BEA AquaLogic Enterprise Security™®

Policy Managers Guide

Version 2.6
Document Revised: April 2007

Contents

1. Introduction

2. Security Policies Overview

3. Writing Policies

4. Advanced Topics

5. Using the Entitlements Management Tool

6. Extending the Entitlements Management Tool

7. Importing and Exporting Policy Data

CHAPTER 1

Introduction

This section describes the contents and organization of this guide—*Policy Managers Guide*. It includes the following topics:

Document Scope and Audience

This document is a resource for system administrators who create and deploy security policies using BEA AquaLogic Enterprise Security™. Typical tasks include writing security policies using the ALES Administration Console, writing security policies outside the console and importing them into ALES, and exporting security policies from ALES and importing them into other ALES installations.

The topics in this document are relevant during the staging, production deployment, and production use phases of a software project. For links to other AquaLogic Enterprise Security documentation and resources, see "Related Documentation" on page 1-2.

It is assumed that readers understand Web technologies and have a general understanding of the Microsoft Windows or UNIX operating system being used. Prior to using this document, you should be familiar with the policy model used by BEA AquaLogic Enterprise Security and described in the *Introduction to BEA AquaLogic Enterprise Security*.

Additionally, BEA AquaLogic Enterprise Security includes many terms and concepts that you need to understand. These terms and concepts, which you will encounter throughout the documentation, are defined in the *Glossary*.

Guide to this Document

This document describes tasks associated with deploying and managing AquaLogic Enterprise Security. It is organized as follows:

- Chapter 2, "Security Policies Overview," describes the different types of policies, describes how to design policies and provides general information about the components of policies: effects, privileges, roles, resources, identities, delegation, and declarations.
- Chapter 3, "Writing Policies," describes how to use the Administration Console to write policies.
- Chapter 4, "Advanced Topics," describes how to write more advanced and complex policies and how to create policy data files.
- Chapter 5, "Using the Entitlements Management Tool," described how to use the Entitlements Management Tool, a user interface based on a hierarchical role-based access control (RBAC) model, to manage roles, users and groups, permissions, and separation of duties constraints.
- Chapter 6, "Extending the Entitlements Management Tool," describes how to extend the Entitlements Management Tool.
- Chapter 7, "Importing and Exporting Policy Data," describes how to import and export policy data to and from the policy database.

Related Documentation

For information about other aspects of AquaLogic Enterprise Security, see the following documents:

- *Introduction to BEA AquaLogic Enterprise Security*—This document provides overview, conceptual, and architectural information for AquaLogic Enterprise Security.
- *Installing the Administration Server*—This document describes installing and configuring the AquaLogic Enterprise Security Administration Application.
- *Installing Security Service Modules*—This document describes installing and configuring Security Service Modules for AquaLogic Enterprise Security.

- *Administration and Deployment Guide*—This document provides an architectural overview of the product and includes step-by-step instructions on how to perform various post-installation administrative tasks.
- *Integrating ALES with Application Environments*—This document describes post-installation integration tasks to configure ALES for use with BEA WebLogic Server, BEA WebLogic Portal, BEA AquaLogic Data Services Platform, BEA AquaLogic Service Bus, Apache Web Server, Microsoft IIS web server and Web Services.
- *Programming Security for Java Applications*—This document describes how to implement security in Java applications. It includes descriptions of the security service Application Programming Interfaces and programming instructions.
- *Programming Security for Web Services*—This document describes how to implement security in web servers. It includes descriptions of the Web Services Application Programming Interfaces.
- *Developing Security Providers for BEA AquaLogic Enterprise Security* —This document provides security vendors and security and application developers with the information needed to develop custom security providers.
- *Javadocs for Java API*—This document provides reference documentation for the Java Application Programming Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Wsdldocs for Web Services API*—This document provides reference documentation for the Web Services Application Programming Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Javadocs for Security Service Provider Interfaces*—This document provides reference documentation for the Security Service Provider Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Javadocs for BLM API*—This document provides reference documentation for the Business Logic Manager (BLM) Application Programming Interfaces that are provided with and supported by this release of BEA AquaLogic Enterprise Security.

Contact Us!

Your feedback on BEA documentation is important to us. Send us e-mail at docsupport@bea.com if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the documentation.

In your e-mail message, please indicate the software name and version you are using, as well as the title and date of your documentation. If you have any questions about this version of BEA AquaLogic Enterprise Security, or if you have problems installing and running BEA AquaLogic Enterprise Security products, contact BEA Customer Support through BEA WebSupport at http://www.bea.com. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using

A description of the problem and the content of pertinent error messages.

Security Policies Overview

This section covers the following topics:

What is an AquaLogic Enterprise Security Policy?

AquaLogic Enterprise Security is a fine-grained entitlements engine that allows the user to centrally define and manage a set of policies to control access for both application software components (for example URLs, EJBs, and EJB methods) as well as the application business objects (for example accounts and patient records) that make up the application. A set of access control policies is evaluated and enforced locally in the application container so application context can be included as part of the access control decision. A major benefit of using AquaLogic Enterprise Security to implement access control is that it allows you to remove security logic from the application. This enables you to take access control decisions out of the hands of your developers and define and manage access control consistently across multiple applications.

Policies are statements that work together to define access control for your business resources. A resource is any object that represents an underlying application or application component that needs to be protected from unauthorized access. A well-written set of policies accurately represents the access control requirements for your business, is easy to manage, and is designed for maximum efficiency.

You write separate policies to grant or deny access privileges to your business resources to users, groups, and roles under some set of conditions, or constraints. Therefore, before you begin to

write policies, you must know the access control requirements of your business, the resources that are to be protected, who the users are and their responsibilities, and what privileges the users are to have on the resources.

There are three types of policies, authorization policies, role mapping policies, and delegation policies, each type having a different function:

- **Authorization policies—**Also referred to as access policies, these define which users, groups, or roles have which privileges on which resources. Authorization policies are used to define the access control for application software components (for example, URLs, JSPs, EJBs, and so on), as well as business objects (such as accounts, customer records, and similar items) in the application.
- **Role mapping policies**—Define what users and groups belong to what roles for what resources. You use role mapping policies to define how, when, and under what constraints roles are assigned to what users and groups.
- **Delegation policies**—Once you have written authorization policies and role mapping policies, you can then write delegation policies. Privileges and roles can be delegated. Typically, delegation policies are used to define the constraints under which a privilege or a role that was previously granted to one user is granted to another user; however, the time period for the delegation can be indefinite. You can use delegation policies to assign access privileges previously granted to one user to another user, group, or role. You can also use delegation policies to assign roles previously granted to one user to another user or group.

Closed-world Security Environment

The policy evaluation strategy imposes a closed-world security environment. This means that before you specifically create an authorization policy granting access privileges to specific resources, users, groups, and roles have no privileges. You must grant privileges with an authorization policy before users can do anything. This means that all privileges to all resources protected by a Security Service Module are implicitly denied until authorization policies grant specific privileges.

Thus, the closed-world security environment has the powerful advantage of having your application security err on the side of caution. That is, if you forget to deploy an authorization policy, someone may be denied access rather than be granted access to something to which they should not have access. A user that is denied privileges will usually let you know that there is a problem (and, if they do not, that is probably okay). On the other hand, a user that has been granted privileges they should not have may not tell you, which may have disastrous

consequences. Once you grant an access privilege, you must explicitly deny it to revoke that right. Explicit `DENY` policies cannot be overruled.

Policy Components

All policies follow a specified sentence-like syntax. The structure of a policy is similar to a sentence and the policy elements can be thought of as parts of the sentence.

The general syntax for an AquaLogic Enterprise Security policy is as follows:

```
Effect (privilege|role, resource, subject, delegator) IF constraint;
```

A single policy can have multiple privileges, resources, and subjects defined.

The functions of each policy component are as follows:

- The **Effect** can be to grant, to deny, or to delegate a privilege or role. A policy can grant, deny, or delegate a privilege or role on a given resource to a subject under some set of constraints. Grant is used to assign a privilege or a role to a subject. Deny is used to deny a privilege or a role from a subject. Delegate is used to assign a privilege or a role that has been granted to one subject to another subject.
- **Privilege|role—**a privilege is an action on a resource. A **role** is a name that can be assigned to a set of users, similar to a group. Authorization policies assign privileges. Role mapping policies assign roles. Delegation policies can delegate privileges or roles.
- A **resource** is a protected object.
- A **subject** can be a user, group, or role. For authorization policies, subjects can be users, groups, or roles. For role mapping policies, subjects can be users or groups.
- A **delegator** is a user, or subject, whose privilege or role is being delegated, or assigned, to another subject. Delegation policies that delegate privileges can delegate the privileges from one user to other users, groups, or roles. Delegation policies that delegate roles can delegate the role to other users or groups. You cannot delegate a role from one user to role.
- **Constraints** are conditions that must be true for the policy to evaluate to true. A broad range of operators and functions can be used to define constraints, but in general, constraints are made up of attribute/value pairs with some comparison operator. Individual constraints can also be combined with logical operators such AND, OR, and NOT. Conditions can include a date, a time, a time period, a day of the week, a day of the month, a day of the year, a location, and other attributes. You may also write custom attributes and use them as conditions. In addition to attributes that you may define specifically for users

and groups and then use them as conditions, you can also define different types of declarations and use them as conditions.

Policies must adhere to the following rules:

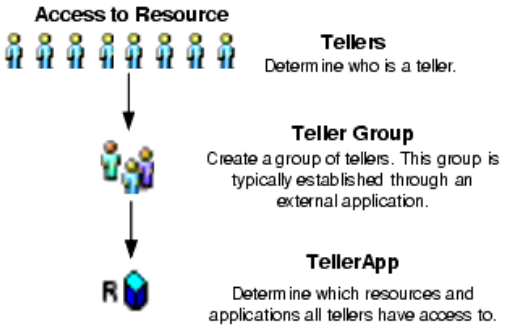
- Parentheses enclose the privilege or role, resource, subject, and delegator, as a group.
- Commas separate the privilege or role, resource, subject, and delegator.
- A delegator cannot be a group or a role.
- The delegator portion of a policy is used only for delegation policies.
- A delegator must be a user, not a group or role.
- For authorization policies, subjects can be users, groups, and roles. However, for role mapping policies, subjects are limited to users and groups. Roles cannot be used as subjects in role mapping policies.
- You may not delegate to a role, only to a user or group.
- The keyword `IF` indicates a constraint.
- All policies end with a semicolon.

Resources

A resource is simply an object used to represent an underlying application or application component (or any resource), that can be protected from unauthorized access using authorization policies. Resources are often hierarchical in nature. Resources can be specific application software components managed by the container (for example, URLs, EJBs, JSPs, and so on) or any business object in the application. Resources may have attributes; for example bank accounts have owners and transfer limits. Resources are hierarchical and child resources inherit policies and attributes from their parent in the hierarchy.

When defining resources you start by defining the top-level resource in the resource tree and then define the resources that make up the tree. Once a resource is defined, you can write authorization policies to grant or deny access privileges to users, groups, or roles for the top-level resource and resources on the tree. Hence, defining a resource tree is a necessary prequisite to writing policies. For example, if you define a resource named `TellerApp` (see Figure 2-1), you can then write an authorization policy that grants or denies access privileges to the `TellerApp` resource.

Figure 2-1 Resource Mapping



Some typical resources that you might want to secure, include:

- an application, an application window, or a dialog box
- specific business transactions, such as a money transfer or security trade
- application controls, such as buttons and menu selections
- database or directory server structures
- Web pages, servlets, and Enterprise Java Beans (EJBs)
- products or services accessed through a BEA WebLogic Portal

Note: In development mode, you may use the Resource Discovery tool to help define resources for a particular application. For more information, see "Resource Discovery" on page 4-84.

For more information about resources, see the following topics:

- "Virtual Resources" on page 2-6
- "Resource Attributes" on page 2-6
- "Privilege Groups" on page 2-6
- "Privileges" on page 2-6

Virtual Resources

In addition to the resources that you define in the resource tree, you have the option of defining virtual resources, which do not appear in the resource tree. This feature offers some flexibility as to the levels of resource hierarchy that must be included in the resource tree so that protections can be assigned. Once you configure a resource to allow virtual resources, any resources below it, that is, its child resources, are, in effect, virtual resources and are protected by the same policies as their parent, even though they do not appear in the resource tree. If you configure a resource to allow virtual resources, you enable policies to be evaluated on resources below the resource which are not explicitly defined inside in the resource tree. For example, if a directory is configured in the resource tree to allow virtual resources, the Authorization and Role Mapping Engine (ARME) can process access requests on files in of the directory even if the files are not explicitly listed in the resource tree.

Resource Attributes

You can associate attributes with resources. An attribute contains information about a characteristic of the resource to which it is associated. Thus, you can use attributes to define additional information about a resource. For example, `filetype` could be a resource attribute that you use to define an html, image, jsp, or pdf file type. Then, you could grant access to all pdf files in a directory by adding the condition: `if filetype = pdf`.

Privilege Groups

A privilege group is constructed by grouping two or more privileges. A privilege can belong to more than one privilege group. In addition to the privilege groups that are provided in the product, you can define your own with distinct characteristics. Privilege groups are not used in policies. They are simply a way to organize privileges and have no meaning when writing a policy and are only provided to simplify the task of choosing the right privilege.

It is common to define a privilege group that applies to a particular application or set of transactions. You can control access to privilege groups (those provided in the product and those that are user-defined) through delegated administration.

Privileges

A privilege represents an action or task in your business policy that can be executed on a resource. Privileges in a policy specify the actions that are granted or denied on a resource. Privileges can be standard actions associated with specific software components (for example `Get` and `Post` for a URL) or a custom action for a business object in an application (for example, transfer for a bank

account). The privileges that may be granted or denied on a particular resource are limited by the operations supported by the resource. For example, a simple text file may support Read, Write, Copy, Edit, and Delete operations. Similarly an executable (`.exe`) may support operations such as Copy, Delete, and Execute. A more complex resource may support far more complex privileges. For example, a checking account application may support operations such as deposit, withdrawal, view account balance, view account history, transfer to savings, and transfer from savings.

In addition to the privileges that are provided in the product, you can define your own privileges. You can also organize privileges into logical groups for ease of management.

You use privileges to write authorization polices as follows:

```
grant(privilege, resource, subject[users, groups, roles]) IF constraint;
```

For example, if you have the business security requirement: "*Only lead tellers can open an account*," you might define an `OpenAccount` privilege and a `LeadTellers` role. Now, to grant `LeadTellers` (the role) the authority to open an account (the privilege), the authorization policy might look like this:

```
grant(//priv/OpenAccount, //app/policy/TellerApp, //role/LeadTellers)
if time24 in [900..1700] AND
dayofweek in [Monday..Friday];
```

When this policy is deployed, only tellers who are assigned the `LeadTellers` role are allowed to used the `TellerApp` to open an account and they may do so only between the hours of 9:00 AM and 5:00 PM (a time-of-day constraint) on Monday through Friday (a days-of-the-week constraint).

Identities

Identity definition includes the definition of directories, users, groups, and roles. An identity directory serves as a logical container for a collection of identity attributes, users, groups, and roles. An identity directory typically represents a set of users, groups, and roles. Therefore, the first step in defining identities is to define the directory. Once you have defined the identity information, you can use it to write authorization policies and role mapping policies. In authorization and role mapping policies, the user identity (users, groups, roles) is defined in the subject element of the policy.

You may define multiple identity directories. The number of directories you define depends on the level of granularity needed to separate your user community. You may want to have one global directory containing all users. In this case, you can populate a single directory using

multiple external repositories. Having one directory for all users requires that you have a unique name for each user and group across all of your identity repositories. If you cannot guarantee this when you integrate your identity repositories, then you should probably maintain separate directories. For example, you might have one directory for customers, one for employees, and one for partners.

The following topics describe user identity components:

- "Identity Attributes" on page 2-8
- "Groups" on page 2-8
- "Users" on page 2-9
- "Roles" on page 2-10

Identity Attributes

A user or group can contain attributes that further describe their characteristics—who they are and what they can do; these are referred to as identity attributes. You can use these identity attributes to define dynamic constraints for a role to which a user or group belongs. For example, consider that account balance is an attribute of a user. To allow customers with an account balance over $100,000 to access the premier banking features of your application, you define `accountbalance` as an attribute and apply it to each customer in the `bankusers` group (`sgrp`). Next, you define the `premierbanking` role and write a role mapping policy that only allows access to the application if the customer is in the `premierbanking` role. Then you write an authorization policy that defines the privileges you want to allow on the `bankapp` resource and define the subject as the role `premierbanking`.

```
Grant(//role/premierbanking,//app/policy/bankapp,
//sgrp/bankusers/customers/) if accountbalance > 100000
```

This role mapping policy allows customers who are assigned the `premierbanking` role to access the resource called `bankapp` if they have an `accountbalance` of over $100,000.

Groups

A group is typically a collection of users that have something in common, such as a department, a job function, or a job title. For example, a group named Accounting might contain users in the accounting department. It is important to realize that this does not directly reflect what access rights they have. A group can contain either users or other groups; users who are assigned to a

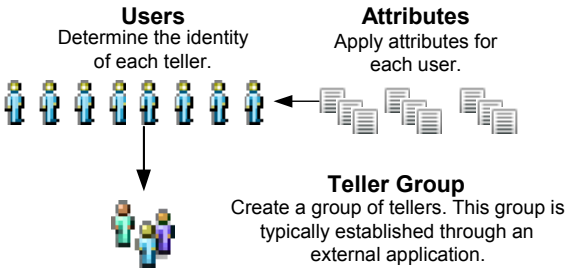
group are called group members. Nested memberships of groups within a group form a hierarchy. Group membership can be assigned only from within the same directory. Groups have a static identity, or name, which you assign.

If a group has subgroups and an authorization policy grants certain privileges to the group, the members of subgroups will have the same privileges. This is true because each member of a subgroup is by default a member of the parent group.

In addition to managing groups in the policy database, AquaLogic Enterprise Security can use group membership information from a corporate directory. Typically, a group hierarchy is based on an organizational model of the company, although this is not a requirement. For example, the source of your user data might be an employee database, where users belong to four groups: the employee group, the Sales department group, the London office group, and the star-salesmen group.

Thus, you want to create groups of users, whose tasks are related and for whom the policy enforcement is the same. In the following example (see Figure 2-2), Tellers are assigned to the Teller Group.

Figure 2-2 Users and Groups



Users

A user corresponds to an individual who makes a request to access a resource, although a user can also be an automated process that accesses a resource. You can assign users to groups from the same directory. Each user within a directory must have a unique identity or user name. Users can be associated with certain characteristics or attributes that contain information about the user. Keep in mind that it may be more efficient to write policies that apply to a collection of users defined as a role or a group. AquaLogic Enterprise Security supports both.

Roles

A role is a set of privileges that can be assigned to a user or group. The actual access privileges assigned to a role are defined by the authorization policy that you write for the role. You write a role mapping policy to assign the role to users and/or groups, thereby granting the access privileges defined by the authorization policy. Once you have written a role mapping policy to assign the role to a user or a group, you can also write a delegation policy to delegate the role from one user to another user or group. Like groups, roles allow you to restrict access to resources for many users or groups at once. However, unlike groups, roles are computed dynamically at runtime based on role mapping policies. Additionally, roles can be associated with specific resources in an application.

Policies

To specify the access control requirements for your resources you write a set of policies that may include role mapping policies, authorization policies, and delegation policies.

The following topics describe the different types of policies:

- "Role Mapping Policies" on page 2-10
- "Authorization Policies" on page 2-12
- "Delegation Policies" on page 2-13
- "Summary of Policy Differences" on page 2-14

Role Mapping Policies

Role mapping policies define when and which to grant roles to users or groups for a particular resource.

The basic format of a role mapping policy is as follows:

```
grant|deny(role, resource, subjects[users, groups]) IF constraints;
```

Where the `grant|deny` portion is the policy effect and either allows or prohibits the role to the subject for the given resource, the `role` defines the role, the `resource` is the application or application component to which the role is scoped, `subjects` specify which users and groups belong to the role, and `constraints` define any conditions that apply to the role.

For example:

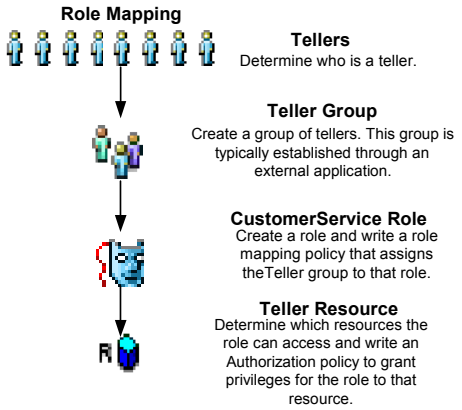
```
GRANT(//role/accountants, //app/policy/acme/payroll, //user/acme/Bill/);
```

grants the `accountants` role to the user `Bill` on the `payroll` resource.

At runtime, user access privileges are computed based on the roles the user has been assigned—either explicitly or through a role mapping policy—at the time an access request is made. Unlike groups, which are relatively static and persist for duration of the runtime session, roles are highly dynamic and are assigned to users by processing role mapping policies. Role mapping significantly reduces the number of policies required and makes features like role delegation easier to manage.

A role may apply to one or more users and groups and usually refers to some set of related tasks. For example, a group of bank tellers might have access to the same set of applications (resources) to perform specific banking tasks; thus, you might have a role called `TellerRole` and assign the BankTellers group to that role. Figure 2-3 shows a group of tellers who belong to a Teller group that has membership in the `CustomerService` role that, in turn, has access to the Teller resource. The privileges, or actions, allowed on that resource are defined by an authorization policy, which you also define. Now, anyone who is not in the `CustomerService` Role does not have access to the Teller resource. You can also apply restrictions and conditions to limit access to the resource at runtime by defining the constraints such as time-of-day or day-of-the-week on the role mapping policy and/or the authorization policy.

Figure 2-3 Role Mapping Policy



Authorization Policies

An authorization policy specifies what a user is allowed to do with a resource. The syntax of an authorization policy is as follows:

```
grant|deny(privilege, resource, subjects[users, groups, roles]) IF
constraints;
```

Where the `grant|deny` portion is the policy effect and either allows or prohibits the privilege to the subject for the given resource, the `privilege` defines the privilege, `resource` defines the application or application component of the privilege, `subjects` specify which users, groups, and roles are granted the privilege, and `constraints` defines any conditions that apply to the privilege.

For example, the policy:

```
GRANT(//priv/any, //app/policy/acme/payroll, //user/acme/agarcia/);
```

grants `any` privileges supported by the `acme payroll` application (the resource) to the user `agarcia` in the `acme` directory. The policy:

```
GRANT(//priv/any, //app/policy/acme/payroll, //role/accountants/);
```

grants any privileges supported by the `acme payroll` application (the resource) to the role `accountants` so only users and groups who have been granted the `accountants` role are granted this privilege. Therefore, before anyone can gain this privilege, a role mapping policy has to be written and deployed that grants this role to a user or a group.

It is important to note that by default, all access to a resource is denied until an authorization policy is written and deployed that explicitly grants an access privilege, or an entitlement, on that resource to a user, group, or role. If the authorization policy only grants an entitlement on a resource to a role, then a role mapping policy must be written and deployed that assigns a user or a group the defined role.

