results in an exception and the application policies do not take effect. This issue is observed in 12c HA domains only.

Example J-1 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 server A, the Administration Server, 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).

Example J-2 Solution

Undeploy the application and, before deploy it, restart all servers in the HA domain. This way, the modified procedure leads to synchronized caches in all servers in the HA domain, and the application policies are seen as expected.

Troubleshooting Connections and Access

This section describes the following issues:

Database Connection Exception

This section explains why a database connectivity exception is thrown for the OPSS security store.

Symptom

OPSS throws the `oracle.security.jps.service.policystore.PolicyStoreConnectivityException` when it fails to connect to the database to read or update security store data.

Diagnosis

Check log files to see if there is `oracle.security.jps.service.policystore.PolicyStoreConnectivityException` printed.

Solution

Check the database status and connectivity from the host where the exception above was thrown.

Other Database Exceptions

This section explains how to identify and solve other database exceptions for OPSS database security store access.

Symptom

OPSS throws the `oracle.security.jps.service.policystore.PolicyStoreException` with `org.eclipse.persistence.exceptions.DatabaseException` and `java.sql.SQLException` as the cause of the exception.

Diagnosis

Get error code from `java.sql.SQLException` such as `java.sql.SQLException: ORA-28000: the account is locked`.

Solution

Contact your database Administrator to solve the SQL exception.

JNDI Connection Exception

This section explains why Java Naming and Directory Interface (JNDI) connections may throw a time out exception.

Symptom

JNDI Connections throw the `javax.naming.NamingException: LDAP response read timed out, timeout used:-1ms` exception.

Diagnosis

This issue is found in domains configured to use an Oracle Identity Directory security store, or when using the User Role API or IGF/IDS against an LDAP identity store on any of the following JDK versions: Java SE 6u85, 7u72, or 8u20.

Solution

Update the JDK to a version supported in this release. For certified JDK versions, see Oracle Fusion Middleware Supported System Configurations.

Failure to Connect to the Embedded LDAP Server

This section explains why a connection to the embedded LDAP server fails.

Symptom

The connections that client applications use to request queries to the embedded LDAP server with the User and Role API, are stored and maintained in a connection pool. By default this pool is the JNDI pool as specified in the `jps-config.xml` file.

If the number of current connections in the pool exceeds the maximum allowed by the LDAP service, then 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, you must adjust the maximum number of concurrent connections allowed by the LDAP service as appropriate to the application's needs. This threshold must be finely tuned up: a too small maximum may not be sufficient (and cause an exception). 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 provider:
  - If the provider that your application uses is the embedded LDAP server, then edit the file *DomainName*/servers/*MyServerName*/data/ldap/conf/vde.prop, and increase the value of the `vde.quota.max.conpersubject` property from the default 100 to, for example, 200, or any other value.
  - Otherwise, if your application is using any other provider, then consult the provider's documentation to learn how to modify the maximum.
- Edit the file *DomainName*/config/fmwconfig/jps-config.xml and remove the `CONNECTION_POOL_CLASS` property from the provider server instance (by default, this property has the value `oracle.security.idm.providers.stdldap.JNDIPool`.

Note that these settings do not exclude each other and, in any case, you must restart the server for the changes to take effect.

Failure to Connect to LDAP Server

This section explains the likely reasons why a connection to an LDAP server may fail. This failure can also happen during reassociation.

Symptom

The migration of data from a source repository to a target LDAP server fails.

Diagnosis

This kind of problem is due to incorrect parameter values in the target LDAP server.

For further probing into WebLogic Server log files, search any of the log files in the `DomainName/servers/AdminServer` or `DomainName/servers/ManagedServers` directories for the following strings: <Error>, <Critical>, and <Warning>.

For more information about identifying and solving errors, see Diagnosing Security Errors.

Solution

Verify that all the target server data provided for the migration is valid. If you use Fusion Middleware Control to reassociate to an LDAP server, then ensure that you use the button **Test LDAP Authentication** before initiating the operation. This test catches incorrect parameters.