If an authorization policy denies a previously granted entitlement, the deny takes precedence over the grant. Explicit `DENY` authorization policies cannot be overruled. A practical use of a `DENY` policy is to explicitly deny an entitlement to ensure that a user or group can never gain access to a specific resource. For example, the `DENY` authorization policy:

```
DENY (//priv/view, //app/policy/acme/payroll, //sgrp/acme/receptionist/);
```

denies the `view` privilege related to the `acme payroll` application to everyone belonging to the group named `receptionist` in the `acme` directory.

Delegation Policies

The syntax of a delegation policy is as follows

```
DELEGATE (privilege|role, resource, subject, delegator) IF constraint;
```

A `DELEGATE` policy that delegates a privilege allows you to share the privileges of one user with another user, group, or role. You may also add a constraint that restricts this sharing to a certain time of day or day of the week, for example:

```
DELEGATE (//priv/any, //app/policy/acme, //user/acme/joe/,
//user/acme/larry/) if dayofweek in [Monday..Friday];
```

At runtime, this policy delegates `any` privileges that `larry` (the delagator) has on the `acme` application to `joe` if the day of the week is Monday, Tuesday, Wednesday, Thursday, or Friday.

A `DELEGATE` policy that delegates a role allows you to share a role of one user with another user or group. You may also add a constraint that restricts this sharing to a certain time of day or date range, for example:

```
DELEGATE (//role/accountants, //app/policy/acme, //user/acme/joe/,
//user/acme/bill/) IF ThisMonth = December;
```

delegates the role `accountants` on the `acme` application from `bill` (the delagator) to `joe` at runtime if the current mouth is December.

Note: Before a delegator's privilege or role can be delegated, the ARME must verify that the delegator has the privilege or role to be delegated on the specified resource. To perform the verification, the ARME uses information about the delegator (password, groups, roles) that is stored in the policy database to build a Subject for the delegator. If the database does not contain the required information, the delegation policy will not be executed.

Summary of Policy Differences

Table 2-1 summarizes the functions of authorization and role mapping policies and highlights the differences.

Table 2-1 Summary of Policy Differences

Policy Component	Authorization Policy	Role Mapping Policy
Effect (Grant, Deny, or Delegate)	`GRANT` permits the specified privilege to a user, group, or role. DENY denies the privilege to a user, group, or role.	`GRANT` permits the specified role to the specified user or group. DENY denies the role to the specified user or group.
Privilege	The privilege granted or denied.	NA
Role	NA	The role granted or denied.
Resources	The resource to which the privilege is granted or denied.	The resource to which the role is granted or denied.
subjects	The user, group, or role to which the privilege is granted or denied.	The user or group to which the role is granted or denied.
Delegator	The user whose privilege is delegated.	The user whose role is delegated.
Constraints	Conditions under which the privilege is granted, denied, or delegated.	Conditions under which the role is granted, denied, or delegated.

Declarations

A declaration is a variable that represents either a predefined value (for example, days of the week) or a value that is dynamically defined at runtime (the date). You use declarations in policies as attributes. To help you design policies, built-in declarations are pre-defined for your use. You can also define custom declarations to suit your requirements.

You can define four types of declarations:

- “Constants” on page 2-15
- “Enumerated Types” on page 2-15
- “Attributes” on page 2-15

Constants

A constant is a named value or set of values that does not change at runtime. You can reference constants in policies. For example, if you set a constant named `Rate` to 12, policies can then refer to the constant `Rate` rather than using its literal value, 12. Using constants in policies makes them more readable and makes changes to values that are used across of set policies easier

Constants are especially useful if the value changes periodically and you use the constant in more than one location. For example, if you enter a rate value 12 into multiple policies, you need to individually change each one. Instead, if you use the constant Rate, you can edit the value once and have it take effect in every policy that refers to the constant.

Enumerated Types

An enumerated type is a type that consists of a predefined list of values from which you create constants and multi-valued attributes. The product comes with a number of predefined enumerated types and allows you to define your own. For example, you could define the enumerated type "color" with the values of "red", "green", or "blue".

Attributes

Attributes represent characteristics that define dynamic values, users, groups, and configurations. Attributes may be associated with users or groups (identity attributes), resources (resource attributes), or policy requests (dynamic attributes). Attributes may be descriptive, may be used to configure policy engine behavior or manage delegated administration, or used in forming policy as part of the policy constraint.

Attributes must have a defined type, which denotes the range of legal values that it may have. A number of predefined types exist, such as string, integer, date, time, and IP address. You can also use custom enumerated types. The value of the attribute may be assigned to only one instance of an attribute. An attribute may be a multi-valued list.

Evaluation Functions

An evaluation function is a named function that you can use in a policy constraint to perform more advanced operations. Each function may have a number of parameters and returns a Boolean result of true or false.

AquaLogic Enterprise Security provides a number of predefined evaluation functions and also allows you to declare your own custom evaluation functions. You can use a predefined function in your application by using a plug-in extension that a programmer creates specifically for your application. To use an evaluation function, you must register it as a plug-in with the authorization and role mapping providers used in the Security Service Module (SSM) configuration and declare it in a policy. For information about creating and using plug-in extensions, see Provider Extensions in the *Administration Reference*.

Writing Policies

The following topics are covered in this section:

Policy Implementation: Main Steps

To write and deploy a set of policies, perform the following tasks (see Figure 3-1):

Task 1: Define the security requirements for your business. Understand the functions of your applications and the various types of users who need to access them under different circumstances.

Task 2: Define resources. Determine which resources you want to protect and define them in a resource tree. Resources include the resources to be protected, the resource attributes, the privileges, or actions, that will be used to access the resources, and, optionally, privilege groups.

Task 3: Define an identity directory and the identity attributes, users, groups, and roles that are to make up the directory.

Task 4: Define declarations to use with the resources and identities and as constraints in authorization policies, role mapping policies, and delegation policies.

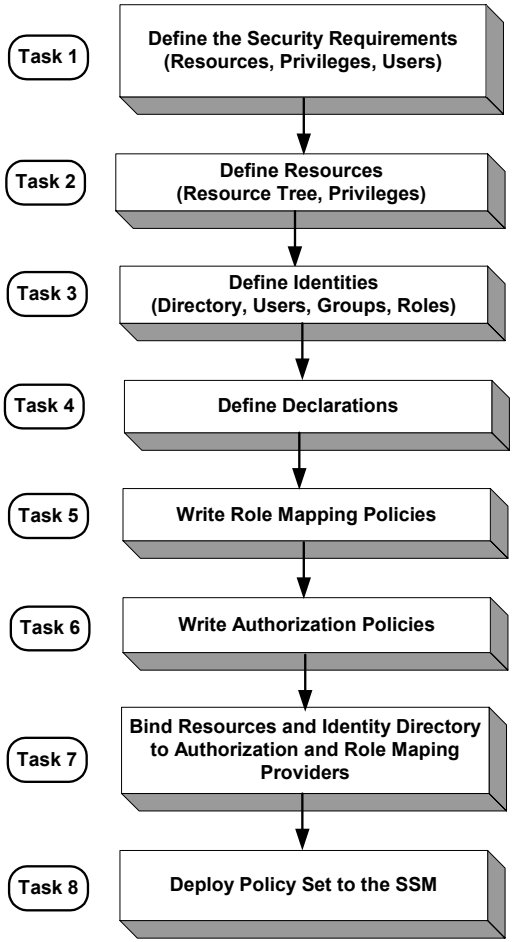
Task 5: Write role mapping policies that control which users and groups have membership in specific roles, under what constraints, and on which resources.

Task 6: Write authorization policies to define which privileges apply to each resource, under what specific conditions, or constraints, and which roles a user or group must have membership in so as to be granted the defined privilege to the specified resource.

Task 7: Bind the top-level resource and the identity directory to the authorization and role mapping providers configured in the security configuration. By doing this you choose which Security Service Module (SSM) enforces policies for these resources.

Task 8: Deploy the set of policies to the SSM. The SSM starts to enforce policies only after the policies are deployed.

Figure 3-1 Policy Implementation Tasks



While the subsequent sections of this document describe how to use the Administration Console to define and manage role mapping and authorization policies, you may also use the Business Logic Manager (BLM) as it offers the same capabilities.

- For instructions on using the Administration Console to perform policy implementation tasks, see "Using the Administration Console to Write Policies" on page 3-6.
- For instructions for using the BLM, see the Javadocs for the Business Logic Manager.

Access Decision Process

For a user to gain access to a resource, AquaLogic Enterprise Security provides the following services and components:

Authentication Service

The authentication service is responsible for authenticating the user. There are two ways a user can be authenticated.

- One way is that the authentication service tells the container what type of credentials are required and then the SSM authenticates those credentials with an external source like a directory or database. ALES supports a wide range of authentication stores and an API is also provided to write custom authentication modules.
- The other way to identify the user is via identity assertion. The ALES SSM authentication service provides a means for plugging in an Identity Assertion service that can be used to validate a token that is provided by an external system. Several Web single sign-on vendors provide Identity Assertion plug-ins for ALES.

You can configure multiple authentication services to authenticate, assert identity, and collect additional group or attribute information at authentication time.

Once the user is authenticated, the service will create a subject object. The subject can contain one or more user principles with attribute information, one or more group principles for the groups in which the user has membership, and any attributes associated with those groups. The subject is provided to the authorization and role mapping services.

Role Mapping Service

Given the subject, which is provided by the authentication service, the role mapping service evaluates the role mapping policies to determine if a user or group is granted a role on the requested resource. Roles provide a level of abstraction between users and the privileges that they

have in a given context (within an application). Role mapping policies can be time-based so that users can delegate their privileges for a limited time (for example, when they go on vacation). Roles are always associated with resources and can be granted broadly or granted only in the context of a particular application resource.

Authorization Service

The actual authorization decision is made (isAccessAllowed) based on identity, group, or role membership.

The authorization service evaluates access control policies to determine what a user can do on a particular resource. Policies can include constraints that the authorization service evaluates against static data, such as user attributes that are retrieved when a user is authenticated. The authorization service can also evaluate constraints against dynamic data that is retrieved at runtime when the policy is evaluated. Further, the authorization service can take application context (for example, EJB parameter values) into account at evaluation time. The authorization service can return authorization decisions or entitlements through the report function.

The authorization service supports the use of multiple authorization providers. This allows you to use ALES in conjunction with an existing entitlements system. ALES can be used to add new capabilities while preserving the existing access control logic. When multiple authorization providers are used, a custom adjudicator is required to determine the outcome when conflicts occur between authorization providers.

Credential Mapping Service

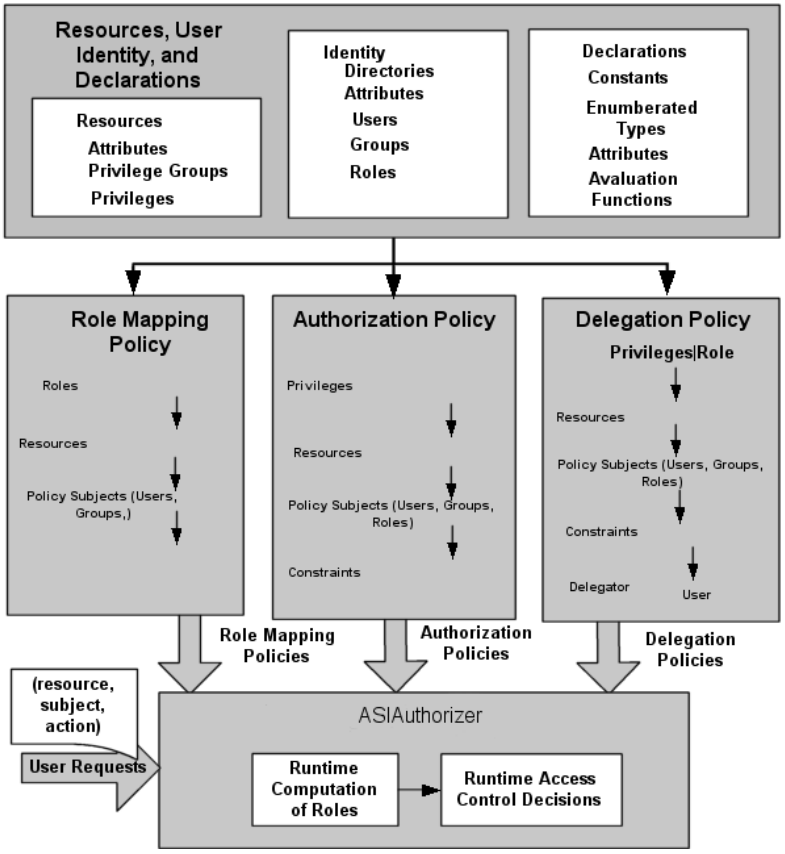
The credential mapping service provides a mechanism to address enterprise single sign-on to enterprise systems. This service can map user identity to an appropriate set of credentials for authentication to enterprise applications like PeopleSoft, SAP, and relational databases. This service can also be used to remove the need to embed credentials within application code. ALES can support a number of identity token formats, such as SAML, to represent the user's identity.

Authorization and Role Mapping Engine

At runtime, the ASI Authorizer, which is also known as the Authorization and Role Mapping Engine (ARME))—located in the Security Service Module instance on which the polices are deployed—uses the role mapping, authorization, and delegation policies to make access control decisions and grant or deny access to users.

Figure 3-2 shows the access control decision process. As illustrated, before you can write policies to define access control for your business resources, you must define those resources, the associated user identity, and, optionally, any custom declarations you may want to use. Once resources and user identify are defined and the policies are written, they do not take effect until they are bound to the Authorization and Role Mapping providers in a Security Service Module (SSM) configuration and deployed. At runtime, user requests to perform actions on specific resources are processed using role-based access control (RBAC) to determine whether the user is granted the requested access (isAccessAllowed).

Figure 3-2 Access Control Decision Process



Using the Administration Console to Write Policies

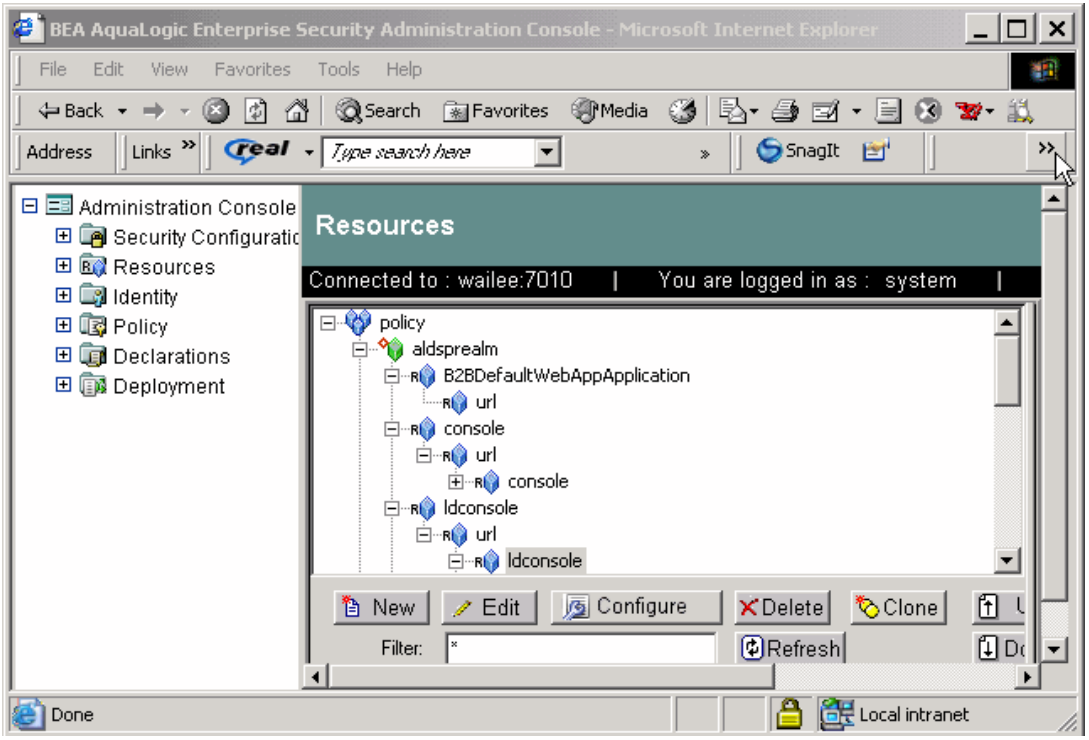
A set of policies control what actions users can perform on resources. A set of policies can be applied to a single resource, an entire application, or implemented globally as a structured collection of entitlements for your organization, representing the superset of all of your application policies.

The following sections describe the components of AquaLogic Enterprise Security policies as presented in the Administration Console and how to write security policies using the Administration Console:

Administration Console Overview

Figure 3-3 shows how the Administration Console represents the various policy components. You use the Security Configuration node to configure security providers for the SSM and to bind the set of policies to the security configuration. You use the Resources, Identity, Policy, and Declarations nodes to write policy. Then you use the Deployment node to deploy the policy set and the security configuration to the SSM.

Figure 3-3 Administration Console



Defining Resources

A resource is a general term that refers to an entity or group of entities that you can protect. Resources can include applications, data, or system components. Resources may also include background services with which the user has no direct interaction.

Figure 3-4 shows the expanded Resource node in the Administration Console.

Figure 3-4 Expanded Resource Node

Clicking Resources displays all of the resources that are defined in the right pane (see Figure 3-5).

Figure 3-5 Defined Resources

The Administration Console displays resources in a tree structure called a resource hierarchy. A page generated from a Java Server Page (JSP) is an example of an application resource. The page can call EJBs or COM resources to execute some business logic. The back office services that transfer money between accounts, issue a payment, or run a report are also resources, although they may not appear on the web page or execute on the application server.

An administrator may define as many resources and levels in the hierarchy as needed to represent data, services, and system components within an application.

Individual resources in the hierarchy are also called nodes and the type of node can convey additional information about the resource. Table 3-1 lists and describes the types of nodes that you can configure using the Administration Console.

Table 3-1 Types of Nodes Supported

Node Type	Console Icon	Description
Organizational		You use organizational nodes to represent organizational structure with the goal of enforcing uniform access control across multiple applications. An organizational node can be configured as a distribution point and to allow virtual resources.
Application		You use application nodes to represent a collection of applications that provide a specific set of services. An application node can also be used to represent an application with a single component, such as a desktop spreadsheet application. An application node can be configured as a distribution point and to allow virtual resources.
Binding		You use binding nodes to represent applications. When you configure the security providers you can use a binding node to bind the authorization and role mapping providers to the application resource tree. A binding node can be configured as a distribution point and to allow virtual resources.
Binding Application		You can use binding application nodes to represent applications. When you configure the security providers you can use a binding application node to bind the authorization and role mapping providers to the application resource tree. A binding application node can be configured as a distribution point and to allow virtual resources.
Resource		You use resource nodes to represent subcomponents of your application. Resource nodes can be used to represent any object within an application, such as data, services, and system components, to which you might want to control access. A binding node or a binding application node can have as many resource nodes as needed at as many levels in the resource hierarchy as necessary. Thus, resource nodes can have other resource nodes as children. A resource node can be configured to allow virtual resources, but it cannot be configured as a distribution point.

Any resource at or above a binding node can be configured as a distribution point for policies. When a distribution, or deployment, is initiated, you can choose to distribute either all updates (by selecting the root node) or you can limit which updates are distributed by selecting resources using the nodes that are configured as distribution points. Only updates that were made at and below the selected nodes are distributed.

Some typical resources that you might want to secure, include:

- An application, an application window, or a dialog box
- Specific business transactions, such as a money transfer or security trade
- Application controls, such as buttons and menu selections
- Database or directory server structures
- Web pages (URLs), servlets, and Enterprise Java Beans (EJB)
- Products or services available through the BEA WebLogic Portal

For instructions on how to define a resource, log into the Administration Console, access the Console Help, find Resources in the left pane, and click Creating a Resource.

The following topics provide more information on configuring resources:

- "Virtual Resources" on page 3-10
- "Resource Attributes" on page 3-11
- "Privileges" on page 3-11
- "Privilege Groups" on page 3-12

Virtual Resources

Any resource defined in the resource tree can be configured as a virtual resource. Once you configure a resource to allow virtual resources, any resources below it, that is, its child resources, are, in effect, virtual resources and are protected by the same policies as their parent, even though they do not appear in the resource tree. For example, given a resource hierarchy URL such as `http://www.myname.com/private/dir1/dir2/`, if you create the resource tree up to `http://www.myname.com/private` and then configure `private` to allow virtual resources, `dir1` and `dir2` are automatically protected by the access control policies you assign to `private`, without having to add `dir1` and `dir2` as explicit resources on the resource tree or assigning them explicit policies.

To configure a resource as virtual, select a resource in the resource tree, click Configure, and check the Allow Virtual Resources check box.

Resource Attributes

A resource attribute is represented under Resources in the Administration Console (see Figure 3-4). All resources can have attributes, which store information about the resources to which they belong. For example, you may create resource attributes to specify resource owner, type of resource, creation date, and so on.

Attributes are inherited by child resources from their parent. If a resource explicitly sets the value for an attribute, this value overrides the inherited one.

For instructions on how to create a resource attribute, log into the Administration Console, access the Console Help, find Resource Attributes in the left help pane, and click Creating a Resource Attribute.

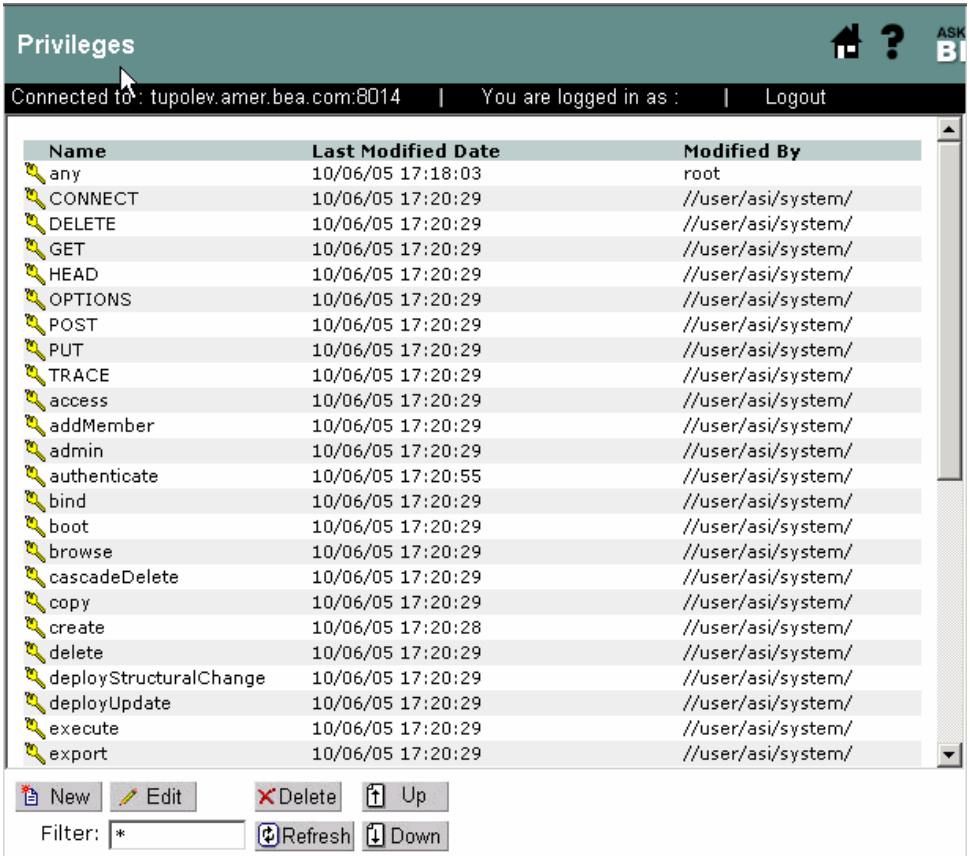
Privileges

A privilege is represented by the key icon in the Administration Console (see Figure 3-4) and is an action that you can perform on a resource. For instance, execute is a typical application privilege; and read and write are typical file-system privileges.

You can use the privileges provided or you can create your own. Figure 3-6 shows how privileges appear in the Administration Console. Notice that each privilege refers to an action. A related collection of privileges may be organized into a privilege group for management purposes.

For instructions on how to create privileges, log into the Administration Console, access the Console Help, find Resources > Privileges in the left help pane, and click Creating a Privilege.

Figure 3-6 Privileges Representation in the Administration Console

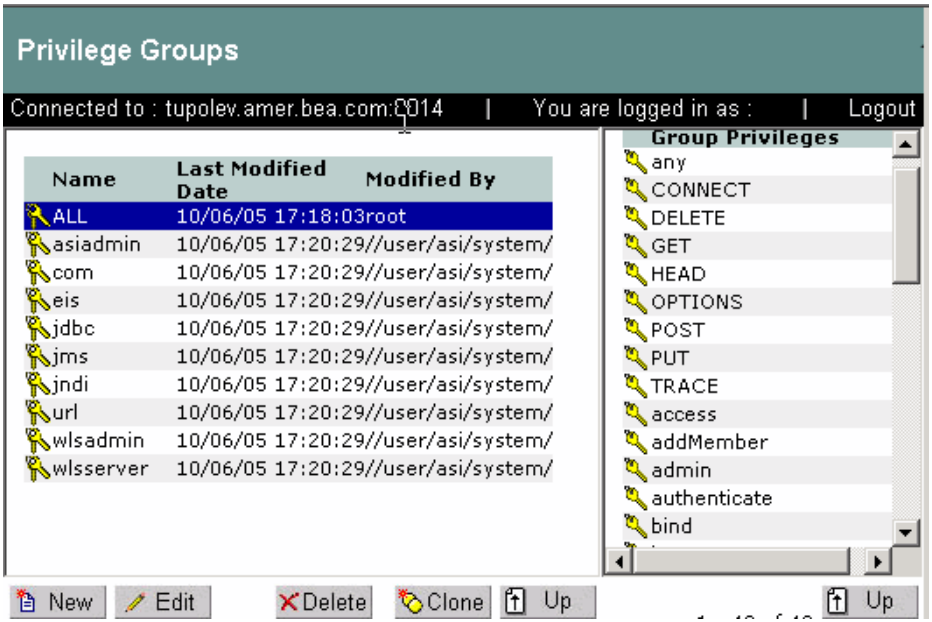


Privilege Groups

A privilege group is represented by the keys icon in the Administration Console (see Figure 3-4) and allows you to organize privileges into logical groups for ease of management. For example, it is common to define a privilege group that applies to a particular application or set of transactions. Privilege groups can be used as filters when constructing policies, although they cannot appear directly in a policy. Figure 3-7 shows an example of how privilege groups and their associated privileges appear in the Administration Console.

For instructions on how to create privilege groups, log into the Administration Console, access the Console Help, find Resources > Privilege Groups in the left help pane, and click Creating a Privilege Group.

Figure 3-7 Privilege Groups Representation in the Administration Console



Defining Identities

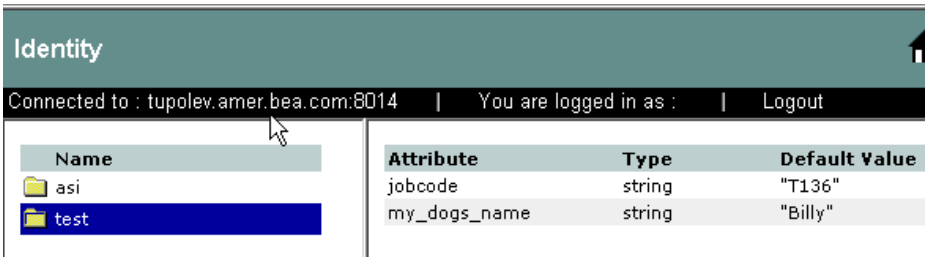
The Identity icon represents your directories and user communities in the Administration Console (see Figure 3-8). Although BEA AquaLogic Enterprise Security provides tools to manage users and groups locally, they are typically managed through an external repository, such as a Lightweight Directory Access Protocol (LDAP) directory server or a network database. User and group information, along with any attributes, is stored as metadata in the policy database and is then available for viewing directly through the Administration Console.

Figure 3-8 shows the expanded Identity node in the Administration Console.

Figure 3-8 Expanded Identity Node

A directory typically represents groups of users of a particular application or resource, or users in a specific location. Each directory has an associated attribute schema. The schema defines the attributes applied to members of the directory. Figure 3-9 shows how directories are represented in the Administration Console. In this example, there is one directory: asi and test. The test directory shows the attributes that are stored for each member of the directory.

Figure 3-9 Directory Representation in the Administration Console



For instructions on how to create directories, log into the Administration Console, access the Console Help, find Identity in the left help pane, and click Creating a Directory.

For more information on configuring identity policy elements, see the following topics:

- "Identity Attributes" on page 3-15
- "Groups" on page 3-15
- "Users" on page 3-16
- "Roles" on page 3-17

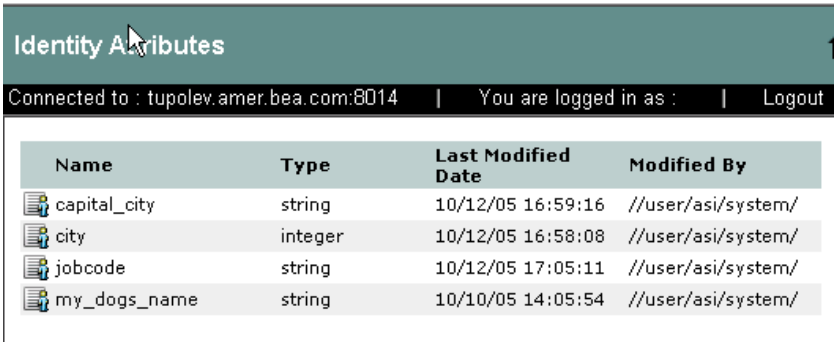
Identity Attributes

Identity attributes are represented under the Identity in the Administration Console (see Figure 3-8). Each user and group can have different characteristics defined as identity attributes. The type of information or attributes collected—a method typically referred to as profiling—also varies and typically includes information such as name and address, phone, e-mail address, personal preferences, and so forth. Identity attributes can be extracted from the external data source.

An identity attribute is declared specifically to contain identity information. An attribute value can be used in policies to set limits for that user. Attributes provide a very powerful way to refer to users and groups indirectly in policies, which results in a more dynamic and versatile policy set.

Figure 3-10 shows how identity attributes are represented in the Administration Console.

Figure 3-10 Identity Attributes Representation in the Administration Console



For instructions on how to create identity attributes, log into the Administration Console, access the Console Help, find Identity Attributes in the left help pane, and click Creating an Identity Attribute.

Groups

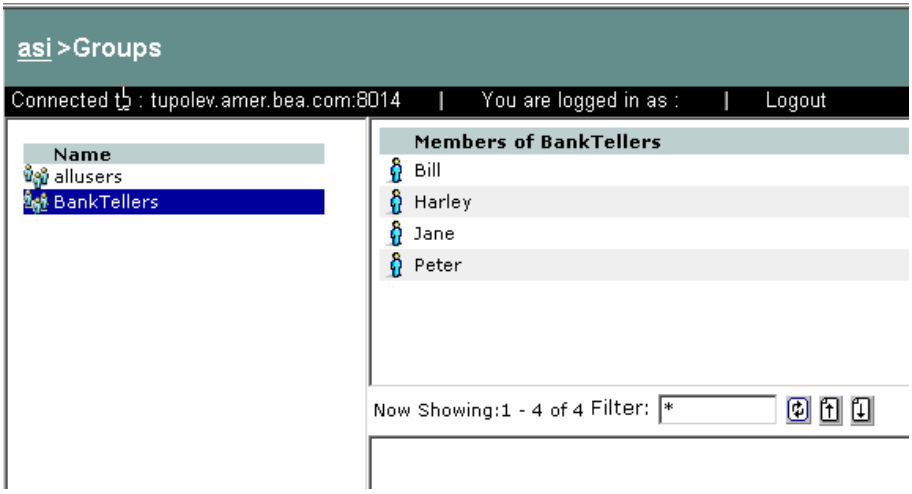
A group is a logical collection of users that share some common characteristics, such as department, job function, or job title. For example, a company may separate its sales staff into two groups, Sales Representatives and Sales Managers, because they want their sales personnel to have different levels of access to resources depending on their job functions.

A group can contain either users or other groups; users who are assigned to a group are called group members. Nested memberships of groups within a group form a hierarchy. Group membership can be assigned only from within the same directory. Groups have a static identity that an administrator assigns.

Managing groups is more efficient than managing large numbers of users individually. By using groups, you do not need to define an access control policy for each and every user. Instead, each user in the group inherits the policies applied to the group; this rule also applies to nested groups. Granting a permission or role to a group is the same as giving that permission or role to each user who is a member of the group. For example, an administrator can specify roles for 50 users at one time by placing the users in a group, and then granting that group the role on a given resource.

Figure 3-11 shows how groups are represented in the Administration Console. Notice that the `BankTellers` group contains four members.

Figure 3-11 Group Representation in the Administration Console



For instructions on how to create groups, log into the Administration Console, access the Console Help, find Identity > Groups in the left help pane, and click Creating a Group.

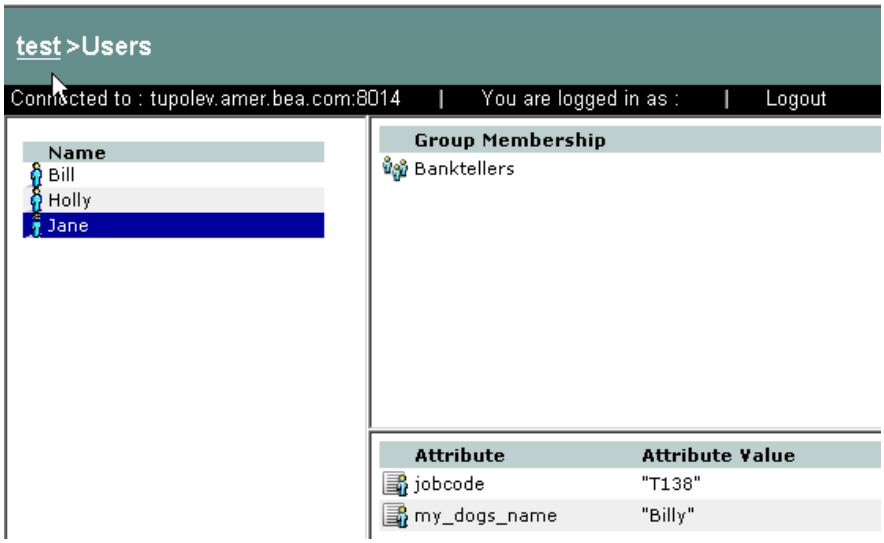
Users

A user corresponds to an individual who makes a request to access a resource, although a user can be an automated process that accesses the system. Users are included in an authorization

policy by assigning users to groups, and then assigning that group to a role or assigning the users directly to roles. Each user within a directory must have a unique identity, or user name.

Users can be associated with certain characteristics, referred to as identity attributes; these attributes store information about the user. The list of attributes that can be set for a user is dictated by the attribute schema of the directory to which the user belongs. Figure 3-12 shows an example of a user representation with identity attributes.

Figure 3-12 User Representation in the Administration Console



For instructions on how to create users, log into the Administration Console, access the Console Help, find Identity > Users in the left help pane, and click Creating a User.

Roles

A role is a dynamic alias used to associate users and groups to role-based functional responsibilities. A role represents a collection of privileges on a resource. Roles are computed and granted to users or groups dynamically based on conditions, such as user name, group membership, identity attributes, or dynamic data, such as the time of day. Roles membership can apply to only specific resources within a single application or can be applied globally across the enterprise. A role can also be delegated from one user to another user. Multiple users or groups can be granted a single security role. Figure 3-13 shows an example of a roles representation in the Administration Console.

Figure 3-13 Roles Representation in the Administration Console

For instructions on how to create roles, log into the Administration Console, access the Console Help, find Identity > Roles in the left help pane, and click Creating a Role.

Writing Authorization and Role Mapping Policies

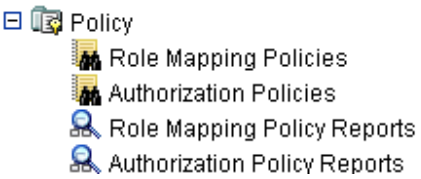
A set of policies can include three types of policies: role mapping policies, authorization polices, and delegation policies:

- Role mapping policies, at a minimum, are written to create roles that define what subjects (user and groups) are assigned to the role for what resources. Role mapping policies can also include constraints.
- Authorization policies are written against resources to define what subjects (users, group, or roles) have what privileges on what resources. Authorization policies can also include constraints.
- Delegation policies are used to assign privileges or roles granted to one user to another user.

Figure 3-14 shows the expanded Policy node in the Administration Console.

Note: Delegation policies that assign roles are listed in the console with role mapping policies. Delegation policies that assign privileges are listed in the console with authorization policies. Delegation policies are distinguished by the Effect type of `DELEGATE` as oppose to `GRANT` or `DENY`.

Figure 3-14 Expanded Policy Node



For more information about policies, refer to the following topics:

- "Role Mapping Policies" on page 3-19
- "Authorization Policies" on page 3-19
- "Role Mapping Policy Reports" on page 3-20
- "Authorization Policy Reports" on page 3-20

Role Mapping Policies

Role mapping policies determine how roles are granted, and to which a user or group can be assigned. Role mapping policies are used to grant users or groups membership into a given role. The membership can be limited based on a number of items including the resource hierarchy, subjects (users and groups), constraints, and delegator. A delegation policy that delegates a role assigns a role granted to one user (the delegator) on a resource to another user or group. A role cannot be delegated to a role. See Figure 3-14 for an illustration of how role mapping policies are represented in the Administration Console.

For instructions on how to write role mapping policies, log into the Administration Console, access the Console Help, find Policy > Role Mapping Policies in the left help pane, and click Creating a Role Mapping Policy.

Authorization Policies

Authorization policies determine what actions can be performed on a resource. Authorization policies are typically written to grant specific privileges upon specific resources to a role with a defined set of constraints. An authorization policy can define privileges, resources, subjects (users, groups, and roles), constraints, and delegators. A delegation policy that delegates a privilege assigns the privileges granted to one user (the delegator) on a resource to another user, group, or role. See Figure 3-14 for an illustration of how authorization policies are represented in the Administration Console.

For instructions on how to write authorization policies, log into the Administration Console, access the Console Help, find Policy > Authorization Policies in the left help pane, and click Creating an Authorization Policy.

Role Mapping Policy Reports

You can use role mapping policy reports to create a role mapping policy inquiry and use it to generate a report that you can use for analysis. You can define inquiries that include a policy subject list (user and group), a role list, a resource list, and a delegator list. Role mapping policy inquiries ask this question, What role is granted to a user or group scoped to a particular resource?

For example, let us say that you want to find out who can access a particular resource. You can run a policy inquiry that includes a resource and an Effect type of GRANT. Such an inquiry produces a complete list of the roles that will be granted to any subject during access to the defined resource. To narrow the inquiry you can add roles, subjects (users and groups) and delegators to the inquiry definition. See Figure 3-14 for an illustration of how role mapping policy reports are represented in the Administration Console.

For instructions on how to create role mapping policy reports, log into the Administration Console, access the Console Help, find Policy > Role Mapping Policy Reports in the left help pane, and click Creating a Role Mapping Policy Report Inquiry.

Authorization Policy Reports

You can use authorization policy reports to create an authorization policy inquiry and use it to generate a report that you can use for analysis. Authorization policy inquiries search for privilege-based policies that match specified characteristics exactly. You can define inquiries that include a policy subject list (user, group and role), a privilege list, a resource list, and a delegator list. Authorization policy inquiries ask this question, Who can do what to what resource?

For example, let us say that you want to find out who with a privilege type of `GRANT` can access a particular resource. You can run a policy inquiry that includes a resource and an Effect type of `GRANT`. Such an inquiry produces a complete list of the users for any subject for any role on the defined resource that has a `GRANT` privilege type. To narrow the inquiry you can add privileges, subjects (users, groups, and roles) and delegators to the inquiry definition. See Figure 3-14 for an illustration of how authorization policy reports are represented in the Administration Console.

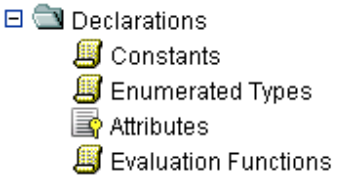
For instructions on how to create authorization policy reports, log into the Administration Console, access the Console Help, find Policy > Authorization Policy Reports in the left help pane, and click Creating an Authorization Policy Report Inquiry.

Defining Declarations

A declaration is a variable that represents either a predefined value (for example, days of the week) or a value that is dynamically defined at runtime (for example, the date). To help you design efficient policies, various built-in declarations are provided for your use.

Figure 3-15 shows the expanded Declarations node in the Administration Console.

Figure 3-15 Expanded Declarations Node



There are four types of declarations:

- **Constants**—A named, predefined, static value, or set of values that you can reference in a policy for a value that does not change at runtime.
- **Enumerated Types**—A type that consists of a predefined list of ordered values from which you create constants and attributes. The system comes with a number of predefined enumerated types and you can define your own. For example, you could define the enumerated type "color" with the values of "red", "green", or "blue".
- **Attributes**—Represents characteristics that define dynamic values, users, groups, resources and configurations. An attribute has an associated type which may either be a built-in type (such as string, integer, date) or an enumerated type. For more information, see "Dynamic Attributes" on page 4-18.
- **Evaluation Functions**—A named function that you can use in a policy constraint to perform more advanced operations. Each function may have a number of parameters and returns a Boolean result. There are a number of built-in evaluation functions and you can declare and use your own custom evaluation functions. Each custom evaluation function must be registered as a plug-in with the authorization and role mapping engine (ARME) that uses it. For more information, see "Evaluation Function Declarations" on page 4-22.

For instructions on how to create the different types of declarations, log into the Administration Console, access the Console Help, find Declarations in the left help pane, and click the following topics:

- Creating a Constant
- Creating an Enumerated Type
- Creating an Attribute
- Creating an Evaluation Function

Binding Policies

You can use the Administration Console to write and deploy a set of policies to protect application resources. A policy set can include role mapping policies, authorization policies, and delegation policies that, taken together, define who has access to which resources. You design and write policies to satisfy the resource access control requirements of your business. Once written, the policy set must be bound to the authorization and role mapping providers that are configured for the SSM that you will use to protect your application resources.

To use the Administration Console to bind the policies to the authorization and role mapping providers, perform the following steps:

1. Open the Security Configuration folder.
2. Open the Service Control Manager folder that contains the Security Service Module whose providers you want to configure or change.
3. Open the Security Service Module folder that contains the providers you want to configure or change.

   Note: If you have not bound the Security Service Module, open the Unbound Configuration folder that contains the providers you want to configure.

4. Open the Authorization folder, and click Authorization.

   The Authorization page appears.

5. Click the Details tab, enter identity directory you defined when you define the user identities and the application deployment parent (`//app/resource`) you defined the top-level resource, and click Save.

Deploying Policies

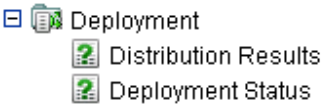
Once you have designed your policies and configuration, you need to deploy them to the SSM so that they can be used to protect your resources. You must distribute both policy and configuration data before they can take effect. You can distribute policy data and configuration data together,

or you can distribute only configuration data as structural changes. After a configuration update, you must restart the SSM for the new configuration to take effect.

Before you distribute policies, you choose the distribution point. The distribution point identifies what portions of the policy updates are distributed.

After the distribution, you can view the results of the policy distribution by clicking on Distribution Results and Deployment Status. Figure 3-16 shows the expanded Deployment node in the Administration Console.

Figure 3-16 Expanded Deployment Node



For more information about deployment and instructions on how to distribute policy data and configuration information, log into the Administration Console, access the Console Help, find Deployments in the left help pane, and click the following topics:

- Distributing Policy
- Viewing Distribution Results
- Viewing Deployment Status

Advanced Topics

This topic describes more advanced aspects of writing role mapping and authorization policies. The following topics are covered here:

Designing More Advanced Policies

All policies, simple or complex, follow the same standard syntax:

```
GRANT|DENY|DELEGATE (privilege|role, resource, subject, delegator) IF
constraint;
```

You can extend the policy syntax to encompass very complex situations by grouping policies and adding constraints.

For more information, see the following topics:

- “Declarations” on page 4-11
- “Policy Inheritance” on page 4-25

Multiple Components

You are not limited to one role, privilege, resource or subject per policy. You may specify sets by enclosing them in brackets [ ] and separating the individual items with commas. For example:

```
GRANT(any, //app/policy/MyApp, [//user/ORG/USER21/, //user/ORG/USER22/]);
```

Policy Constraints

A constraint is a statement that limits when or under what circumstances permission is granted, denied or delegated. All constraints start with the keyword `IF`. Simple constraints usually contain two values separated by an operator. The following example shows an authorization policy with a simple constraint:

```
GRANT(//priv/any, //app/policy/MyApp, //sgrp/ORG/allusers/) IF
purchaseAmount < 2000;
```

In this policy, any `user` of the resource `MyApp` who is in the `ORG` directory is allowed to spend any amount less than $2000.

Constraints are very useful because they allow your application to have different responses based on dynamic application, data, business environment, or real-time conditions. For example, you might use a constraint to grant a user access to a resource only during certain hours of the day.

When checking if a value is within an attribute, the constraint must be written as: <value> in [attribute]. For example if checking to see that the requested resource name is in a list of userentitlements, you would say:

```
sys_obj IN [userentitlements]
```

To limit the user in the previous example to having privileges only in December and January, you would add the constraint:

```
IF month IN [december, january]
```

To limit the user to accessing the application from a computer with a particular static IP address, you would add the constraint:

```
IF clientip = 207.168.100.1
```

Several types of attributes are provided that are automatically computed for you (see “Declarations” on page 4-11).

Once a grant result is determined at runtime by the ASI Authorizer (also called the Authorization and Role Mapping Engine (ARME)) for a particular resource, the rest of the applicable `GRANT` policies, which may contain additional constraints, are ignored. Therefore, if your business logic requires the evaluation of multiple constraints, you must combine them into a complex constraint using an `AND` operator to achieve the desired result. For example, given the following two policies:

```
GRANT(//priv/any, //app/policy/MyApp, //sgrp/ORG/allusers/) IF
purchaseAmount < 2000;
```

```
GRANT(//priv/any, //app/policy/MyApp, //sgrp/ORG/allusers/) IF month IN
[december, january];
```

The conditions under which `allusers` would be granted access would be determined by which policy the ASI Authorizer evaluates first. If the goal is to grant access only if both constraints are true, you must combine these policies into one policy using the `AND` operator as follows:

```
GRANT(//priv/any, //app/policy/MyApp, //sgrp/ORG/allusers/) IF
purchaseAmount < 2000 AND month IN [december, january];
```

For more information on combining multiple constraints into one policy, see "Boolean Operators" on page 4-7.