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 following is logged:

```
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.

Solution

To grant the permission that an application requires to access the credential store, include the appropriate `CredentialAccessPermission` permission in the application's `jazn-data.xml`. This grant takes effect when you deploy or redeploy the application.

To add a permission with Fusion Middleware Control, see Managing Policies with Fusion Middleware Control.

To add a permission with a WLST command, see Managing Policies with WLST.

The following example illustrates how to grant all code in the `myApp` application permission to read all credentials in the `myAlias` folder:

```
<jazn-data>
    <!--  codesource 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>
```

Security Access Control Exception

This section explains a reason why your code may run into a security access control exception.

Symptom

At runtime, your application prints 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

This 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 alternative is to grant permission to all application code in the application Enterprise ARchive (EAR) or web application archive (WAR) files. In this case, the call to the operation can be inserted anywhere in the application code.

The second alternative is to grant permission to just a JAR file. In this case, the call to the 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 example illustrates how to set permission to read any key within the map MY_MAP in the credential store to *any* code within the `BasicAuth` directory:

```
<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>
```

When granting permissions to code in a particular EAR or WAR file, then change the `url` specification to one like the following:

```
<url>file:${domain.home}/servers/_WL_user/jpsBasicAuth/.../BasicAuth.ear</url>
```

When granting permissions to just the code in a particular JAR file, the change the `url` specification 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 `Foo.jar` file that calls a read operation on the credential store must be placed in an `AccessController.doPrivileged` block:

```
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 because our example grant allows a read permission only, none of the set or reset operations work, even inside a privileged block.

Failure to Establish an Anonymous SSL Connection

This section explains the likely reasons why you are not able to establish an anonymous Secure Sockets Layer (SSL) connection while reassociating policies and credentials.

Symptom

Reassociation of a file security store to an LDAP server with Fusion Middleware Control involves testing the anonymous SSL connection to the LDAP server. When you click **Test LDAP**, the test fails.

Diagnosis

Your target LDAP server must be trusted by WebLogic Server and the port number you are using to connect to the LDAP server must be an SSL port.

Solution

Establishing SSL connections to an LDAP server requires additional LDAP server configuration. For information about this configuration, see Prerequisites to Using the LDAP 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 LDAP server has not been configured with such a port, then the connection fails.

Ensure that the supplied LDAP server port is an SSL port configured to listen in anonymous SSL mode, and that the supplied server name reachable.

Oracle Business Intelligence Publisher Time Zone

You may see some problems with audit reports if BI Publisher and the database are installed in sites with different time zones. Ensure that BI Publisher and the database are installed in the same time zone.

Troubleshooting Searching

The following sections describe query issues:

- Search Failure When Matching Attribute in Security Store
- Search Failure with an Unknown Host Exception
- Search Failure When Matching Attribute in Security Store
- Search Failure with an Unknown Host Exception

Search Failure When Matching Attribute in Security Store

This section describes a reason why you must catalog attributes.

Symptom

When querying the security store, you run into the following exception:

```
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

You must catalog `orcljpsresourcetypename` before you use it in a filter to search the security store.

Solution

An LDAP 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 `reassociateSecurityStore` WLST command.

To catalog attributes manually use the `ldapmodify` command:

```
>ldapmodify -h <host> -p <port> -D <bind DN> -w <bind password> -v -f <catalogue
modify ldif file name>
```

To catalog the `createtimestamp` and `modifytimestamp` attributes, for example, use an LDAP Data Interchange format (LDIF) file like the following:

```
dn: cn=catalogs
changetype: modify
add: orclindexedattribute
orclindexedattribute: modifytimestamp
orclindexedattribute: createtimestamp
```

The list of LDAP 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
```

Search Failure with an Unknown Host Exception

When searching Microsoft Active Directory (configured for LDAP 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 with Microsoft Active Directory when the browser runs on a system different from Microsoft Windows, or the browser runs on Microsoft Windows but not in the server where Microsoft Active Directory is running.

Diagnosis

This problem can arise due to LDAP referral chasing. An LDAP 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 target so that the client can conduct a DNS search for another domain controller. If the client is configured to chase referrals, then the search can continue.

For the scenario where the user has a Windows-based computer, an issue can occur with LDAP referrals if trust is not established between the client's domain controller and the Active Directory domain controller.

Solution

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 at:

```
C:\WINDOWS\system32\drivers\etc\host
```

On a Unix-based system, add this entry to the `/etc/hosts` file, using the format:

```
ADhost_IPaddress ADhost_name
```

where *AD_host_name* is the host name specified in the referral, such as `123.123.123.123 my2003ad.com`.

Troubleshooting Versions

The following sections describe the version issues:

Incompatible Versions of Binaries and Security Store

This section describes the reason why the server would throw the `PolicyStoreIncompatibleVersionException` exception.

Symptom

You encounter an error like the following:

```
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 exception indicates that the domain OPSS binaries version (11.1.1.4.0) and the security store version (11.1.1.5.0) used by that domain are incompatible. The version of the security store is established during reassociation and that version is used until you upgrade the security store to a newer version.

OPSS domain binary versions are backward compatible with security store versions, but they are not forward compatible. So, the error indicates that the security store has version newer that the version of the OPSS binaries.

Here are three scenarios where OPSS binaries ends up being incompatible:

- Scenario 1
  - Domain1 and Domain2 point to the same security store. Domain1, Domain2, and that security store have all version 11.1.1.3.0.
  - The binaries in Domain1 are upgraded to 11.1.1.4.0.
  - The security store is upgraded to 11.1.1.4.0 (using the `upgradeOPSS` command).
  - When the Domain2 is brought up again, the binary and security store versions are incompatible.
- Scenario 2
  - Domain1 points to a security store, and both binaries and the security store have version 11.1.1.3.0.

  - An attempt to migrate the security store to a 11.1.1.4.0 fails because the migration would render a scenario with incompatible versions.

    Migration is supported only when the OPSS binaries and the security store versions are same.

- Scenario 3
  - A 11.1.1.3.0 Domain1 attempts to join a 11.1.1.4.0 security store (in some other domain), using the `reassociateSecurityStore` WLST command with the `join` argument.
  - The operation fails because the sharing would render a scenario with incompatible versions.

    Reassociation is supported only when the OPSS binaries and the security store versions are same.

Solution

The solution to all three scenarios is either one of the following:

- Update OPSS binaries to match the version of the security store the domain is using.
- Reassociate the security store to a security store that has version not newer than the version of OPSS binaries.

Incompatible Versions of Security Stores

This section describes the reason why you may encounter the `PolicyStoreIncompatibleVersionException` exception when migrating the security store.

The `PolicyStoreIncompatibleVersionException` exception indicates that the version of the source store is *higher* than the version of the target store. Migration carries on 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.

Troubleshooting Other Errors

The following sections describe a variety of issues:

- Runtime Permission Check Failure
- Tablespace Needs Resizing
- Oracle Internet Directory Exception
- User and Role API Failure

Runtime Permission Check Failure

This section explains a reason why a permission may fail to pass a permission check.

Symptom

Your application prints an error like the following one:

```
[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@1
4061a8
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 applications loading the same class thereafter fail the permission check and print a similar error.

Note that even though the permission class is in the permission collection, the check fails and the access is denied, because the environment contains several instances of a permission class with the same name.

Solution

Ensure that if two or more applications running in the same domain share a permission class, then you include that class in the system class path.

Tablespace Needs Resizing

This section describes a reason why SQL operations fail in a DB security store.

Symptom

While performing policy operations, the database issues a the following error:

```
"ORA-1652: unable to extend temp segment by 128 in tablespace"
```

Diagnosis

The reason is that the temporary table used by the operations has no more free blocks available.

Solution

To solve this issue:

1. Extend the size of the temporary table to 4096 M:

   ```
   ALTER TABLESPACE "<TEMP_TABLESPACE_NAME>" ADD TEMPFILE '<data_file_name>' size
   4096M
   ```

2. Restart the Administration Server.

Oracle Internet Directory Exception

A domain using Oracle Internet Directory 11.1.1.6.0 as the security store runs into the following exception:

```
[LDAP: error code 53 - OID-5018: Cataloging for attr orcljpsextensiontype
is already in progress.]:cn=catalogs
```

You must 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 Using an LDAP Security Store.

User and Role API Failure

This section explains reasons why you may fail to access data in an authentication provider with the User and Role API.