The following topics provide more information on constraints:

Comparison Operators

Constraints support the comparison operators listed in Table 4-1.

Table 4-1 Comparison Operators

Symbol	Operation	Applicable Types
=	Equal to	All
!=	Not equal to	All
>	Greater than	All except String
<	Less than	All except String
=>	Greater than or equal to	All except String
=<	Less than or equal to	All except String
LIKE	Matches regular expression	String
NOTLIKE	Does not match regular expression	String
IN	Included in a list	List of any type
NOTIN	Not included in a list	List of any type

Regular Expressions

There are two comparison operators, `LIKE` and `NOTLIKE`, that are used to perform regular expression matching on attribute values or string literals. This is typically used for pattern matching on resource names. For example, the following policy provides the `GET` access privilege to all JPGs in a web application (`//app/policy/MyWebApp`).

```
GRANT(//priv/GET, //app/policy/MyWebApp, //role/webusers)

IF sys_obj LIKE '.*\.JPG';
```

The regular expression syntax follows certain policies.

Any character that is not a special character matches itself. Special characters are:

```
+     *     ?     .     [     ]     ^     $
```

A backslash (\) followed by any special character matches the literal character. For example:

```
"\*u"
```

matches `"u"`.

A period (.) matches any character. For example:

```
".ush"
```

matches any string containing the set of characters, such as "Lush" or "Mush".

A set of brackets ([]) indicates a one-character regular expression matching any of the characters in the set. For example:

```
"[abc]"
```

matches either "a", "b", or "c".

A dash (-) indicates a range of characters. For example:

```
"[0-9]"
```

matches any single digit.

A caret (^) at the beginning of a set indicates that any character outside of the set matches. For example:

```
"[^abc]"
```

matches any character other than "a", "b", or "c" not including an empty string.

The following policies are used to build a multi-character regular expressions.

Parentheses (( )) indicate that two regular expressions are combined into one. For example:

```
(ma)+
```

matches one or more instances of "mad's".

The `OR` character (|) indicates a choice of two regular expressions. For example:

```
bell(y|ies)
```

matches either "belly" or "bellies".

A single-character regular expression followed by an asterisk (*) matches zero or more occurrences of the regular expression. For example:

```
"[0-9]*"
```

matches any sequence of digits or an empty string.

A single-character regular expression followed by an plus sign (+) matches one or more occurrences of the regular expression. For example:

```
"[0-9]+"
```

matches any sequence of digits but not an empty string.

A single-character regular expression followed by a question mark (?) matches either zero or one occurrence of the regular expression. For example:

```
"[0-9]?"
```

matches any single digit or an empty string.

A concatenation of regular expression matches the corresponding concatenation of strings. For example:

```
[A-Z][a-z]*
```

matches any word starting with a capital letter.

When you use a regular expression that contains backslashes, the constraint evaluator and the regular expression operation both assume that any backslashes are used to escape the character that follows. To specify a regular expression that exactly matches "a\a", create the regular expression using four backslashes as follows:

```
LIKE "a\\\\a"
```

Likewise, with the period character "." you need to include two backslashes in the expression:

```
LIKE "\\."
```

Constraint Sets

There are two operators, `IN` and `NOTIN`, used to test the memberships of sets in your constraint. A constraint set is a definition of a set of items, notated by one or more values separated by commas, enclosed in square brackets, and prefaced with either the keyword `IN` or `NOTIN`. For example, rather than writing:

```
. . . IF NextMonth = january or
. . . NextMonth = february or
. . . NextMonth = march;
```

You can write:

```
. . . IF NextMonth IN [january, february, march] ;
```

The keyword `IN` means in this set of values, and `NOTIN` means not in this set of values. Neither keyword is case sensitive.

You can also specify a range of values in a set of constraints. For example, the statement:

```
IF age NOTIN[1..100]
```

says if the age value is not between 1 and 100 (inclusive), then the statement is true. The keywords `IN` and `NOTIN` work well with attributes based on enumerated types and constant sets.

String Comparisons

You can test for specific text strings in your constraints by using the keywords `LIKE` and `NOTLIKE`. For example, assume you have a user attribute called `GroupID`. This attribute contains a string of data indicating information about the group the user belongs to:

```
GroupID = "59NY20BREQ";
```

To check for and exclude users in the New York office, you can test the `GroupID` attribute for `NY` as follows:

```
(Grant policy) IF GroupID NOTLIKE "*NY*";
```

where * represents any number of characters. Similarly, if you want to ensure that the user was in New York, you can add this constraint:

```
(Grant policy) IF GroupID LIKE "*NY*";
```

Similar to `IN` and `NOTIN`, `LIKE` and `NOTLIKE` are not case sensitive.

To compare a string to a policy element in the constraint, replace the first characters of the element with a wildcard. Normally, the system does not evaluate a policy element as a string. For example, to compare a user, enter the constraint using the following format:

```
IF user like “??user/acme/Joe/";
```

Boolean Operators

You can build complex policy constraints by using logical operators. Boolean operators allow you to string multiple constraints together and to have the whole constraint return true only if certain patterns of the component constraints are true. For instance, if the whole constraint is only true if both component constraints are true.

If one of them is not true, then the whole constraint is not true, as the following example:

```
(whole constraint) is true IF (first constraint is true) AND (second
constraint is true)
```

Or in another example, where it is true if either component is true:

```
(whole constraint) is true IF (first constraint is true) OR (second
constraint is true)
```

Boolean operators are nothing more than a way to make these kinds of statements. You can write a complex Boolean constraint like this:

```
IF userBudget < 2000 AND ThisMonth = December
```

This constraint is only true if `userBudget` is less than $2000 and the current month is December. Table 4-2 lists the three Boolean operators allowed.

Table 4-2 Boolean Operators

Operator	Description
`AND`	Each component must be true.
`OR`	At least one component must be true.
`NOT`	The component cannot be true.

The third Boolean operator is `NOT`, which simply reverses the truth of a constraint. For example, if you want to make sure it is not December, you can write:

```
IF NOT ThisMonth = December
```

The use of these Boolean operators can get as complex as you want. For example, you can have the following constraint:

```
IF A AND B OR NOT C
```

In English, this means, *If both A and B are true or if C is not true, then the constraint is true.* With a little thought, that is easy enough, but what about a complex constraint, such as:

```
IF A AND B OR C AND NOT D
```

Does it mean, *if A and B are true or C is true and D is not true, grant the privilege*, or does it mean, *if A and B or C is true and D is not true, grant the privilege*, or does it mean something else?

Associativity and Precedence

One way to decipher Boolean expressions is to understand *keyword precedence*, which is the order in which keywords are evaluated, and *associativity*, which is the direction in which terms are grouped. The order of precedence is:

1. NOT
2. AND

3. OR

AND and OR are left associative and NOT is right associative. That is, with AND and OR the system always looks to the immediate left of the keyword for the first value and to the immediate right for the second value. With NOT, the system only looks to the immediate right because NOT does not compare two or more values; it affects only one value. If our earlier example is evaluated using associativity and precedence, it means, *If either both A and B are true or if C is true and D is not, the constraint is true*.

Grouping with Parentheses

Rather than remembering the policies about associativity and precedence, the easiest thing to do is to use parentheses to logically group your AND, OR, and NOT statements.

In the previous example:

```
IF A AND B OR C AND NOT D
```

you can evaluate the statement by applying the policies of associativity and precedence or you can logically group the statements in parentheses as follows:

```
IF (A AND B) OR (C AND NOT D)
```

This eliminates ambiguity from the statement. It becomes clear that there are two constraints: `(A AND B)` and `(C AND NOT D)`, and that one of those constraints must be true for the statement to be true because the two statements have an OR between them.

Changing the location of the parentheses can change the meaning of the statement. For example:

```
IF (A AND B OR C) AND (NOT D)
```

changes the statement completely. Now there are two constraints: `(A AND B OR C)` and `(NOT D)`, in which both must be `true` for the statement to be true.

You may nest parentheses within parentheses to clarify or change the logic of the statement. For example:

```
IF ((A AND B) OR C) AND (NOT D)
```

is the same statement as the previous example, but it is now even clearer. However, if the parentheses are changed slightly, as in:

```
IF (A AND (B OR C)) AND (NOT D)
```

the meaning completely changes.

To understand complex grouped statements with parentheses, follow these policies:

- Evaluate the statements within parentheses first.
- If there are nested parentheses, evaluate the inner ones first.
- Once the statements in parentheses are evaluated, evaluate the other statements.
- If necessary, use associativity and precedence on the simplified statements.

Boolean Operators and Constraint Sets

Rather than building long `OR` or `AND` statements, you can define sets of constraints for your policies. A constraint set defines a set of items. For example, rather than writing:

```
If ThisMonth = january OR ThisMonth = february
OR ThisMonth = march
```

you can write:

```
IF ThisMonth IN [january, february, march]
```

The keyword `IN` means *in this set of values*, and `NOTIN` means *not in this set of values*.

You can also specify a range of values in a set of constraints. For example, the following statement:

```
IF age NOTIN[1..100]
```

says if the `age` value is not between 1 and 100 (inclusive), then the statement is true.

The keywords `IN` and `NOTIN` work well with attributes based on enumerated types and with constant sets.

You may be wondering about the value of constraint sets when the constraint statement is nearly as long as the chain of `OR`s that you would instead have to write. Besides the ability to specify ranges of values, the real benefit to constraint sets is that you can predefine them as constants ("Constant Declarations" on page 4-11). Using the previous example:

```
IF ThisMonth in [january, february, march]
```

using a predefined a constant list called `FirstQuarter`, you can write:

```
IF ThisMonth in FirstQuarter
```

rather than the longer bracketed statement.

Declarations

Declarations allow you to add new keywords to the policy language. These keywords can represent new data types, constants, attributes, or evaluation functions. Declaration names must start with a letter or an underscore. There are four types of declarations:

- Constants–States one definition for a value that is used over and over.
- Enumerated Types–Defines the structure of the other declarations.
- Attributes–Contains data and must have a declared type. There are several types of attributes, including identity attributes (user and group attributes), resource attributes, and built-in system attributes.
- Evaluation Functions–Returns a true or false value from a plug-in.

For programmers, type declarations are enumerated types. Type declarations declare the composition of the enumerated type and define an ordered list of acceptable values. Attributes and evaluation functions declare an instance (variable) of a built-in or enumerated type. Attributes are based on predefined or user-defined types, and evaluation functions are based on Boolean types.

For more information on declarations, see the following topics:

- "Constant Declarations" on page 4-11
- "Enumerated Type Declarations" on page 4-14
- "Attribute Declarations" on page 4-15
- "Evaluation Function Declarations" on page 4-22

Constant Declarations

A constant is a named value or set of values that does not change at runtime. For instance, if you set a constant named `Rate` to `12`, policies can then refer to the constant `Rate` rather than using its literal value, `12`. You use constants to:

- make policies more readable
- make policy-wide value changes easier

Constants are especially useful if the value changes periodically and you use the constant in more than one location. For example, if you enter a rate value 12 into multiple policies, you need to

individually change each one. Instead, if you use the constant `Rate`, you can edit the value once and have it take effect in every policy that refers to it.

Simple Constant

Here are some examples of simple constant declarations:

```
CONST Insurance = "home";

CONST InterestRate= 12;
```

Constants can contain other constants in their value:

```
CONST ClosurePoints = 2;
```

Or even enumerated types:

```
CONST FavoriteVehicle = Motorcycle;
```

If you enclose `Motorcycle` in quotation marks, this constant would contain a string without any special meaning. If you use `Motorcycle` without quotation marks, it is recognized as the special value `Motorcycle` of type `Vehicles`.

Constants List

A constant can also contain a list of more than one value. For example, you may define a constant called `MyColors` with the values `red`, `green`, `blue`, `white` and `black`.

Constant lists differ from enumerated type lists. Types are used to restrict the values an attribute may contain. For example, an integer may only contain numerals and a constant list is simply a declared list or range of values with no implied order. A constant list always has an underlying type. In the previous example, the underlying type is a string. You can also create lists of any other type.

The rules for defining constant lists are as follows:

- Ensure all the constants in a list represent the same data type, including enumerated types.
- Use commas to separate the items in the list.
- Use square brackets [ ] to enclose the whole list.
- Enclose strings in the list with quotation marks.
- If values in a list are a range, indicate the range with two dots. For example, [1..100]. A list of one item is still a valid list, as long as you enclose it in brackets.

Here are some examples of constant lists:

```
CONST MyPets = ["Dogs", "Cats", "Birds"];

CONST CurrentAge = [1..120];

CONST WorkWeek = [monday..friday];

CONST Transportation = [Motorcycle];
```

You can even place another constant list within a constant list, like this:

```
CONST FamilyPets = ["Ferrets", "Birds", MyPets];
```

One benefit of a constant list is that it saves you from having to write multiple policies or string-together constraints to test if a value belongs in a group. Without constant lists, you would need to compare your value to each independent constant, rather than perform one quick test to see if the value belongs in the list. For example, given the constant list:

```
CONST Manager = ["Bert", "Marty", "Sandy"];
```

If you want to find out if your string attribute called `Active` contains a value that is in the `Manager` list, you could write constraints to test for these three possibilities:

```
IF Active = "Bert"

OR Active = "Marty"

OR Active = "Sandy"
```

or you could simply write:

```
IF Active IN Managers
```

As mentioned before, there is no implied order to the `Manager` list. So, even if Bert is clearly a more privileged `Manager` than `Sandy`, the following test is invalid.

```
If "Bert" > "Sandy"
```

For the test to work, you need to create an enumerated type containing the names of the three managers.

Enumerated Type Declarations

An enumerated type defines a class or group of values from which you can create constants and attributes. It is a template for constants and attributes. For example, an attribute of the type integer (a predefined, built-in type) may only have integer values. Many attributes can use the same type declaration, but each attribute is limited to one type, and this type cannot change without deleting and recreating the attribute. For example, you could have dozens of integer attribute variables, but each one is based on the same integer type declaration. Think of an enumerated type declaration as a cookie cutter and attributes as the cookies.

Pre-Defined, Built-In Enumerated Types

The following types are pre-defined and built into the product and are available for you to use. They cannot be modified.

- `date` – A type that limits the data to the format `MM/DD/YYYY` and allows you to compare date values and date ranges within constraints.
- `integer` – A type that contains a whole number with no decimal places that may be negative, positive, or zero. You can use integers in comparisons and ranges.
- `ip` – A type that limits the data to the format allowed for IP (Internet Protocol) addresses: xxx.xxx.xxx.xxx, where xxx is any numeral between 0 and 255, inclusive. You can compare IP addresses, but when defining ranges of IP values, the host number, which is represented by the last three digits, is allowed to vary.
- `string` – A type that contains an alphanumeric text value. Strings do not allow comparison or range operations because they are not ordered. However, you can use wildcard comparisons using `LIKE` and `NOTLIKE` operations.
- `time` – A type that limits data to the format `HH:MM:SS` and allows you to compare time values and time ranges within constraints.

Note: Different types of declarations cannot have the same names as they share the same namespace. For example, you cannot have a constant and an attribute both named `account_number`. In addition, the values of enumerated types share this namespace. So, continuing with our example, you could not create constants or attributes with the values Crows, Ducks, or Geese (or Birds).

User-Defined Types

You can also create custom types. For example, you might create a type called `Insurance` that contains the values `Truck`, `Car` and `Motorcycle`. You would declare it like this:

```
enum_Insurance = (Truck,Car,Motorcycle)
```

Once you declare a type, you must declare an attribute to use the type in your policy. You can declare an attribute based on your new type like this:

```
cred Transportation : Insurance;
```

Once declared, you must give the attribute a value, like this:

```
Transportation = Motorcycle;
```

As mentioned earlier, you can compare the value based on your type by testing if the value is greater to or less than a value in the list. For example, to make your list order represent the relative level of insurance of a vehicle, you might use this constraint to see if your `Transportation` attribute is greater than a `Car` enumeration:

```
IF Transportation > Car
```

If `Transportation` is a `Motorcycle`, given the order of the list defined earlier, this would return `TRUE` and your constraint allows implementation of the policy.

Attribute Declarations

An attribute is a variable that you can use in policies. Attributes store values that are predefined or dynamically defined at runtime.

Declaring an attribute allows you to associate an instance of that attribute with an identity or a resource. For example, you can declare a identity attribute named "`email`" of type "`string`", and then associate email addresses to users.

Attributes make policies more legible by replacing certain constraint values with logical names. You can use attributes to put values in constraints that depend on conditions unknown when you write the policy, such as `timeofday`. Attributes contain values for your input data that your policies can manipulate. That is, they can serve as variables, for example, `account_balance` could be used as an attribute.

There are several ways to use attributes:

- Resource Attribute—Provides a value defined and associated with a resource.
- Identity Attribute—Provides a value defined and associated with a user or group.
- Dynamic Attribute—Provides a value computed or retrieved when the policy is evaluated.

– System Attribute – A dynamic attribute that is computed automatically by the Authorization Provider and available for use in your policy. These attributes usually begin with the prefix `sys_`.

– Time and Date Attributes – System attributes that provide time and date information.

Attributes are specific instances of a declared type. For example, an attribute of the type integer can only contain an integer value. Attributes can represent any type, whether provided as part of the product or defined by you. Here are some examples of attribute declarations:

```
cred month : month_type;
cred timeofday : time;
cred pencils_swiped : integer;
```

For a description of the different types of attributes, see the following topics:

- "Resource Attributes" on page 4-16
- "Identity Attributes" on page 4-17
- "Static Attributes" on page 4-17
- "Dynamic Attributes" on page 4-18
- "Time and Date Attributes" on page 4-18
- "Request Attributes" on page 4-20

Resource Attributes

Resource attributes store information about the entity to which they belong. For example, the `Banking` application might have an attribute called `Version` that contains the current version number for the application, denoted as a string.

Resource attributes behave differently from identity attributes. While they do inherit attributes and their values, they do not merge any values of redundant attributes. If the same attribute exists in more than one place in a tree, the resource first attempts to take the attribute from itself. Failing that, the resource takes the value of the attribute from the first resource above it on the tree that contains the attribute. The attributes of the same name on still higher nodes are ignored; once an instance of the attribute is found, the search ends.

For example, assume that you have an application resource called `Banking` that contains a variety of banking features. `Deposit` is a resource of the `ATMCard` application, which in turn is an application node below the `Banking` organization node. If both the `ATMCard` resource and the

Banking application have the Version attribute defined with a value (and Deposit does not), Deposit inherits the value of the Version attribute from ATMCard. The Banking Version attribute is ignored.

Identity Attributes

User attributes store information about an individual user. For instance, you could have an attribute called AgeRange that stores a range of dates. Attributes are associated with a directory through a directory schema. The schema states that all users of a given directory have a given set of available attributes. Additionally the schema determines if the attribute value is a list.

You can also assign attributes to groups (although groups may only contain list attributes). Thus, users can inherit the attributes of all groups to which they belong. However, a user can still have a unique value for an inherited attribute. If you do not assign the user attribute a value, then the user inherits the value of the attribute from the group. This is how group attributes provide default attribute values for users who are members of those groups. If a user has the same attribute as a group, but a different value is assigned to the user attribute, the value of the user attribute always takes precedence of the value of the group attribute.

Even an empty string, " ", is considered a value for purposes of this rule. Therefore, if you do not assign a value, the user attribute does not take precedence over a group attribute of the same name. However, if you placed an empty string in the user attribute, it does take precedence.

Group attributes behave very differently from user attributes. Group attribute values are cumulative — if the same attribute exists in more than one place in the inheritance path of a user, the values of the attributes are merged and passed on to the user. For example, assume you have a user called Bob, and Bob is a member of the Manager group, which in turn is a member of the Employee group. If both Manager and Employee both have an attribute called WorkPlace with the values primary and secondary respectively, Bob would inherit a WorkPlace attribute with the value primary and secondary (a list attribute). In fact, to support this merging of attribute values, all group attributes must be list attributes. If the attribute merging finds the same value more than once, it eliminates the redundancy from the final list value.

Static Attributes

Many attributes are specific instances of a declaration type. These attributes are often user (identity) attributes. For example, if you had a type called ColorType, you might have the static

credentials `HairColor` and `EyeColor`, which are both of type `ColorType`. You can attach these static attributes to a user. Table 4-3 lists some examples of user attributes.

Table 4-3 User Attributes

Instance	Type
`MonthBorn`	`month_type`
`ArrivalTime`	`time`
`Pencils_needed`	`integer`

As previously discussed, there are several attribute types. Attributes differ from constants in that their value may change, but not the name and value type. Depending on the user making the request, a different value can be calculated for the attribute. In contrast, constants have a static value, as well as a static name and type. The declaration for a user attribute is attached to one or more directories. Because of this, all users in the same directory have the same user attribute names but not necessarily the same values for those attributes. Attributes can be applied to users, groups, and resources; however, each one behaves a bit differently.

Dynamic Attributes

A dynamic attribute is an attribute with a value that may change at policy evaluation time. Dynamic attributes have their value set by the provider, your application, or through a plug-in function. These attributes can have any type of value.

Additionally, plug-ins can be registered to compute the value of dynamic attributes. These plug-ins can retrieve the values of other attributes and use them to compute the attribute value needed.

Time and Date Attributes

Numerous time and date system attributes are pre-defined and built in. Most system attributes allow you to use comparison and range operators. Table 4-4 lists the built-in time and date attributes provided for you to use.

Table 4-4 Built-In Time and Date System Attributes

Attribute	Value	Range or Format
time24	integer	0–2359
time24gmt[1]	integer	0–2359
dayofweek	Dayofweek_type	Sunday–Saturday
dayofweekgmt	Dayofweek_type	Sunday–Saturday
dayofmonth	integer	1–31
dayofmonthgmt	integer	1–31
dayofyear	integer	1–366
dayofyeargmt	integer	1–366
daysinmonth	integer	28–31
daysinyear	integer	365–366
minute	integer	0–59
minutegmt	integer	0–59
month	month_type	January–December
monthgmt	month_type	January–December
year	integer	0–9999
yeargmt	integer	0–9999
timeofday	time	HH:MM:SS
timeofdaygmt	time	HH:MM:SS
hour	integer	0–23
hourgmt	integer	0–23
currentdate	Date	MM/DD/YYYY
currentdategmt	Date	MM/DD/YYYY

1. gmt is an abbreviation for Greenwich Mean Time

Request Attributes

There is a set of system attributes that contain details of the request. Table 4-5 describes these attributes and provides and example of each one.