Symptom

The User and Role API fails to access data.

Diagnosis 1

The User and Role API can access data only in the first LDAP provider configured in a domain. At least one such provider must be present in a domain. The API access to that first LDAP provider fails if the target user is not present in that provider, even though that user is present in some other authentication provider.

Solution 1

Enter the missing user in the first LDAP authentication provider or reorder the list of LDAP providers in your domain.

Diagnosis 2

Assume that the target user on which the API that fails is present in the first LDAP provider configured in your domain.

By default, the User and Role API uses the `uid` attribute to perform user search in an LDAP provider. 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 authentication provider have the `uid` attribute set.

If you are developing a Java SE application and you want the User and Role API to use an attribute other than `uid` to search users, say `mail` for example, then you must configure the `username.attr` and `user.login.attr` attributes in the LDAP provider instance of the identity store (in the `jps-config-jse.xml` file):

```
<serviceInstance provider="idstore.ldap.provider" name="idstore.ldap">
   ...
   <property name="username.attr" value="mail"/>
   <property name="user.login.attr" value="mail"/>
   ...
</serviceInstance>
```

For information about adding properties to a provider instance with a prescribed script, see Configuring Services with Scripts.

Characters in Policies

The following sections explain several issues related to characters in policies:

- Special Characters in Oracle Internet Directory 10.1.4.3
- Characters in File Security Stores
- Characters in Application Role Names
- Missing Newline Characters in File Store

- Special Characters in Oracle Internet Directory 10.1.4.3
- Characters in File Security Stores
- Characters in Application Role Names
- Missing Newline Characters in File Store

Special Characters in Oracle Internet Directory 10.1.4.3

When the security store is Oracle Internet Directory 10.1.4.3, 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.

Characters in File Security Stores

The issue explained in this section applies to file stores only.

The following characters:

```
< "  &  $ ? * , / \ ` :    ( ) ^ ' % +  { }
```

are not recommended in role names within file stores.

If you use one of those characters to create a role, then ensure that you escape such characters in arguments to methods like `ApplicationPolicy.searchAppRoles` so they return correct results. For example, if you have an application role named `appRole^$`, then pass the argument as in `appRole\\^\\$` to find the match in the security store.

Alternatively, you could use a wild card in the search expression without including these escaped special characters, as in `appRole*`.

Characters in Application Role Names

An application role name is a string of alphanumeric characters (ASCII or Unicode) and other printable characters (such as underscore or square brackets) that contains no leading or trailing white space. This rule applies to roles names in any type of storage.

Missing Newline Characters in File Store

In a file store, a new-line character is required between the closing of a `<permission>` or `<principal>` tag and the opening of the following one.

The following are examples of strings illustrating incorrect and correct formats.

Incorrect:

```
<permission>
    <class>java.lang.RuntimePermission</class>
    <name>getClassLoader</name>
 </permission><permission>
      <class>java.io.FilePermission</class>
      <name>/foo</name>
      <actions>read,write</actions>
 </permission>
```

Corrected:

```
<permission>
    <class>java.lang.RuntimePermission</class>
    <name>getClassLoader</name>
</permission>
<permission>
      <class>java.io.FilePermission</class>
      <name>/foo</name>
      <actions>read,write</actions>
</permission>
```

Invalid Key Size

This section explains why an invalid key size exception occurs.

Symptom

You encounter the following exception:

```
java.security.InvalidKeyException: Illegal key size
```

Diagnosis 1

During domain creation, OPSS uses the keystore service to create an Advanced Encryption Standard (AES) symmetric key used to encrypt 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 not supported either, then it creates a 128-bit key. If 128-bit keys are not supported, the `invalidKeyException` exception is thrown.

Diagnosis 2

During OPSS startup, 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 `invalidKeyException` exception is thrown.

Solution

Install the Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files in the JDK, and then retry the domain creation or OPSS startup. You can download the JCE files at `http://www.oracle.com/technetwork/java/javase/downloads/jce8-download-2133166.html`

Need Further Help?

To find more information about Oracle Support, visit `http://myoraclesupport.oracle.com`. If you do not find a solution to your problem, log a service request.