Table 4-5 Built-In Request System Attributes

Attribute	Value	Range or Format
sys_defined	Evaluation function	Returns true if all arguments passed to it are defined attributes (either single valued or list). Using an undefined attribute in a policy causes a runtime error. This can occur when the value of the attribute is determined from the application code, either through the context handler or the resource object. If there is a chance that the attribute does not have a value, then use the sys_defined evaluation function to ensure that a value exists before it is used. For example `grant(…) if sys_defined(foo) and foo = "bar";`
sys_external_attributes	list of strings	A resource attribute set through the Administration Console on an application resource to indicate what attributes are needed for dynamic evaluation. This contains a list of attribute names.
sys_rule_subj_q	string	Qualified subject user or group name in the currently evaluated policy: //user/ales/system/
sys_rule_subj	string	Unqualified subject user or group name in the currently evaluated policy: system
Servername	string	Name of the server, where ALES process is running.
sys_rule_obj_q	string	Qualified resource name for the currently evaluated policy: //app/policy/foo
sys_rule_obj	string	Unqualified resource name for the currently evaluated policy: foo
sys_rule_priv_q	string	Qualified current policy privilege: //priv/write

Table 4-5 Built-In Request System Attributes (Continued)

Attribute	Value	Range or Format
sys_rule_priv	string	Unqualified current policy privilege: write
sys_subjectgroups_q	list of string	List of groups to which the current user belongs: ["//sgrp/ales/admin/," "//sgrp/ales/managers/"]
sys_subjectgroups	list of strings	List of unqualified group names to which user belongs: ["admin", "managers"]
sys_dir_q	string	Directory of the user: //dir/ales
sys_dir	string	Directory of the user, unqualified form: ales
sys_user_q	string	Current user: //user/ales/system/
sys_user	string	Current user: unqualified form: system
sys_obj_type	enumeration	Set through the Administration Console on the resource. Valid values include: • Organizational node (orgnode) • Application node (appnode) • Binding node (bndnode) • Application Binding node (bndappnode) • Resource node (resnode)
sys_obj_distribution_point	Boolean enumeration {yes, no}	Distribution point set through the Administration Console on the resource. Setting this to yes, displays the resource on the distribution page as a potential point of distribution.
sys_suppress_rule_exceptions	Boolean enumeration {yes, no}	Set through the Administration Console to indicate whether to continue evaluation if a policy with missing data is encountered.
sys_app_q	string	Name of the binding resource for the resource on which query is performed: //app/policy/ALES/admin

Table 4-5 Built-In Request System Attributes (Continued)

Attribute	Value	Range or Format
sys_app	string	Unqualified name of the binding resource for the resource on which the query is performed: admin
sys_obj_q	string	Resource on which the query is performed: //app/policy/foo/bar
sys_obj	string	Resource on which the query is performed: bar
sys_priv_q	string	Effect of the current policy: //priv/foo
sys_priv	string	Unqualified form of the effect of the current policy: foo

sys_privilege	string	Unqualified name of the action on which the resource is being queried.
		The following two policies are equivalent:
		grant(//priv/READ, //app/policy/library, //role/Reader);
		grant(any, //app/policy/library, //role/Reader) if sys_privilege="READ";
		The attribute can also be used in a role-mapping policy. For example, the following policy assigns the role Reader to all users if the requested action is READ:
		grant(//role/Reader, //app/policy/library, //sgrp/asi/allusers/) if sys_privilege="READ";

sys_direction	enumeration	Defines the direction of authorization: once, post or prior.

Evaluation Function Declarations

An evaluation function is a declaration that returns one of two values: true or false. These values come from a predefined function and are included by using a plug-in extension that a programmer creates specifically for your application. Additionally, you can use any of the built-in evaluation functions available in all applications.

For instance, your programmer might create a plug-in for your accounting application that includes an evaluation function called Overdrawn that contains the results of a calculation of

whether the account was overdrawn for that month. A constraint for a deny policy might use that function like this:

```
[Deny user access to something] IF Overdrawn();
```

Like functions and procedures in programming, evaluation functions can take zero or more parameter values, which are passed to the plug-in. For example, if you wanted to provide the overdrawn amount, you might use it like this:

```
[Deny user access to something] IF Overdrawn(500);
```

Evaluation functions can dynamically take different numbers or types of parameter values each time they are referenced in a policy. It is up to the programmer writing the evaluation function code to correctly handle the parameters.

Authorization Caching Expiration Functions

Authorization caching allows the system to cache the result of an authorization call and use that result if future identical calls are made. The cache is smart and automatically invalidates itself if there is a policy change or other client side change that would affect the authorization results. However, the cache is not smart enough to know when authorization decisions depend on dynamic data. Dynamic data includes date and time values, as well as evaluation plug-ins that reference external sources. If you are using authorization caching you need to set expiration times on policies that reference dynamic data. For additional information on caching, see Authorization Caching, in *Integrating ALES with Application Environments*.

Note: By default, authorization caching is turned on.

Table 4-6 lists the expiration functions for the authorization cache that let you set an expiration time for the authorization decision. This way you can instruct the cache to only hold the value for a given period of time, based on Greenwich Mean Time (GMT), or not to hold it at all.

Table 4-6 Expiration Functions for Authorization Cache

Function	Argument Type	Description
`valid_for_mseconds`	`integer`	Valid for a given number of milliseconds.
`valid_for_seconds`	`integer`	Valid for a given number of seconds.
`valid_for_minutes`	`integer`	Valid for a given number of minutes.
`valid_for_hours`	`integer`	Valid for a given number of hours.

Table 4-6 Expiration Functions for Authorization Cache (Continued)

Function	Argument Type	Description
valid_until_timeofday	time	Valid until the specified time on the date the evaluation is performed.
valid_until_time24	integer	Valid until the specified time on the date the evaluation is performed.
valid_until_hour	integer	Valid until the specified hour on the date the evaluation is performed.
valid_until_minute	integer	Valid until the specified minute of the hour the evaluation is performed.
valid_until_date	Date	Valid until the specified date.
valid_until_year	integer	Valid until the specified year.
valid_until_month	month_type	Valid until the specified month of the year the evaluation is performed.
valid_until_dayofyear	integer	Valid until the specified day of the year the evaluation is performed
valid_until_dayofmonth	integer	Valid until the specified day of the month the evaluation is performed.
valid_until_dayofweek	Dayofweek_type	Valid until the specified day of the week the evaluation is performed.
valid_until_timeofday_gmt	time	Valid until the specified time on the date the evaluation is performed in GMT time.
valid_until_time24_gmt	integer	Valid until the specified time on the date the evaluation is performed in GMT time.
valid_until_hour_gmt	integer	Valid until the specified minute of the hour the evaluation is performed in GMT time.
valid_until_minute_gmt	integer	Valid until the specified minute of the hour the evaluation is performed in GMT time.
valid_until_date_gmt	Date	Valid until the specified date in GMT time.
valid_until_year_gmt	integer	Valid until the specified year in GMT time.

Table 4-6 Expiration Functions for Authorization Cache (Continued)

Function	Argument Type	Description
`valid_until_month_gmt`	`month_type`	Valid until the specified month of the year the evaluation is performed in GMT time.
`valid_until_dayofyear_gmt`	`integer`	Valid until the specified day of the year the evaluation is performed in GMT time.
`valid_until_dayofmonth_gmt`	`integer`	Valid until the specified day of the month the evaluation is performed in GMT time.
`valid_until_dayofweek_gmt`	`Dayofweek_type`	Valid until the specified day of the week the evaluation is performed in GMT time.

For example, suppose you have the following authorization policy:

```
GRANT(//priv/order,//app/restaurant/breakfast,//group/customers/allusers)
if hour < 11;
```

With authorization caching enabled (it is enabled by default), the results of this grant decision is cached until the next policy distribution.

On the other hand, if you call the `valid_until_hour()` expiration function in the authorization policy as follows:

```
GRANT(//priv/order,//app/restaurant/breakfast,//group/customers/allusers) if
hour < 11 and valid_until_hour(11);
```

with authorization caching, the result of this policy is cached until 11:00 AM, at which time it expires. Therefore, with authorization caching enabled, it is important to update your time dependent policies appropriately.

Policy Inheritance

Using policy inheritance can reduce the number of policies that have to written to protect a set of resources. The following topics describe how inheritance works:

- "Group Inheritance" on page 4-26
- "Direct and Indirect Group Membership" on page 4-26
- "Restricting Policy Inheritance" on page 4-27

- "Resource Attribute Inheritance" on page 4-27

Group Inheritance

Users or groups inherit the right (privilege or role) of any group to which they belong, either directly or through their parents. Group inheritance allows each user in the group to assume all the group rights to which they are members, either directly or indirectly through their parent groups (or the groups of their parents). Both users and groups can have parent groups but only groups can have children. Group inheritance is very powerful as it allows you to define entitlements once and have the policy apply to all members.

Note: BEA recommends that you define your role mapping policies using groups, rather than individual users. Role mapping policies written using users should be used for exceptions and to handle unusual or infrequent situations.

It is important to note that parent groups usually have fewer rights than their children. As you move from the bottom of the resource tree to the top, the groups inherit the rights of their ancestors and are directly granted.

Direct and Indirect Group Membership

The immediate members of a group are called direct members. Direct members appear immediately below their parent on the inheritance tree. A member that has an inherited membership is called indirect member. The collection of all groups available, either directly or through inheritance, is referred to as group closure.

Group inheritance behavior is affected by how group membership searching is configured in your security providers. Two attributes control group membership searching:

- GroupMembershipSearching—Specifies whether group membership searching traverses all the group's children or a limited number of levels of child groups.
- MaxGroupMembershipSearchLevel—If GroupMembershipSearching is set to limited, specifies the number of levels of child groups to search. A value of 0 indicates only direct group memberships will be found. A positive number indicates the number of levels to go down.

If you set GroupMembershipSearching to unlimited, all indirect members will be considered when a policy is evaluated. If you set GroupMembershipSearching to limited, only indirect members within the number of levels of inheritance specified by MaxGroupMembershipSearchLevel will be considered.

Restricting Policy Inheritance

Policies are inherited in a number of ways:

- Policies written on a resource apply to the descendants of that resource.
- Policies written on a group apply to all members of that group.
- Policies written on a role apply to everyone who has been granted that role.
- Policies written on the `any` privilege apply to all privileges.

You can restrict policy inheritance by limiting its applicability. For example, you can limit the applicability of a `GRANT` role mapping policy by adding a constraint. The following policy illustrates this:

```
GRANT(//role/admin, //app/policy/www.myserver.com/protected,
//sgrp/acme/manager/) IF sys_obj_q =
//app/policy/www.myserver.com/protected;
```

where: `sys_obj_q` is a system attribute on which the query is performed.

The sys_obj_q constraint keeps this policy from being applicable to the descendants of the `protected` resource, thus blocking policy inheritance.

Resource Attribute Inheritance

Like users and groups, descendant resources also inherit the attributes of any parent resource. Resource inheritance allows each child resource in the tree to assume all of the attributes of the parent resource. Resource attribute inheritance is also very powerful as it allows you to define attributes on the parent resource, and have the attributes be inherited to all child resources automatically.

Note: BEA recommends that you define your attributes on parents, rather than individual child resources. When an attribute is explicitly defined for a child, the attribute overrides any inherited value. Policies written directly for child resources should be used for exceptions or short-lived policies so as to handle unusual circumstances.

WebLogic Resource Type Conversions and Resource Trees

This section describes how ALES converts the different resource types supported by WebLogic Server, WebLogic Portal, AquaLogic Data Services Platform, and AquaLogic Service Bus and how they are represented in a resource tree in the Administration Console.

Table 4-7 lists the resource types supported for WebLogic Server, WebLogic Portal, AquaLogic Data Services Platform, and AquaLogic Service Bus.

Table 4-7 Supported Resource Types

Target System	Supported Resource Types
WebLogic Server	<adm>, <app>, <com>, <eis>, <ejb>, <jdbc>, <jms>, <jndi>, <ld>, <svr>, <url>, <web>, <webservices>
WebLogic Portal	All WebLogic Server resources plus <wlp>.
AquaLogic Data Services Platform	All WebLogic Server resources plus <ld>.
AquaLogic Service Bus	All WebLogic Server resources plus <wlsb-proxy-service> and <alsb-proxy-service>.

Understanding Resource Nodes

An authorization policy involves a resource, action, subject and attributes. Every resource is represented as a node within a tree, and the node is referenced using a path-like expression. The nodes are delimited by the '/' character and can include the following hierarchy of nodes:

1. root node
2. application deployment parent node
3. application node
4. resource type node
5. resource parent node

6. resource node

Root Node

The root node of resources in ALES is a node named `//app/policy`.

Application Deployment Parent Node

Typically a node called an application deployment parent follows the root node. Using multiple application deployment parent nodes helps to organize resources according to their physical, organizational or logical structure. The application deployment parent can be set in the Administration Console under the authorization provider.

Application Node

The application deployment parent is followed by the application node that corresponds to an application a resource is associated with. Not every resource belongs to a particular application (for example, a JDBC resource); in that case, the keyword `shared` substitutes for the name of the application.

Resource Type Node

The next level in the resource path is the resource type node. The name of this node corresponds to a resource type being addressed, for example, `jms`, `ejb`, `jndi`, etc.

Resource Parent Node

The resource type node is followed by the resource parent node. The resource parent node helps to organize resources within an application and its value depends on the type of the resource.

Resource Node

The final element in a resource description is the name of the resource itself, which follows the resource parent node.

Thus, to address any resource in the resource tree, it is necessary to know the following resource path elements:

- application deployment parent
- application name
- resource type

- resource parent
- resource name

The application deployment parent depends only on the configuration of the authorization provider; the remaining four elements vary from one resource type to another.

Table 4-8 gives an example of how the different resource type can be represented in the Administration Console resource tree.

Table 4-8 Examples of Mapping Resource Types to Resource Nodes

Resource Type	Sample Resource Tree Conversions

<adm>	//app/policy/ALES/shared/adm
	//app/policy/ALES/shared/adm/Configuration
	//app/policy/ALES/shared/adm/FileDownload
	//app/policy/ALES/shared/adm/FileUpload
	//app/policy/ALES/shared/adm/ViewLog

<app>	//app/policy/essdemo/myapplication/app //app/policy/essdemo/anotherapplication/app
<com>	//app/policy/essdemo/comapplication/com/classpackage/classname
<eis>	//app/policy/essdemo/shared/eis
<ejb>	//app/policy/essdemo/ess/ejb/netuix.jar //app/policy/essdemo/ess/ejb/netuix.jar/PortalCustomizationManager
<jdbc>	//app/policy/essdemo/shared/jdbc/ConnectionPool //app/policy/essdemo/shared/jdbc/ConnectionPool/MyPool-DB
<jms>	//app/policy/essdemo/shared/jms/queue //app/policy/essdemo/shared/jms/queue/jms

<jndi>	//app/policy/essdemo/shared/jndi/jms
	//app/policy/essdemo/shared/jndi/weblogic
	//app/policy/essdemo/shared/jndi/weblogic/jms
	//app/policy/essdemo/shared/jndi/weblogic/jms/MessageDrivenBeanConnectionFactory
	//app/policy/essdemo/shared/jndi/weblogic/jms/S:MedRecServer
	//app/policy/essdemo/shared/jndi/weblogic/management
	//app/policy/essdemo/shared/jndi/weblogic/management/home
	//app/policy/essdemo/shared/jndi/weblogic/management/home/localhome

Table 4-8 Examples of Mapping Resource Types to Resource Nodes

Resource Type	Sample Resource Tree Conversions
<svr>	//app/policy/essdemo/shared/svr

<url>	//app/policy/essdemo/ess/url/demolaunch
	//app/policy/essdemo/ess/url/demolaunch/launch.portal
	//app/policy/essdemo/ess/url/demolaunch/framework
	//app/policy/essdemo/ess/url/demolaunch/framework/skins
	//app/policy/essdemo/ess/url/demolaunch/resources
	//app/policy/essdemo/ess/url/demolaunch/resources/images

<webservices>	//app/policy/essdemo/shared/webservices

<wlp>	//app/policy/essdemo/ess/wlp/essWeb/com_bea_p13n
	//app/policy/essdemo/ess/wlp/essWeb/com_bea_p13n/Page
	//app/policy/essdemo/ess/wlp/essWeb/com_bea_p13n/Desktop
	//app/policy/essdemo/ess/wlp/essWeb/com_bea_p13n/Book
	//app/policy/essdemo/ess/wlp/essWeb/com_bea_p13n/Portlet

<ld>	//app/policy/essdemo/shared/ld //app/policy/myrealm/RTLApp/ld/DataServices/RTLServices/CustomerView.ds/CUSTOMER/ORDERS/ORDER_SUMMARY/OrderDate

Resource Paths and Policies for Common Resources

This section describes the values of resource path elements for most popular resource types. For each resource type, we describe how to specify the resource path and privileges, list dynamic resource attributes are available, and give examples of policies for that resource type. In the examples in this section, we assume that the application deployment parent node is `//app/policy/AppParentNode`.

- "EJB Resources" on page 4-32
- "JNDI Resources" on page 4-34
- "URL Resources" on page 4-36
- "JDBC Resources" on page 4-41
- "JMS Resources" on page 4-44

EJB Resources

Table 4-9 shows the mapping of the resource path elements for an EJB resource.

Table 4-9 EJB Resource Path Elements

Element Name	Value
application name	Same as the EJB application name
resource type	`ejb`
resource parent	Same as the EJB module name
resource name	EJB_NAME/METHOD_NAME, where: • EJB_NAME is the name of the EJB • METHOD_NAME is the invoked method name

EJB Resource Path Example

For the purposes of this example, suppose you have an EJB application named `MyEjbApplication` and a module named `MyManagers`, configured by the following EJB application declaration:

```
<Application Name="MyEjbApplication" Path="./applications"
             StagingMode="nostage" TwoPhase="true">
  <EJBComponent Name="MyManagers" Targets="myserver" URI="managers.jar"/>
</Application>
```

Listing 4-1 shows how an EJB named `AccountService` could be defined in the standard EJB `ejb-jar.xml` deployment descriptor:

Listing 4-1 EJB Configuration

```
<enterprise-beans>
<!-- Session Beans -->
```

```
  <session>
    <display-name>AccountService</display-name>
    <ejb-name>AccountService</ejb-name>
    <home>com.bea.security.examples.ejb.AccountServiceHome</home>
    <remote>com.bea.security.examples.ejb.AccountService</remote>
    <ejb-class>ejb.AccountServiceSession</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Bean</transaction-type>
  </session>
</enterprise-beans>
```

In the case of a `getBalance` method call on the `AccountService` stateless session bean defined by the configuration in this example, the fully qualified resource path would be:

```
//app/policy/AppParentNode/MyEjbApplication/ejb/MyManagers/AccountService/
getBalance
```

EJB Resource Privilege Mappings

ALES version 2.6 changes the way the EJB Resource converter works. As of this release, the EJB method becomes a part of the resource URL, instead of being the privilege. The privilege now is always equal to `execute`.

Consider the example of `AccountEJB` with method `transferMoney`. Prior to ALES version 2.6, the resource would be `$some_prefix/AccountEJB` and the privilege would be `transferMoney`. As of this release, the resource is `$some_prefix/AccountEJB/transferMoney` and the privilege is `execute`.

In releases prior to ALES 2.6, the privilege required to access an EJB resource was the method name called on the EJB. For example, assume that the `AccountService` bean has a business method called `getBalance()`. To be able to call the `getBalance()` method, the user must be granted the `getBalance` privilege. In order for the user to be able to instantiate the remote interface by calling the `create()` method on the EJB home interface, the user must be granted the `create` privilege.

EJB Resource Dynamic Resource Attributes

The following attributes are supported by EJB resources and can be used as a part of an authorization policy:

application
: The name of the application

module
The name of the module

ejb
the name of the EJB

method
the name of the method

methodinterface
One of the values Home, Remote, LocalHome, or Local

Param<N>
A value of the Nth parameter in the method, e.g. Param1, Param2…

For an example that illustrates EJB resources, see `WLS_SSM_HOME/examples/EJBAppExample`.

JNDI Resources

Table 4-10 shows the mapping of the resource path elements for a JNDI resource.

Table 4-10 JNDI Resource Path Elements

Element Name	Value
application name	`shared`
resource type	`jndi`
resource parent	The JNDI resource path
resource name	Not used

JNDI Resource Path Example

Listing 4-2 is an extract from `weblogic-ejb-jar.xml` that defines the JNDI name of the `AccountService` EJB used in "EJB Resource Path Example" on page 4-32.

Listing 4-2 JNDI Name Definition

```
<weblogic-ejb-jar>
  <weblogic-enterprise-bean>
    <ejb-name>AccountService</ejb-name>
```

```
    <stateless-session-descriptor></stateless-session-descriptor>
    <reference-descriptor></reference-descriptor>
    <jndi-name>AccountService</jndi-name>
  </weblogic-enterprise-bean>
</weblogic-ejb-jar>
```

The fully qualified resource name that corresponds to the physical JNDI name of the AccountService EJB home interface would be:

```
//app/policy/AppParentNode/shared/jndi/AccountService
```

JNDI Resource Privilege Mappings

The privilege for a JNDI call is the JNDI action name. The privilege value can have one of the following values:

modify
: Required whenever an application modifies the JNDI tree in any way (that is, adding, removing, changing). This includes the `bind()`, `rebind()`, `createSubContext()`, `destroySubContext()`, and `unbind()` methods.

lookup
: Required whenever an application looks up an object in the JNDI tree. This includes the `lookup()` and `lookupLink()` methods.

list
: Required whenever an application lists the contents of a context in JNDI. This includes the `list()` and `listBindings()` methods

JNDI Dynamic Resource Attributes

The following dynamic attributes are supported by JNDI resources and can be used as a part of an authorization policy:

application
: Always `shared`.

path
: The JNDI resource path.

action
: The JNDI action name (`modify` | `lookup` | `list`).

JNDI Resource Policy Examples

The following policy grants the group `Everyone` a privilege to perform the `lookup` operation on any JNDI resource. Note that the resource `//app/policy/AppParentNode/shared/jndi` must be a virtual one.

```
grant(//priv/lookup, //app/policy/mybank/shared/jndi, //role/Everyone) if
true;
```

The following policy grants the role `Admin` a privilege to modify a JNDI resource named `DataSource`. This will allow a user who has been assigned the `Admin` role to perform such operations as `bind()` and `unbind()`.

```
grant(//priv/modify, //app/policy/mybank/jndi/DataSource, //role/Admin) if
true;
```

URL Resources

Table 4-11 shows the mapping of the resource path elements for a URL resource.

Table 4-11 URL Resource Path Elements

Element Name	Value
application name	The name of the web application that contains the resource
resource type	`url`
resource parent	The context path of the web application
resource name	The resource URI after the context path

URL Resource Path Example

In this example, assume that:

- there is a web resource accessible through the URL `http://localhost/helloworld/HelloWorld.jsp`
- the web server configuration file references the web application as `HelloWorldApp` with the context path `/helloworld`

To protect the resource, it is necessary to know how the JSP page is represented in the resource tree. In this example, the elements of the resource path are:

- Application Name – the name of the web application,. `HelloWorldApp`
- Resource Type – `url`
- Resource Parent – the context path of the web application, `helloworld`
- Resource Name – the JSP name, `HelloWorld.jsp`

The resulting resource representation is:

```
//app/policy/AppParentNode/HelloWorldApp/url/helloworld/HelloWorld.jsp
```

URL Resource Privilege Mappings

In case of a URL resource, the privilege name is mapped to the HTTP request method name: GET, POST, PUT, HEAD, DELETE, TRACE, CONNECT, etc.

URL Dynamic Resource Attributes

The following dynamic attributes are supported by URL resources and can be used as a part of an authorization policy:

application
: The name of the web application.

contextpath
: The context path of the web application.

uri
: The URI of the resource.

httpmethod
: The HTTP method (same as privilege).

transporttype
: The transport guarantee required to access the URL resource, as it appears in the corresponding `<transport-guarantee>` element in the deployment descriptor. The value can be one of `INTEGRAL` or `CONFIDENTIAL`.

authtype
: The name of the authentication scheme used to protect the servlet. The value can be one of: `BASIC`, `FORM`, `CLIENT_CERT` or `DIGEST`.

pathInfo
: Extra path information associated with the URL sent by the client when it made a request.

pathtranslated
Extra path information after the servlet name but before the query string is translated to a real path.

querystring
The query string that is contained in the request URL after the path.

remoteuser
The login of the user making the request, if the user has been authenticated.

requestedsessionid
The session ID specified by the client.

requesturi
The part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.

requesturl
The URL the client used to make the request. The returned URL contains a protocol, server name, port number, and server path, but it does not include query string parameters.

servletpath
The part of this request's URL that calls the servlet.

characterencoding
The character encoding used in the body of the request.

contenttype
The MIME type of the body of the request.

locale
The preferred Locale of the client.

protocol
The name and version of the protocol, for example, HTTP/1.1.

remoteaddr
The Internet Protocol address of the client or last proxy that sent the request.

remotehost
The fully qualified name of the client or the last proxy that sent the request.

scheme
The name of the scheme used to make this request, for example, `http`, `https`, or `ftp`.

servername
The host name of the server to which the request was sent.

serverport
The port number to which the request was sent.

issecure
A boolean indicating whether this request was made using a secure channel, such as HTTPS.

HTTP Request Context Elements

HTTP request context elements such as servlet attributes, URL query parameters, HTTP request headers and cookies are available as name/value pairs. This section describes how to access the following elements while creating authorization policy constraints:

- "Servlet Attributes" on page 4-39
- "URL Query Strings" on page 4-40
- "HTTP Request Headers" on page 4-40
- "Cookies" on page 4-40

The attributes that correspond to servlet attributes, URL query parameters, HTTP request headers and cookies are case insensitive; however, an assumption that the attribute names are case sensitive will slightly improve the performance.

If names of a servlet attribute, URL query parameter, HTTP request header, or cookie collide, only one attribute will be available in policy constraints. The order the framework searches for a matching attribute is:

1. URL query parameters
2. servlet attributes
3. HTTP request headers
4. cookies

Servlet Attributes

Servlet attributes are name/value pairs that can be internally added to a request by a servlet container. Usually the attributes are added by calling method `setAttribute` of the

`ServletRequest` interface. The policy attribute names correspond to the names of servlet attributes. The names are represented as strings and case insensitive.

URL Query Strings

The attribute names that correspond to the parameters in a URL query string are the same as the parameter names. The names are represented as strings and are case insensitive. The attributes refer to the query string variable encoded within the request. For example, if a URL includes a query such as `?test=endcoded%20char`, the parameter can be accessed in the constraint of an authorization policy in the following way:

```
"if test= "encoded char"
```

HTTP Request Headers

The attribute name of an HTTP request header corresponds to the name of the header. The name is returned as a string and is case insensitive. Examples of the headers often available are: `date`, `if-modified-since`, `referrer`, or `user-agent`.

Note: The date header, which is usually a date type, is returned as a string.

Cookies

The attribute names that correspond to cookies in an HTTP request are the same as the cookie name in the request. The names are returned as strings and case insensitive. The value of the cookie returned is application specific and may need further decoding. For example, if you are using the ALES cookie, the attribute name is:

```
"ALESIdentityAssertion"
```

URL Resource Policy Examples

The following policy grants user `anonymous` (any unauthorized user) a privilege to view current currency exchange rates (the page `currentRates.jsp`) but only if the connection is secure (for example, through HTTPS).

```
grant( //priv/GET,
//app/policy/mybank/bankapp/url/currencyExchange/currentRates.jsp,
//user/myusers/anonymous/) if issecure=yes;
```

The following policy grants the role `Manager` a privilege to post new currency exchange rates (the page `postNewRates.jsp`) but only if the user updates the data from local machine.

```
grant( //priv/POST,
//app/policy/mybank/bankapp/url/currencyExchange/postNewRates.jsp,
//role/Manager) if remotehost="localhost";
```

Let us imagine a web application that allows a customer to buy stocks online. When the customer clicks on the link `mybroker/buyStocks.do`, the browser sends an HTTP request that is mapped to a Java servlet. The servlet is responsible for fetching balances of all customer's accounts and calculating the customer's purchasing power, the amount of money he or she can spend on buying new stocks. Then the servlet then sets a request attribute named `purchasingPower` and forwards the request to a page located at `mybroker/buyStocks.jsp` The `mybroker/buyStocks.jsp` shows the customer's purchasing power and asks about the amount he or she wants to spend.

The `mybroker/buyStocks.jsp` page should not be displayed if a customer's purchasing power is not positive. The following rule grants access to the page only if a customer has a positive purchasing power by checking the `purchasingPower` servlet attribute.

```
grant( //priv/GET, //app/policy/mybank/bankapp/url/mybroker/buyStocks.jsp,
//role/Client) if purchasingPower>0;
```

Again, let us imagine an application that allows a customer to trade stocks online. Before the customer can trade stocks, he or she must open a brokerage account. The account can be opened online by clicking on the `mybroker/openAccount.jsp` link. The first page that is displayed contains a trading agreement text and asks the customer to accept it. The checkbox is linked to an HTML form parameter named `customerAgreed`. When the HTML form is posted, this parameter is set to true if the customer has accepted the trading agreement.

The following rule allows customer to proceed only if he or she accepted the trading agreement by ticking the checkbox off. The rule checks the `customerAgreed HTTP` request parameter.

```
deny( //priv/POST,
//app/policy/mybank/bankapp/url/mybroker/openAccount.jsp, //role/Client)
if Not customerAgreed="true"
```

JDBC Resources

Table 4-12 shows the mapping of the resource path elements for a JDBC resource:

Table 4-12 JDBC Resource Path Elements

Element Name	Value
application name	The application name or `shared` if the resource is global
resource type	`jdbc`
resource parent	the module name (if any) + the resource type (`ConnectionPool` or `MultiPool`)
resource name	The resource name

JDBC Resource Path Example

Listing 4-3 shows the configuration of a JDBC resource.

Listing 4-3 JDBC Resource Configuration

```
<JDBCConnectionPool DriverName="oracle.jdbc.driver.OracleDriver"
                    Name="MyJDBCConnectionPool"
                    PasswordEncrypted="{3DES}B2Bl+tp70Eh3D1pT53/anw=="
                    Properties="user=wles" Targets="myserver"
                     TestTableName="SQL SELECT 1 FROM DUAL"
                     URL="jdbc:oracle:thin:@localhost:1521:ASI"/>
  <JDBCTxDataSource JNDIName="MyDataSource"
                    Name="MyJDBCDataSourceName"
                    PoolName="MyJDBCConnectionPool"
                     Targets="myserver"/>
```

Because the resource is global and does not belong to any particular module, the fully qualified resource name is:

`//app/policy/AppParentNode/shared/jdbc/ConnectionPool/MyJDBCConnectionPool`

where `ConnectionPool` is the resource type and `MyJDBCConnectionPool` is the resource name.

JDBC Resource Privilege Mappings

The privilege name of a JDBC resource is mapped to a JDBC operation name and can take one of the following values:

admin
: Privilege to perform the admin operations such as `clearStatementCache`, `suspend`, `forceSuspend`, `resume`, `shutdown`, `forceShutdown`, `start`, `getProperties`, and `poolExists`.

reserve
: Privilege to reserve a connection in the data source by looking up the data source and then calling `getConnection`.

shrink
: Privilege to shrink the number of connections in the data source.

reset
: Privilege to reset the data source connections by shutting down and re-establishing all physical database connections.

JDBC Resource Path Example

Listing 4-4 gives an example of code that uses the JDBC resource we defined earlier.

Listing 4-4 JDBC Resource Code Example

```
javax.naming.InitialContext initialContext = new
javax.naming.InitialContext();

javax.sql.DataSource ds = (javax.sql.DataSource)
initialContext.lookup("MyDataSource");

java.sql.Connection conn = ds.getConnection();

PreparedStatement statement =
conn.prepareStatement("SELECT accountName FROM accounts WHERE balance <
0");

ResultSet result = statement.executeQuery();

if (result.next()) {
    String accountName = result.getString(1);
    System.out.println("The first account with negative balance is " +
accountName);

}
```

In the example, the code calls the `getConnection()` method on the data source instance. This initiates an authorization check to verify the reserve privilege against the `//app/policy/AppParentNode/shared/jdbc/ConnectionPool/MyJDBCConnectionPool` resource.

JDBC Dynamic Resource Attributes

The following dynamic attributes are supported by JDBC resources and can be used as a part of an authorization policy:

application
The name of an application that hosts the resource.

module
The name of a module the resource belongs to.

category
The resource type (ConnectionPool | MultiPool).

resource
The name of the resource.

action
The JDBC operation name (admin | reserve | shrink | reset).

JDBC Resource Policy Examples

The following policy grants the role `ExternalApplication` a privilege to reserve (open) a JDBC connection from a connection pool called `ExternalDataPool`:

```
grant(//priv/reserve,
//app/policy/mybank/shared/jdbc/ConnectionPool/ExternalDataPool,
//role/ExternalApplication) if true;
```

The following policy grants the role `Admin` the admin privilege that will allow him or her to shut down any JDBC resources except a resource named `SystemJdbcPool`. Note that the `//app/policy/mybank/shared/jdbc` resource must be a virtual one.

```
grant(//priv/admin, //app/policy/mybank/shared/jdbc, //role/Admin) if Not
resource="SystemJdbcPool";
```

JMS Resources

Table 4-13 shows the mapping of the resource path elements for a JMS resource:

Table 4-13 JMS Resource Path Elements

Element Name	Value
application name	The application name or `shared` if the resource is global
resource type	`jms`
resource parent	The destination type (topic or queue)
resource name	The resource name

JMS Resource Path Example

Listing 4-5 gives an example of how a JMS queue named `MyJMSQueue` might be configured.

Listing 4-5 JMS Resource Configuration Example

```
<JMSServer Name="WSStoreForwardInternalJMSServermyserver"
           Store="FileStore" Targets="myserver">
  <JMSQueue CreationTime="1150241964468"
         JNDIName="JMSQueue" Name="MyJMSQueue"/>
</JMSServer>

<JMSConnectionFactory JNDIName="JmsConnectionFactory"
                      Name="MyJMSConnectionFactory" Targets="myserver"/>
```

To insure the client can use the JMS queue named `MyJMSQueue`, it should be granted rights to access resource `//app/policy/AppParentNode/shared/jms/queue/MyJMSQueue`.

JMS Resource Privilege Mappings

The privilege name of a JMS resource is mapped to the JMS operation name. It can have one of the following values:

send

Required to send a message to a queue or a topic. This includes calls to the `MessageProducer.send()`, `QueueSender.send()`, and `TopicPublisher.publish()` methods.

receive

Required to create a consumer on a queue or a topic. This includes calls to the `Session.createConsumer()`, `Session.createDurableSubscriber()`, `QueueSession.createReceiver()`, `TopicSession.createSubscriber()`, `TopicSession.createDurableSubscriber()`, `Connection.createConnectionConsumer()`, `Connection.createDurableConnectionConsumer()`, `QueueConnection.createConnectionConsumer()`, `TopicConnection.createConnectionConsumer()`, and `TopicConnection.createDurableConnectionConsumer()` methods.

browse

Required to view the messages on a queue using the `QueueBrowser` interface.

JMS Resource Example

Listing 4-6 gives an example of a JMS client that uses the JMS queue declared above.

Listing 4-6 JMS Client Example

```
//Instantiate the inital context
javax.naming.InitialContext initialContext = new
javax.naming.InitialContext();

//Look up the JMS connection factory and the message queue
Queue messageQueue = (Queue) initialContext.lookup("JMSQueue");
JMSConnectionFactory factory =
    (JMSConnectionFactory) initialContext.lookup("JmsConnectionFactory");

//Create the queue connection and session
QueueConnection queueConnection = factory.createQueueConnection();
QueueSession session =
    queueConnection.createQueueSession(false, Session.AUTO_ACKNOWLEDGE);

//Create a text message
TextMessage textMessage = session.createTextMessage();
textMessage.setText("Hello from the client!");
```

```
//Send message to the queue
QueueSender sender = session.createSender(messageQueue);
sender.send(textMessage);
```

In the example, the client sends a text message to `MyJMSQueue`. This requires the `send` privilege for resource `//app/policy/AppParentNode/shared/jms/queue/MyJMSQueue` to successfully execute the code.

JMS Dynamic Resource Attributes

The following dynamic attributes are supported by JMS resources and can be used as a part of an authorization policy:

application
The name of an application that hosts the resource.

destinationtype
The JMS destination type (queue | topic).

resource
The name of the resource.

action
The JDBC operation name (send | receive | browse).

JMS Resource Policy Examples

The following policy grants the role `Client` a privilege to send messages to a JMS queue named `FeedbackQueue`:

```
grant(//priv/send, //app/policy/mybank/shared/jms/queue/FeedbackQueue,
//role/Client) if true;
```

The following policy grants the user `FeedbackProcessor` a privilege to receive messages from a JMS queue named `FeedbackQueue`:

```
grant(//priv/receive, //app/policy/mybank/shared/jms/queue/FeedbackQueue,
//user/myusers/FeedbackProcessor);
```

Web Services Resources

Table 4-14 shows the mapping of the resource path elements for a Web Services resource:

Table 4-14 Web Services Resource Path Elements

Element Name	Value
application name	The name of the application that contains the resource
resource type	`webservices`
resource parent	The application context
resource name	The Web Service name

Web Services Resource Path Example

Listing 4-7 shows the configuration of a web application named `BasicWS` that contains a Web Service implementation named `BasicWS_Component`.

Listing 4-7 Web Application Configuration Example

```
<application Name="BasicWS"
             Path="applications/BasicWS.ear"
             StagedTargets="myserver"
  <WebServiceComponent Name="BasicWS_Component"
                       Targets="myserver"
URI="BasicWS.war"/>
</application>
```

Listing 4-8 shows how the `application.xml` file from the `BasicWS.ear` enterprise archive defines the web application context:

Listing 4-8 Web App Context Example

```
<module>
  <web>
    <web-uri>basic_javaclass.war</web-uri>
    <context-root>myservices</context-root>
```

```
  </web>
</module>
```

Listing 4-9 shows the configuration of a Web Service named `HelloWorld`, which is defined in the descriptor `web-services.xml` inside the WAR file.

Listing 4-9 Web Service Example

```
<web-services>
  <web-service useSOAP12="false"
               name="HelloWorld"
               style="rpc"
               uri="/HelloWorld">
    <operations>
        <operation name="sayHello"
                   method="sayHello(int,java.lang.String)"/>
    </operations>
  </web-service>
</web-services>
```

The fully qualified name of this Web Service resource is:

```
//app/policy/AppParentNode/BasicWS/webservices/myservices/HelloWorld
```

Web Services Resource Privilege Mappings

The privilege for accessing a Web Service is mapped to the name of the Web Service operation.

Web Services Resource Policy Examples

To call the operation `sayHello` in the HelloWorld service defined in this section, the client must be granted the privilege `sayHello`. Note that some of the clients may require access to the WSDL file, which is actually a URL resource. Consider the following client code:

```
String wsdlUrl = "http://localhost:7001//HelloWorld?WSDL";
HelloWorld service = new HelloWorld_Impl(wsdlUrl);
HelloWorldPort port = service.getHelloWorldPort();
String result = port.sayHello(34, "Josh");
```

Before calling method `sayHello`, the client accesses the WSDL file. To make the code run successfully, the client must be granted the privilege GET on the resource

`//app/policy/AppParentNode/BasicWS/url/myservices/helloworld` in addition to the Web Service resource. Thus, the following policies must be created:

```
grant(//priv/GET,
//app/policy/AppParentNode/BasicWS/url/myservices/helloworld,
//role/SomeUser) if true;
```

```
grant(//priv/sayHello,
//app/policy/AppParentNode/BasicWS/webservices/myservices/HelloWorld,
//role/SomeUser) if true;
```

Note that for the URL Resource, the resource name was changed to lower case.

Web Services Dynamic Resource Attributes

The following dynamic attributes are supported by Web Services resources and can be used as a part of an authorization policy:

application
The name of the application

contextpath
The context part of the web application

webservice
The name of the web service

method
The name of the web service operation called

Param<N>
A value of the Nth parameter in the method, for example,. Param1, Param2…

Web Services Resource Policy Examples

The following policy grants the role `Client` a privilege to call the operation `getDelayedQuote` on a Web Service named `StockQuoteService`:

```
grant(//priv/getDelayedQuote,
//app/policy/mybank/myservices/webservices/publishedservices/StockQuoteSer
vice, //role/Client) if true;
```

The following policy grants the role `Client` a privilege to call the operation `getRealtimeQuote` on a Web Service named `StockQuoteService`, but only if he or she has a premium subscription type:

```
grant(//priv/getRealtimeQuote,
//app/policy/mybank/myservices/webservices/publishedservices/StockQuoteSer
vice, //role/Client) if subscriptionType="premium";
```

Server Resources

A Server resource determines who can control the state of a WebLogic Server instance. When users start server instances by invoking the `weblogic.Server` class in a Java command, the policy on the Server resource is the only security check that occurs. You can create security policies that apply to all WebLogic Server instances in a domain or to individual servers.

The following table shows the mapping of the resource path elements for a Server resource:

Table 4-15 Server Resource Path Elements

Element Name	Value
application name	`shared`
resource type	`svr`
resource parent	Not used
resource name	The server instance name

Server Resource Path Example

Listing 4-10 gives an example of the configuration of a WebLogic Server instance named `myserver`.

Listing 4-10 WebLogic Server Instance Configuration

```
<Server ListenAddress=""
        ListenPort="7001"
        Machine="mymachine"
        Name="myserver"
        NativeIOEnabled="true"
        ReliableDeliveryPolicy="RMDefaultPolicy"
        ServerVersion="8.1.5.0">
    <SSL Enabled="false" HostnameVerificationIgnored="false"
```

```
        IdentityAndTrustLocations="KeyStores" Name="myserver"/>
</Server>
```

The fully qualified name of the resource that corresponds to the server instance is `//app/policy/AppParentNode/shared/svr/myserver`.

Server Resource Privileges Mapping

The privilege name of a Server resource is mapped to the operation name. It can have one of the following values:

boot
: Privilege required to start a WebLogic Server instance, either an Administration Server or Managed Server.

shutdown
: Privilege required to shut down a running WebLogic Server instance, either an Administration Server or Managed Server.

suspend
: Privilege required to prohibit additional logins (logins other than for privileged administrative actions) to a running WebLogic Server instance, either an Administration Server or Managed Server.

resume
: Privilege required to re-enable non-privileged logins to a running WebLogic Server instance, either an Administration Server or Managed Server.

Server Dynamic Resource Attributes

The following dynamic attributes are supported by Server resources and can be used as a part of an authorization policy:

server
: the name of the server the resource is associated with.

action
: the name of an operation performed on the server instance (boot | shutdown | suspend | resume).

Server Resource Policy Examples

The following policy grants the role `Admin` a privilege to boot all WebLogic Server instances. Note that the resource `//app/policy/mybank/shared/svr` must be a virtual one.

```
grant(//priv/boot, //app/policy/mybank/shared/svr, //role/Admin) if true;
```

The following policy grants the role `Admin` a privilege to shutdown or suspend a WebLogic Server instance named `CentralServer` but only on Sundays or other days between 2 a.m. and 4 a.m.:

```
grant[//priv/shutdown,//priv/suspend],
//app/policy/mybank/shared/svr/CentralServer, //role/Admin) if timeofday in
[2:0:0..4:0:0] Or dayofweek=Sunday;
```

Subject Mapping

All authorization policies in ALES are applied considering a subject that accesses a resource. A subject representation uses the standard `javax.security.auth.Subject` class that contains a set of `java.security.Principal` objects. The subject consists of a directory name, a user name, and a set of group names. The user and groups are considered to exist within the specified directory. The directory name is a part of configuration of the authentication and role-mapping providers, and can be modified using the ALES Administration Console.

The providers iterate the principals, selecting those that implement the `weblogic.security.spi.WLSUser` and `weblogic.security.spi.WLSGroup` interfaces. The first `WLSUser` principal is used to retrieve the user name. All of the `WLSGroup` principals are used to build of the group names.

Note that running an application under the ALES framework does not require any changes on the client or server side in terms of credential handling. The actual methods of supplying credentials depend on the resource type. For example, to access a URL resource, the user can supply its credentials in the browser's prompt dialog or, if the client is Java code, it can send the credentials as the HTTP Authorization header of the request. Listing 4-11 shows an example how the credentials can be supplied using standard methods before accessing enterprise resources:

Listing 4-11 Supplying Credentials

```
Hashtable properties = new Hashtable();
properties.put(InitialContext.PROVIDER_URL, "t3://localhost:7001");
properties.put(InitialContext.SECURITY_PRINCIPAL, "username");
properties.put(InitialContext.SECURITY_CREDENTIALS, "password");
properties.put(InitialContext.INITIAL_CONTEXT_FACTORY,
    "weblogic.jndi.WLInitialContextFactory");
```

```
javax.naming.InitialContext initialContext =
    new javax.naming.InitialContext(properties);

//Look up and access the resources here...
```

Policy Element Naming

The policy language uses standard naming conventions called qualifiers to refer to privileges, applications, resources, roles, and identity elements (directories, users and groups). These conventions ensure that each component has a unique name, even if you use the same name in other locations. The Administration Console hides these qualifiers from you during most operations. See "Fully Qualified Names" on page 4-55 for additional information on naming conventions.

The following rules apply to policy element names:

- Most names are case sensitive. Declarations and attribute names are the exception; they are case insensitive. Internally, when a declaration name or attribute name is saved, it is saved in all lowercase. For example, the user names `//user/ales/system/` and `//user/ales/System/` reflect the same user.
- A qualified name is a name with qualifier prefix prepended to the non-qualified name. Some names, like user and group names, have an ending suffix also. See Table 4-16 for examples. Declaration names do not have a qualified form.
- The characters used for the names and the length of the names are restricted. See "Size Restriction on Policy Data" on page 4-56.

Table 4-16 Examples of Qualified Names

Policy Element	Example
resource	`//app/policy/banking/transfer`
directory	`//dir/extranet`
privilege	`//priv/place_order`
privilege group	`//grp/trading_privileges`
user	`//user/extranet/JohnDoe/`
group	`//sgrp/extranet/trader/`

Table 4-16 Examples of Qualified Names (Continued)

Policy Element	Example
role	`//role/roleName`
logical name	`//ln/ShortHandForResource`

For more information on policy element naming, see the following topics:

- "Fully Qualified Names" on page 4-55
- "Policy Element Qualifiers" on page 4-56
- "Size Restriction on Policy Data" on page 4-56
- "Character Restrictions in Policy Data" on page 4-58
- "Special Names and Abbreviations" on page 4-64

Fully Qualified Names

A fully qualified name references the full name for a policy element. This name consists of a series of simple names separated by forward slashes (/). Fully qualified names have the following parts, in order:

- A starting double forward slash: //
- A qualifier followed by a forward slash
- For users and groups, a directory name followed by a slash mark, and a final slash after the name
- A name:

For example, in `//user/Accounting/JJBob/`

- `user` is the qualifier
- `Accounting` is the directory
- `JJBob` is the user name

For resources, the qualified name starts with `//app/policy/`. Additional names may appear, each separated by a single slash. This naming convention defines the resource tree. Each resource

name is represented as a node on the tree, but the entire string represents the fully qualified name of the final resource. For example:

```
//app/policy/trading_system/PersonalTrades/BondOrder/Order
```

Policy Element Qualifiers

Qualifiers are built in. You cannot create your own qualifier or change the existing ones. They represent one of the basic policy elements and always begin with a double slash (//) followed by a single slash (/). Table 4-17 lists the built-in qualifiers.

Table 4-17 Policy Element Qualifiers

Qualifier	Policy Element
`//priv`	privilege
`//grp`	privilege group
`//user`	user
`//sgrp`	group
`//app`	resource
`//dir`	directory
`//bind`	engine
`//role`	role
`//ln`	logical name

There is no qualifier for a declaration. Declarations are identified by a different method. For a discussion of declarations, see "Declaration Names" on page 4-63.

Size Restriction on Policy Data

There are some limits on the size of names, attribute values, and rules, restricted by the type of database that you use. The restriction by the database is determined by the `VARCHAR` column size and the key size allowed by the database. Table 4-18 summarizes the limit for different policy data for the Oracle and Sybase databases.

Note: As of ALES version 2.5, additional databases are also supported. See Installing the Administration Server for additional information.

Table 4-18 Database Restrictions on Policy Data

Policy Data	Oracle	Sybase 12.5 2K[1]	Sybase 12.5 4K	Sybase 12.5 8K	Sybase 12.5 16K
Qualified privilege name Qualified privilege group name Qualified role name Qualified resource name Qualified user name Qualified subject group name Qualified logical name Qualified security provider name	2000	580	1200	2500	5000
All privileges in the privilege field of a rule All roles in the privilege field of a rule All resources in the object field of a rule All user and group in subject field of a rule All roles in subject field of a rule	2000	580	1200	2500	5000
Rule conditions	4000	1160	2400	5000	10000
Rule text-combined text of all fields in a rule (privilege, object, subject, delegator, and conditions, plus the syntax delimiters)	N/A[2]	1962	4010	8106	16298
Declaration name Attribute name The individual declaration name inside a type declaration	2000	580	1200	2500	4000
Declaration text-the combined text of declaration name, kind, and value, plus the syntax delimiters	4000	1160	2400	5000	10000
A single attribute value	2000	580	1200	2500	4000

Table 4-18 Database Restrictions on Policy Data (Continued)

Policy Data	Oracle	Sybase 12.5 2K[1]	Sybase 12.5 4K	Sybase 12.5 8K	Sybase 12.5 16K
A quoted literal string in the declaration value	4000	1160	2400	4000	4000
A quoted literal string in a constraint for a rule	4000	1160	2400	4000	4000
A qualified name in the declaration value	2000	580	1200	2500	4000
A qualified name in a constraint for a rule	2000	580	1200	2500	4000
Integer value of a constant declaration	9 digits*[3]	9 digits*	9 digits*	9 digits*	9 digits*
All attribute values combined for a user, group or resource attribute	40000	40000	40000	40000	40000

1. Sybase 12.5 has a dependency on the logical page size that you choose when you set up the database server. The supported logical page size varies from 2K, 4K, 8K, and 16K.
2. N/A means that there is no limit.
3. An asterisk (*) indicates that the limit is imposed.

Character Restrictions in Policy Data

There are several restrictions on the character set that you can use to define policy data. The following common rules apply and Table 4-19 describes the extended character set restrictions.

- All names without qualifier or called non-qualified names allow alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_). These names include privilege group name, resource name, directory name, security provider name, declaration, and attribute name.
- The role name, user name, and subject group name can be multi-byte values.
- All names, except privilege, user, and subject group name, must start with an alpha character or underscore. Numeric characters are not allowed.

Table 4-19 Policy Data Character Restrictions

Policy data	Extra characters allowed
Privilege Name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
Privilege Group Name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
Resource name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore. Pound sign (#), apostrophe (`), dash (-), period (.), colon (:), at (@), tilde (~), ampersand (&).
Directory Name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
Security Provider Name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
Declaration	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
Attribute Name	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
User name or Subject group name	All printable characters. A forward slash (/) in the name must be escaped by a backward slash (\), because a forward slash (/) is used as field separator.
Role	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
ARME or SCM	Alphanumeric characters (a-z, A-Z, and 0-9) and the underscore (_), must start with an alpha character or underscore.
String typed attribute value	All printable characters are allowed.

Table 4-19 Policy Data Character Restrictions (Continued)

Policy data	Extra characters allowed
Literal string in the value of a constant declaration	All printable characters are allowed except the double quote (") and a backslash (\). When used, these characters may cause parsing problems.
Literal string in a condition for a rule	All printable characters are allowed except the double quote (").

The following topics provide more information:

- "Data Normalization" on page 4-60
- "Directory Names" on page 4-63
- "Logical Name" on page 4-63
- "Declaration Names" on page 4-63

Data Normalization

When using the ASI Authorization or ASI Role Mapping providers, there are certain data transformations that you must consider. The policy database limits what characters are allowed in certain policy elements. This set is more restrictive than the set allowed by the Security Framework.

The ASI Authorization and ASI Role Mapping providers perform normalization of input data to ensure that they abide by the restrictions imposed by the authorization management system. The management system does not currently perform any automatic normalization, so it is important to understand the normalization mechanism because it must be preformed manually when writing policy. Unless otherwise stated, the substitutions listed Table 4-20 apply to the following elements: resource, attribute, privilege, role, and directory names.

Additionally, any nonprintable character is translated into the numeric hexadecimal equivalent; for example, the ASCII character code 1 (a smiley face) is represented as __0x1_. Table 4-20 shows the characters that are normalized and the character substitution applied at runtime. When writing policy, you must substitute these characters.

Table 4-20 Character Substitution

Character	Character Substitution
\n	(carriage return) __CR_ also applies to user and group names
0	__0_ 1st character only
1	__1_ 1st character only
2	__2_ 1st character only
3	__3_ 1st character only
4	__4_ 1st character only
5	__5_ 1st character only
6	__6_ 1st character only
7	__7_ 1st character only
8	__8_ 1st character only
9	__9_ 1st character only
\t	(tab) __TAB_
‘ ‘	(space) __SP_
!	__EXPL_
“	__DQUOT_
#	__HASH_ 1st character of resource, or any character in attr, priv, role, dir
.	__PRD_ 1st character of resource, or any character in attr, priv, role, dir
%	__PRCT_
(	__OPRN_
)	__CPRN_
*	__ASTR_

Table 4-20 Character Substitution (Continued)

Character	Character Substitution
+	__PLUS_
,	__COMMA_
/	__FSLSH_
;	__SCLN_
<	__LT_
=	__EQ_
>	__GT_
?	__QTM_
[	__OSQB_
\	__BSLSH_
]	__CSQB_
‘	__CSQUOT_
{	__OCRL_
|	__PIPE_
}	__CCRL_
&	__AMP_ Applies only to attr, priv, role, dir
-	__DASH_ Applies only to attr, priv, role, dir
:	__CLN_ Applies only to attr, priv, role, dir
@	__AT_ Applies only to attr, priv, role, dir
~	__TLD_ Applies only to attr, priv, role, dir

Directory Names

A directory further separates qualifiers. You define directories to store and scope users and groups. For example, if you had an application called `Bankers`, the directory that stores users and groups might look like this:

```
//dir/Bankers
```

Once declared, the directory is used with the user and group qualifier to fully qualify subjects. For example, `//sgrp/Bankers/loans/` is a group called `loans` that belongs to the `Bankers` group and `//user/Bankers/BSilva/` is a user named `BSilva` that belongs to the `Bankers` application.

Note: A directory name must start with a letter and can contain any number of alphanumeric or underscore characters. Spaces are not allowed in the name. Directory names are not case sensitive.

A directory name does not necessarily need to represent a resource. For example, a directory name might represent users in a particular location (as in `//dir/NewYork`) or a department (as in `//dir/Accounting`). Essentially, you can use them any way you want to delineate groups of users and groups.

A privilege group is not part of the policy language but is provided for administrative convenience. Each privilege in a group is defined as an individual privilege in the actual policy.

Logical Name

A logical name is a shorthand method used to represent a resource. Once you map a logical name to a fully qualified name, your developers can use the logical name when coding your application.

```
//ln/name
```

Declaration Names

A declaration name is not qualified. In fact, that is exactly how they are identified. Any policy element without a fully qualified name and not in quotation marks (indicating a string), is assumed to be a declaration. When defined, declarations are preceded by one of the following identifiers:

`const` – Constant Declaration

`type` – Type Declaration

`cred` – Credential (or Attribute) Declaration

`eval` – Evaluation Function Declaration

Special Names and Abbreviations

There are several special names, referred to as keywords, that are shortcuts for denoting groups of objects. The keywords keep you from having multiple rules or multiple rule queries in certain reoccurring situations. By using these keywords, you can define very powerful, yet generic rules. The keywords are as follows:

- any—Signifies any privilege. When specifying `any` in a rule, it means you do not care what privilege a user invokes when applying the rule.
- ALL—Signifies a privilege group containing all privileges. You must use the `grp` qualifier with the keyword `ALL` (`//grp/ALL`). The keyword `ALL` is mainly used for grouping purpose in the console and, by default, every privilege defined belongs to the privilege group `ALL`.
- allusers—For each user directory, there is an implied group called `allusers`. This group refers to all users in a directory. For example:

  ```
  //group/Acct/allusers
  ```

  This example refers to `allusers` for the `Acct` directory and eliminates the need to individually address each user or to create a named group for all of the users.

  Note: The keyword `allusers` is only a limited pseudo group. It does not have many of the qualities of a regular group; you cannot map it to anything, you cannot add or remove members, and it cannot be a member of group hierarchy. You can delegate to `allusers` groups.

Table 4-21 describes the rules for using keywords.

Table 4-21 Rules for Using Keywords

Characteristic	any	ALL	allusers
Policy Element	Built-in privilege	Built-in privilege group	Built-in local group
Represents	any privilege	All privileges including built-in and user-defined	All users in one local directory
Used in rules	Yes	No	Yes
Needs qualifier	No	Yes `//grp/ALL`	Yes `//sgrp/ [directory name]/allusers`

Table 4-21 Rules for Using Keywords (Continued)

Characteristic	any	ALL	allusers
Used in policy queries	Yes Only finds rules with the literal any. That is, it does not return all rules (rules with any privilege).	No	Yes Finds rules that specifically entitle the group allusers
Controlled by delegation	No Must be in a group that is delegated	Yes Controls access to all privileges regardless of privilege group	Yes You must be specifically delegated access to this group
Case-sensitive	Yes	Yes	Yes

Sample Policy Files

A policy file is a text file that lists the relevant policy elements using their fully qualified names. The ALES Administration Server installation includes sets of sample policies for BEA WebLogic Portal, BEA AquaLogic Data Services Platform, and BEA AquaLogic Service Bus. You can import these sample policies and use them as a starting point for developing a full set of policies for your applications. For information about how to import the sample policies, see the README files in each of the sample directories and see also "Importing Policy Data" on page 7-1. Table 4-22 shows the location of the samples.

Table 4-22 Sample Policy Files

BEA Product	Sample Policy Directory
WebLogic Portal	`BEA_HOME/ales26-admin/examples/policy/portal_sample_policy`
AquaLogic Data Services Platform	`BEA_HOME/ales26-admin/examples/policy/aldsp_sample_policy`
AquaLogic Service Bus	`BEA_HOME/ales26-admin/examples/policy/alsb_sample_policy`

In addition, this section provides examples of policy files. Sample files for each policy element are provided with the product and are installed in the following directory:

```
BEA_HOME\ales26-admin\examples\policy
```

For a description of each of these files, see the following topics. The policy data filenames are shown in brackets ("[]").

Application Bindings [binding]

This file contains an example of the Authorization provider and Service Control Manager bindings. The resources that can be bound are the resources that are created as binding nodes.

Each line contains a name of an Authorization provider or Service Control Manager, followed by a binding node name. A Security Provider can only bind policy resources and the Service Control Manager can only bind configuration resources.

Examples:

```
//bind/myAuthorizationProvider  //app/policy/myApplication/myBinding
```

```
//bind/mySCM //app/config/myConfiguration/configBind
```

Attribute [attr]

This file lists the subject attribute for users and subject group. The attribute value property must comply with user attribute schema defined for `//dir/dirName`. If the property is "L", the attribute value must be enclosed in brackets ([]), with items separated by commas. In general, the attribute value for all users must be set according to the specification defined in user attribute schema. However, if an attribute is not set when this file is created, its record may be left out in this file.

Note: Both user and credential declarations must exist in the policy database before it can be loaded successfully. Further, the user attribute schema must be defined before the user attribute can be assigned in Attribute Value file.

Examples:

Given the user attribute schema shown in Listing 4-12, the user attribute values and subject attribute value are defined as shown in Listing 4-13 and Listing 4-14.

Listing 4-12 User Attribute Schema

```
//dir/CA_Office my_host_ip S

//dir/CA_Office my_favorite_color L [blue,green]

//dir/NY_Office email_address S "user@crosslogix.com"

//dir/NY_Office my_birthday   S

//dir/NY_Office my_favorite_color L [red]
```

Listing 4-13 Sample User Attributes

```
//user/CA_Office/user_a@mycom.com/ my_host_ip 121.1.100.25
//user/CA_Office/user_b@mycom.com/ my_host_ip 121.1.100.26
//user/CA_Office/user_c@mycom.com/ my_host_ip 121.1.100.50
//user/CA_Office/user_d@mycom.com/ my_host_ip 121.1.100.225
//user/CA_Office/user_e@mycom.com/ my_host_ip 132.99.25.77

//user/CA_Office/user_a@mycom.com/ my_favorite_color [red]
//user/CA_Office/user_b@mycom.com/ my_favorite_color [white,green]
//user/CA_Office/user_c@mycom.com/ my_favorite_color [red,blue]

//user/NY_Office/user_1/ email_address     "user1@crosslogix.com"
//user/NY_Office/user_1/ my_birthday       1/1/1960
//user/NY_Office/user_1/ my_favorite_color [blue]
```

Listing 4-14 Sample Subject Group Attribute

```
//sgrp/NY_Office/role1/ my_favorite_color [green]
```

Declarations [dec]

AquaLogic Enterprise Security supports four kinds of declarations that are used in rules, user attributes, and resource attributes. You must create the declaration before you use it in a rule. The kinds of declarations are: enumerated types (ENUM), constants (CONST), attributes (CRED), and evaluation functions (EVAL). You can use this file to declare each one. Each line contains the declaration text, starting with declaration type. Declaration names are case-insensitive and are always saved in lower case. The four kinds of declaration text must conform with the following syntax.

```
ENUM enum_name = (enum1, enum2, ..., enumn);

CONST constant_name_1 = constValue;

CONST constant_name_2 = [value1, value2, ..., valuen];CRED cred_name :
datatype;

EVAL eval_name;
```

Examples:

```
ENUM color_type = (red, blue, green, white);

CONST my_favorite_color = green;

CONST my_birth_date = 07/04/1980;

CONST favorite_colors_for_tom = [blue, white];

CONST colors_of_my_choice = [my_favorite_color, red];

CONST a_few_cities = ["New York", "Boston", "San Francisco"];

CONST a_magic_number = 28;

CRED string_cred_1 : string;

CRED color_cred : color_type;

CRED date_cred : date;

CRED weight_in_pound : integer;

EVAL is_good_number;
```

Directories [dir]

Multiple directories can be used to separate users and groups that come from different user stores. A directory is also associated with a schema and the types of attributes the users in that directory contains.

This file lists the name of some sample directories. The directory name must start with the prefix:

```
//dir/
```

Examples:

```
//dir/CompanyA

//dir/CompanyB
```

Directory Attribute Schemas [schema]

A directory defines all users and user groups. Before a user or a user group can be assigned an attribute, you must declare the directory to accept their attributes. You can use this file to declare the attributes that a directory can have.

Each line in the file contains a directory name, an attribute name (the attribute declaration as in file "decl"), a value type (single- or multi-value), and an optional template value matching the

data type of the attribute. The single-value type is denoted by S and multi-value type by L (from list-value).

You must enter a multi-value (list) attribute with all values enclosed in square brackets [] and separated by commas, and enclose each value for a string data typed attribute with double quotes ("). You cannot use another double quote (") and backslash (\) in the template value.

Examples:

```
//dir/CompanyA my_host_ip S 111.111.111.111

//dir/CompanyA my_favorite_color S

//dir/CompanyA email_address L ["user@bea.com", "xyz@yahoo.com"]

//dir/CompanyB my_birthday   S

//dir/CompanyB my_favorite_color L [blue,green]
```

list of exclusive sgrp pair.

Mutually Exclusive Subject Groups [excl]

This file lists the subject groups that are mutually exclusive from one another. An exclusive subject groups record has the following format:

```
//sgrp/dirName/aSubjectGroupName/ //sgrp/dirName/anotherSubjectGroupName
```

For subject groups to be mutually exclusive, they must comply with the following requirements:

- Both subject groups must be in the same directory.
- The subject groups must not share a common sgrp member or user member.
- Both subject groups in a pair must exist in the policy database before the pair can be defined and loaded successfully.

Example:

```
//sgrp/CA_Office/trader/    //sgrp/CA_Office/salesPerson/
```

Resources [object]

In general, resources are constructed as a tree below two tree roots: the policy resources tree and the configuration tree. The policy tree has a resource name that starts with the prefix `//app/policy/` (for resource configuration) and configuration tree that starts with the prefix

`//app/config/` (for provider configuration). However, you do not see the provider configuration in the tree. This file lists all the resource names in order, from the root to the child nodes, together with the resource type and the logical name for the resource.

There is a special resource type, denoted by **A**, indicating that the resource node is bound by an ASI Authorization Provider or a Service Control Manager. This special resource node is called a binding node. All other resources are denoted by **O** and are called non-binding nodes.

A logical name or alias is a short name for a resource and can be optionally associated with a resource. Only binding nodes derived from the resource can have an alias. A logical name used as an alias must start with prefix:

```
//ln/
```

and must be unique to the entire resource tree. Each line contains a resource name, an optional resource type, and an optional alias. If the resource type is missing, it defaults to **O**. If there is an alias, the resource type must be specified.

Examples:

```
//app/policy/myApplication

//app/policy/myApplication/myBinding A

//app/policy/myApplication/myBinding/myresource.one O //ln/myres1

//app/policy/myApplication/myBinding/myresource.two O

//app/policy/myApplication/myBinding/myresource.three


//app/config/myConfiguration O

//app/config/myConfiguration/configBind A //ln/configBind
```

Resource Attributes [object]

Because a resource is also referred to as object, a resource attribute is also referred to as an object attribute. Each line contains a resource name (as in file "object"), an attribute name (the declaration as in file "decl"), a value type (single- or multi-value), and values matching the data type of the attribute. The single-value type is denoted by **S** and multi-value type is by **L** (from list-value). You can enter a multi-value attribute either in multiple lines, with the same resource name, attribute name and value type (**L**); or, you can enter it using one line, with all the values enclosed in square brackets [] and separated by commas. You must enclose each value for a string attribute with double quotes ("). You cannot use another double quote and backslash (\) in the attribute value.

Examples:

```
//app/policy/myApplication/myBinding  string_attr_1 S  "A value to be decided"

//app/policy/myApplication/myBinding/myresource.one string_attr_1  L  "1st
Value"

//app/policy/myApplication/myBinding/myresource.one string_attr_1  L  "2nd
Value"

//app/policy/myApplication/myBinding/myresource.one string_attr_1  L  "3rd
Value"

//app/policy/myApplication/myBinding/myresource.two string_attr_1  L ["ABC",
"DEF", "XYZ"]

//app/policy/myApplication/myBinding/myresource.three color_attr_1 L [red,
blue]

//app/policy/myApplication/myBinding/myresource.three integer_attr_1 S  1001

//app/policy/myApplication/myBinding/myresource.three date_attr_1 L
[01/01/2003, 01/01/2004]
```

Policy Distribution [distribution]

This file provides the parameters used for policy distribution issued by the Policy Import tool when the distribution is enabled in a configuration. The policy distributor takes a list of user directories and distribution point combinations. Therefore, each line contains a directory and a distribution point separated by white spaces.

The distribution point is a resource node on or above a binding resource node. The directory can be either a specific directory or `//dir/*` to include all user directories.

Note: You cannot use applications pending deletion as distribution points. Select a node higher in the tree as the distribution point.

Examples:

```
//dir/* //app/policy/myApplication
```

```
//dir/CompanyA //app/policy/myApplication/myBinding
```

Policy Inquiry [piquery]

AquaLogic Enterprise Security stores the contents of a policy inquiry in the policy database. This file contains examples of policy inquiries to import and store in the policy database. Each query can span multiple lines, can have multiple lines of each type, but must have a minimum of one line. The first line of each query must specify the privilege, the effect (grant or deny), the query owner and the query title.

Each line has the following syntax:

```
P/O/S oneQualifiedName grant/deny queryOwner queryTitleMayhaveSpace
```

where `P/O/S` stands for `privilege`, `object` (resource), and `subject`.

Listing 4-15 shows policy inquiry examples.

Listing 4-15 Policy Inquiry Examples

```
# Sample query 1:

P //priv/delete     grant //user/ales/system/ Saved Policy Inquiry #1
O //app/policy/myApplication/myBinding  grant //user/ales/system/
      Saved Policy Inquiry #1
S //ales/ales/userid/ grant //user/ales/system/ Saved Policy Inquiry #1

# Sample query 2 (same content as query 1):

P //priv/delete     grant //user/ales/system/ Policy Inquiry #2
O //app/policy/myApplication/myBinding
S //ales/ales/userid/

# Sample query 3:

P //priv/delete   grant //user/ales/system/ Policy Inquiry #3

# Sample query 4:
```

```
P //priv/delete                deny //user/ales/system/ PIQuery4
P //priv/create
O //app/policy/myApplication
```

Policy Verification [pvquery]

AquaLogic Enterprise Security stores the contents of a policy verification in the policy database. This file defines policy verification queries to import and store in the database. Each query spans multiple lines. The first line of each query must have the owner and title, in the following syntax:

```
LP/RO/RP/RO oneQualifiedName queryOwner queryname
```

A query name may contain spaces.

Listing 4-16 shows policy verification examples:

Listing 4-16 Policy Verification Examples

```
# Sample query 1:

LP //priv/delete //user/ales/system/ Policy Verification #1
LO //app/policy/myApp/firstResource //user/ales/system/ Policy Verification #1
RP //priv/create //user/ales/system/ Policy Verification #1
RO //app/policy/myApp/secondResource //user/ales/system/ Policy Verification #1

# Sample query 2 (query content is the same as query 1):

LP //priv/delete //user/ales/system/ Policy Verification #2
LO //app/policy/myApp/firstResource
RP //priv/create
RO //app/policy/myApp/secondResource

# Sample query 3:

LP * //user/ales/system/ Policy Verification #3
LO //app/policy/myApp/firstResource //user/ales/system/ Policy Verification #3
RP //priv/delete //user/ales/system/ Policy Verification #3
RO //app/policy/myApp/secondResource //user/ales/system/ Policy Verification #3

# Sample query 4:

LP * //user/ales/system/  PolicyVerification#4
LO //app/policy/myApp/firstResource //user/ales/system/  PolicyVerification#4
RP * //user/ales/system/  PolicyVerification#4
RO //app/policy/myApp/secondResource //user/ales/system/  PolicyVerification#4
```

Privileges [priv]

This file contains a sample list of privilege names. Each privilege name must start with the prefix:

```
//priv/
```

Examples:

```
//priv/read
//priv/Read
//priv/search_file
//priv/search_text
```

Privilege Bindings [privbinding]

This file contains examples of how privileges are bound to privilege groups. Each line contains a privilege group followed by a privilege.

Examples:

```
//grp/myPrivGroup //priv/read
//grp/myPrivGroup //priv/search_file
//grp/myPrivGroup //priv/search_text
//grp/DevelopmentGroup //priv/read
//grp/DevelopmentGroup //priv/Read
```

Privilege Groups [privgrp]

This file contains examples of privilege group names. Each privilege group name must start with the prefix:

```
//grp/
```

Examples:

```
//grp/myPrivGroup
//grp/DevelopmentGroup
```

Role [role]

This file defines a list of role names. Roles are used to construct policies. Each line contains a role name. Each role name is prefixed with:

```
//role/
```

Examples:

```
//role/manager
//role/QA
//role/trading_Manager
//role/salesEngineer
//role/junior_trader
//role/salesPerson
//role/trader
```

Rule [rule]

Rules are used by the ASI Authorizer to make authorization and role mapping decisions. This file lists rules with their rule text conforming to rule syntax. Each line contains one rule, a grant, deny, or delegate rule. Sample entries assume all of the referenced roles, privileges, resources, users, groups and declarations exist in the policy database.

Examples:

```
grant(//role/Administrators, //app/policy/myApplication,
//user/ales/system/);
grant(//priv/read, //app/policy/myApplication, //sgrp/ales/allusers/);
deny([//priv/read, //priv/search_text],
//app/policy/myApplication/myBinding/confidentialDocument.one,
//role/public);
delegate(//role/Administrators, //app/policy/myApplication,
//user/ales/John Doe/, //user/ales/system/) if dayofweek in weekend;
```

Distribution Targets

There are two types of distribution targets in BEA AquaLogic Enterprise Security:

- The Authorization and Role Mapping providers that enforce policy
- The Service Control Manager that manages configuration changes

Both of these targets retrieve their policy data from the policy distributor. The security providers receive only policy related changes and the Service Control Manager retrieves only configuration related changes. The file called `engine` lists the names of the security providers and the Service Control Manager and respective type.

The name is qualified by the prefix:

```
//bind/
```

The names are referred to by the application binding file (`binding`) and must be imported before the application binding file.

Examples:

```
//bind/mySCM SCM
```

Subject Group Membership [member]

This file lists subject group membership. Each record has one of the following formats:

```
//sgrp/dirName/aSubjectGroupName/ //sgrp/dirName/aSubjectGroupMemberName/
//sgrp/dirName/aSubjectGroupName/ //user/dirName/aUserMemberName/
```

When you define subject group memberships, the subject group and members must comply with the following requirements:

- The subject group and the member must be in the same directory.
- One user may belong to many subject groups.
- One subject group may be a member of many subject groups.
- Two subject groups that have common members cannot become mutually exclusive.
- Both subject groups and their members must exist in the policy database before the membership can be loaded successfully.

For an example of a Member policy file, see Listing 4-17.

Listing 4-17 Sample Member Policy File

```
//sgrp/CA_Office/junior_trader/   //sgrp/CA_Office/trader/
//sgrp/CA_Office/trader/          //sgrp/CA_Office/senior trader/
//sgrp/CA_Office/senior trader/   //sgrp/CA_Office/trading_Manager/
//sgrp/CA_Office/salesEngineer/   //sgrp/CA_Office/salesManager/
//sgrp/CA_Office/salesPerson/     //sgrp/CA_Office/salesManager/

//sgrp/CA_Office/junior_trader/   //user/CA_Office/user_a@mycom.com/
//sgrp/CA_Office/senior trader/   //user/CA_Office/user_b@mycom.com/
//sgrp/CA_Office/trading_Manager/ //user/CA_Office/user_c@mycom.com/
//sgrp/CA_Office/salesPerson/     //user/CA_Office/user_d@mycom.com/
//sgrp/CA_Office/customer/        //user/CA_Office/user_e@mycom.com/
```

Subjects [subject]

This file contains a list of users and subject groups. Each record must have one of the following formats:

```
//user/dirName/aUserName/
//sgrp/dirName/aSubjectGroupName/
```

The directory name must be formatted as `//dir/dirName` and it must exist in the policy database before its subjects can be loaded successfully.

For an example of a Subjects policy file, see Listing 4-18.

Listing 4-18 Sample Subjects Policy File

```
//user/CA_Office/user_a@mycom.com/
//user/CA_Office/user_b@mycom.com/
//user/CA_Office/user_c@mycom.com/
//user/CA_Office/user_d@mycom.com/
//user/CA_Office/user_e@mycom.com/
//sgrp/CA_Office/junior_trader/
//sgrp/CA_Office/trader/
//sgrp/CA_Office/senior trader/
//sgrp/CA_Office/salesEngineer/
//sgrp/CA_Office/salesPerson/
```

```
//sgrp/CA_Office/salesManager/
//sgrp/CA_Office/trading_Manager/
//sgrp/CA_Office/customer/

//user/NY_Office/user_1/
//sgrp/NY_Office/sgrp1/
```

Using Response Attributes

Response attributes are defined as a list of the attributes you want to return from the authorization system when a request is made by an application. Response attributes provide a mechanism for allowing the authorization system to pass arbitrary information back through the Security Framework to the caller. The use of this information is typically application specific. Some examples of how you can use response attributes include:

- **Personalization**—The decision as to what resources to display on a portal may be tied closely to the security policy. Suppose that when a user enters the portal, the portal displays a list of accounts and menu options denoting operations on the accounts. If a user attempts to access a particular item and the attempt is rejected for security reasons, the portal has limited effectiveness. That is, the portal may serve as an information source used for future attacks. By tying the security policy directly to the portal, only the resources that the user is allowed to access are displayed.
- **Business process flow** —Business processes often have inter-task dependencies. For example, suppose that a senior trader has the ability to override the rejection of a trade placed by a junior trader. To make this decision, the senior trader would have to take into account the reasons why the proposed trade violates the security policy, which could be the trade amount, the risk profile, or any of several other reasons. By enhancing the authorization decision with that context, subsequent authorization decisions based on that context can be enabled.
- **Transaction specific data**—An application may need specific facts about authorized or rejected transactions. For example, the application may want to display the post-trade balance for an executed transaction, information that typically would be calculated as part of the authorization process but not returned as part of the authorization decision.

Response attributes are typically specified using built-in evaluation functions that report name/value pairs. There are two functions for returning attributes: `report`() and `report_as`(). These functions always return `TRUE` (if there are no errors), and their information is passed to your application as response attributes, embedded within the `ResponseContextCollector`.

You use report() and report_as() in the policy after an IF statement used in a constraint. It is best to use them in a logical *if this policy is evaluated, then* manner, even though "then" does not exist in the language.

For example:

```
if (constraint) and report_as (name,value);
```

Note: The evaluated policy must result in a GRANT or DENY decision in order for the report() and report_as() functions to be invoked. Consider the following usage:

- If the evaluated policy is applicable (based on action, resource, subject, and constraint expression evaluated to TRUE), then report() and report_as() are invoked for a GRANT or DENY decision.
- If the evaluated policy is not applicable, for example in the case of an ABSTAIN decision, it is just skipped. (If an authorization policy has no policies set on a resource an ABSTAIN result is returned.) If the policy is skipped, then report() and report_as() are not invoked.

While the functions are run when the policy is evaluated, they are not really constraints of the policy. Data reported by the functions are returned only if the adjudicated authorization decision agrees with the policy. This means the attributes returned from GRANT policies are not passed to the caller unless the overall access decision is PERMIT.

The following topics provide more information on using response attributes:

- "report() Function" on page 4-80
- "report_as() Function" on page 4-81
- "Report Function Policy Language" on page 4-81
- "Using Evaluation Plug-ins to Specify Response Attributes" on page 4-82

report() Function

The report function takes one or more attributes as input parameters and sets a corresponding response attribute with the name/value pair of the supplied attributes. For example, suppose you have the attribute called department, containing the value Accounting. If the following constraint was evaluated:

```
IF report(department);
```

the response attribute (department = accounting) is set in the response context results. Your client application can then use this information in many ways, for example:

- As a parameter in a database query where it filters the query results by department
- To personalize a portal page with an accounting department template
- To update the record being modified with the department information

report_as() Function

The `report_as` function loads a named response attribute with a specified value. The value may be an attribute, a constant or a string literal. You can specify multiple values, in which case the response attribute is returned as a list.

```
IF report_as("error","Your account balance is too low");

IF report_as("query", "Select * from record_table where dept_type = ",
department);

IF report_as("userlogin", trading_login,trading_password);

IF report_as("url","http://www.xyz.com/userinfo/xyz100383.htm");
```

Report Function Policy Language

The `report` function returns the name/value pair of the specified attribute. The value may be a one or more strings and is determined using the attribute retrieval mechanism of the authorization system. This means that the attribute can come from the following sources: system, user, resource or context.

The `report_as` function allows you to write the policy to specify both the attribute name and value:

```
report_as("company", "BEA Systems")
```

Additionally, you can specify a list of values, as follows:

```
report_as("accounts", "123", "456", "789")
```

The value portion of the report function supports de-referencing. Assume the user attribute `favorite_color` is part of a user profile. You can put the following statement into a policy:

```
report_as("window_background", favorite_color)
```

This allows you to set the response attribute `window_background` with the value of the favorite color that is stored in another attribute. You can use any of the supported language data types as values, but they are all returned to the provider using their string representation and no additional type data is transmitted.

Reporting the same attribute multiple times from the same policy results in only the last report clause date being used. For example:

```
grant (p,o,s) if report as (”car”, ”porche”) and report_as (”car”, “ford”);
```

where: `(p,o,s)` is shorthand for privilege, object, and subject, results in the response attribute `car = ford`.

Using Evaluation Plug-ins to Specify Response Attributes

The ASI Authorization and ASI Role Mapping providers support the use of custom evaluation plug-ins to generate response attributes. The `report` and `report_as` functions are just special implementations of ASI Authorization and ASI Role Mapping provider plug-ins. Using custom evaluation functions, you can write even more complex statements. For example, the following policy retrieves the current stock price from an authoritative source.

```
grant(//priv/lookup, //app/policy/stockprice, //role/everyone)
if report_stock_price("BEAS");
```

A plug-in that implements this function must handle all of the logic required to obtain the actual stock price and then return it in a response attribute. Listing 4-19 shows an example of how to use a plug-in to implement this function.

Listing 4-19 Stock Price Function Implementation

```
TruthValue report_stock_price(Session &sess, const char *fname,
      char **argv) {
   const char* stock_symbol=argv[0];
   //lookup stock price using custom logic
   double price;
   bool found = lookup_stock_price(stock_symbol, &price);
   if(!found) {
         return TV_FALSE;//price not found
         }
   //change numeric value into a string
   char pricestr[1024];
   snprintf(pricestr,1023,"%f",price);
   //setup the return data
   sess.appendReturnData("stock_price",
                         new AttributeValue((const char*)pricestr));
```

```
   return TV_TRUE;
}
```

For additional information on using ASI Authorization and ASI Role Mapping provider plug-ins, see Provider Extensions in the *Administration Reference Guide*.

Using queryResources and grantedResources

This feature allows a caller to query the authorization system to determine access on a set of resources rather then a single resource. The ASI Authorization provider determines access to all child nodes of the node specified in the access query, and returns lists indicating which nodes are granted and which nodes are denied.

The client performs an `isAccessAllowed` query on the `parentResource`. This resource must be a binding node or a resource of a binding node.

The `queryResources` functionality evaluation is triggered by the presence of some `qrvalue` value in the `com.bea.security.authorization.queryResources` attribute of the `ContextHandler`. The access decision for the `parentResource` is returned, as normal. One of the return attributes for this decision is a `com.bea.security.Authorization.grantedResources` return attribute. One of the return attributes for this decision is a `com.bea.security.Authorization.deniedResources` return attribute.

For `grantedResources`, the value of this attribute is a list of values for the `qrvalue` resource attribute; or, if the `qrvalue` is an empty string, the value is the internal ASI Authorizer name for the resource. This list is an intersection of all child nodes of `parentResource` and all resources for which the ASI Authorization provider and ASI Role Mapping provider and role policy evaluates to `GRANT`. If the `qrvalue` attribute is not defined on a particular child node, it is omitted to allow an application to deal with identification of the resource other than the internal ASI Authorizer representation of it, which is not trivial to convert back to the framework resource.

This list can contain duplicate values. If the empty value for the `qrvalue` is used, the returned resource name is unique and defined for each child node.

The same applies for the `deniedResources`, except for the resources that the policy evaluates to `DENY`. For example, assume that an application makes an `isAccessAllowed` call on the `//app/policy/Foo` resource and sets the value of the `queryResources` attribute to

object_id. The authorization policy has no policies set on the Foo resource, thus an ABSTAIN result is returned.

Now let's assume that Foo has child nodes Foo/A, Foo/B, Foo/C. The authorization policy allows access to Foo/A and Foo/C, given the role policy on Foo by all providers, and the role policy for A and C for a security provider. Assume that A and C have an object_id resource attribute equal to "rA" and "rC". Then, the above query returns an attribute grantedResources with the value ["rA", "rC"].

For role providers other than the ASI Role Mapper provider, roles granted on the parentResource are assumed to apply to all child nodes of the parentResource. For the role policy, it is evaluated as usual for all child nodes.

To receive the results, you must supply a ResponseContextCollector in the ContextHandler request.

When the application needs to call into the Security Framework to query resources it passes in:

```
AppContextElement qrElement = new SimpleContextElement(
"com.bea.security.authorization.", "queryResources", "name");
appContext.addElement(qrElement);
```

When it retrieves the list of resources from the response, for granted resources, it must call:

```
AppContextElement granted = responseContext.getElement(
"com.bea.security.Authorization.grantedResources");
```

or, for denied resources:

```
AppContextElement denied = responseContext.getElement(
"com.bea.security.Authorization.deniedResources");
```

Note: The case for authorization on the request and the response is not the same.

Resource Discovery

When developing policy for use with a Security Service Module, you can use the Discovery mode to help build your policy. Understanding how to write a policy requires that you understand the application you want to protect, the policy representations, and the mapping between the two.

Note: You should never use Discovery mode in a production environment. Discovery mode is used in the development environment to create the bootstrap security policy.

A typical policy consists of the following elements:

- Resources

- Privileges
- Roles
- Attributes (user, group and resource)
- Attributes, privileges, and resource associations
- Rules to grant privileges on resources through roles
- Rules to grant roles to users

The ASI Authorization and ASI Role Mapping providers support a Discovery mode that helps make this task easier. Typically, these providers answer questions about security, but when in Discovery mode, the providers record information about those questions to build your policy (for example, what privileges and resources must be granted to view a particular web page).

To use Discovery mode, you must modify the command line that starts your Security Service Module by adding the following system properties:

```
com.bea.security.providers.authorization.asi.AuthorizationProviderImpl.discoverymode=true
```

```
com.bea.security.providers.authorization.asi.RoleProviderImpl.discoverymode=true
```

You set the system properties using the `-D` command-line switch in the appropriate file, depending on which Security Service Module (SSM) you are targeting. Table 4-23 lists the files and their default locations for each type of SSM.

Table 4-23 Setting System Properties for Discovery Mode

Security Service Module Type	File Name	File Default Location
Apache	`set-env.sh` (UNIX only)	*BEA_HOME*`\ales26-ssm\apache-ssm\instance\<`*instancename*`>\bin`
IIS Web Server	`set-env.bat` (Windows only)	*BEA_HOME*`\ales26-ssm\iis-ssm\instance\<`*instancename*`>\bin`
Java	`set-env.bat (.sh` for UNIX)	*BEA_HOME*`\ales26-ssm\java-ssm\instance\<`*instancename*`>\bin`

Table 4-23 Setting System Properties for Discovery Mode

Security Service Module Type	File Name	File Default Location
Web Services	`wlesws.wrapper.conf`	*BEA_HOME*`\ales26-ssm\webservice-ssm\instance\<`*instancename*`>\config`
WebLogic Server 8.1	`set-wls-env.bat` (`.sh` for UNIX)	*BEA_HOME*`\ales26-ssm\wls-ssm\instance\<`*instancename*`>\bin`

Note: The policy files are stored in the domain directory from which the `startWeblogic.bat` or `startWeblogic.sh` script is started and configured to run in Discovery mode.

A sample policy is recorded by the providers as you traverse an application. This is a learning process for the providers and they can only learn about the parts of the application that you use. If a portion of the application is not used, no information or policy about it is recorded. The generated policy is output to a set of files that you can import later, by using the Policy Import tool described in "Importing Policy Data" on page 7-1.

Among other things, the ASI Authorization and ASI Role Mapping providers transform their requests into a proprietary format. A request consists of the following four elements:

- Subject
- Resource
- Action
- Attributes

The ASI Authorizer providers build this information based on data contained in the request to the provider. Each of these elements has different restrictions on the allowable character set. The providers automatically normalize any invalid characters to produce a valid entry. See "Character Restrictions in Policy Data" on page 4-58 for further details.

Using the Entitlements Management Tool

This section covers the following topics:

What is the Entitlements Management Tool?

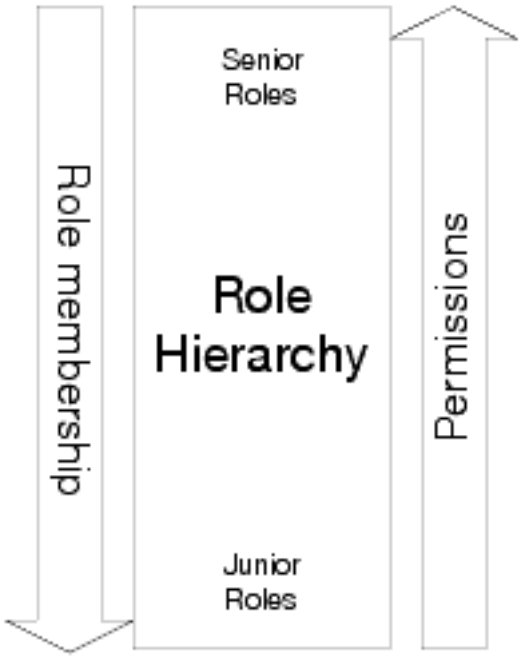
In addition to the Administration Console, AquaLogic Enterprise Security includes another user interface, the Entitlements Management Tool. The Entitlements Management Tool can be used by business users to manage roles, users and groups, and entitlements. This user interface enables you to manage your users' entitlements based on the Roles Based Access Control (RBAC) model. The Entitlements Management Tool enables business users to answer questions like "what entitlements does a user have?" or "which users are allowed to access a given resource?"

Understanding the RBAC Model

The Entitlements Management Tool implements a hierarchical RBAC model. In this model, there is a seniority relationship between roles: senior roles inherit the permissions of their juniors and junior roles inherit members of their parents. In this way, data is inherited in both directions, with

membership moving down and permissions moving up the tree. Using a hierarchical model makes it possible to aggregate permissions associated with users.

Figure 5-1 Roles Based Inheritance



The Entitlements Management Tool displays which permissions and roles are assigned directly and which are inherited.

ALES RBAC Model Concepts

Four important concepts to understand in the ALES RBAC Model are:

- roles
- permissions
- permission sets
- separation of duties constraints

Roles are defined to represent job functions in an organization (for example, teller or executive VP of Finance). A role's collection of permissions (and permission sets) represent the set of entitlements for a role – all of the actions that a role can take within a system. Roles can have attributes that will be available to the authorization system at runtime. Attributes follow the RBAC inheritance model from the child to the parent, so a parent role will have all the attributes of its child roles. Users and groups are assigned to roles via membership rules. A user or group can be assigned directly to a role, or conditionally assigned to a role.

Permissions are named objects that represent an action on a resource. Permission sets are named hierarchical objects that are collections of permissions. Under the inheritance model of the permission set hierarchy, parent permission sets inherit permissions and attributes from their children. Like roles, permissions and permission sets can have attributes that will be available to the authorization system at runtime. Permissions can either be assigned directly to a role, or to a permission set that is in turn assigned to a role. Attributes follow the RBAC inheritance model from the child to the parent, so a parent permission set will have all the attributes of its child permissions or permission sets.

Separation of duties constraints define exclusive relationships between roles. A separation of duties constraint allows you to say, for example: If a user has the role auditor he can not also have the role of trader. At runtime, the system will not allow a user to have both roles at the same time and thereby prevent a user from having incompatible permissions.

Summary of Entitlements Management Tool Functions

The Entitlements Management Tool provides the following functions:

- "Role Management Functions" on page 5-3
-
- "Permission Management Functions" on page 5-4
- "Separation of Duties Functions" on page 5-4
- "Entitlements Reporting Functions" on page 5-4

Role Management Functions

Use the Roles node in the Entitlements Management Tool to:

- Create, delete, and copy roles
- Add and remove attributes to roles

- Add and remove users and groups to roles
- Add or remove permission sets to roles
- Add or remove permissions to roles
- Set role mapping constraints
- Scope roles to resources

For more information, see "Working with Roles" on page 5-9.

Permission Management Functions

Use the Permissions node in the Entitlements Management Tool to:

- Create, delete, or copy permissions and permission sets
- Add or remove a permission or a permission set to a permission set

For more information, see "Working with Permissions and Permission Sets" on page 5-16

Separation of Duties Functions

Use the Separation of Duties node in the Entitlements Management Tool to:

- Add or remove separation of duty constraints on roles
- Find conflicts in separation of duty constraints on roles

For more information, see "Separation of Duties Constraints" on page 5-20

Entitlements Reporting Functions

Use the Reports node in the Entitlements Management Tool to:

- What permissions does a user have
- What users have a permission
- What groups is a user a member of
- What roles is a user a member of
- What users are a member of a role
- What permissions does a role have

- What roles is a permission associated with

For more information, see "Generating Reports" on page 5-22

Setting Up the Entitlements Management Tool

Before you can use the Entitlements Management Tool, you need to do the following:

- "Load the Entitlements Management Tool Policies" on page 5-5
- "Deploy the Entitlements Management Tool Web Application" on page 5-5

Load the Entitlements Management Tool Policies

Before you can use the Entitlements Management Tool, you need to load the policies that govern access to and use of the tool. To load these policies:

1. Using the Policy Import tool (`policyloader`), load the policies using the edited `load.entitlements.conf` file, using a command like the following:

   ```
   ..\..\bin\policyloader.bat
   <bea_home>\ales26-admin\entitlements\policy\load.entitlements.conf
   ```

   For more information about the Policyloader utility, see "Importing Policy Data" on page 7-1.

2. Make sure there were no errors on import.
3. Set new passwords for the default users:
   a. Log in to the ALES Administration Console and open the Identity node in the navigation tree.
   b. In the right pane, select `asi`, then select `user`.
   c. Select one of the default Entitlements Management Tool users, for example `rolesadmin`, and click the Edit button to set a new password for this user.
   d. Repeat for each of the other Entitlements Management Tool users (`rbacadmin`, `permissionsadmin`, `reportingadmin`).

Deploy the Entitlements Management Tool Web Application

The procedure for deploying the Entitlements Management Tool Web application depends on which servlet container you are using:

Deploying on WebLogic Server 9.x

1. Start the WebLogic Server Administration Console for the WebLogic Server instance where the ALES Administration Server is installed.
2. In the Change Center of the WebLogic Server Administration Console, click Lock & Edit.
3. In the Domain Structure panel of the WebLogic Server Administration Console, click Deployments.
4. Click Install.
   a. Locate the Entitlements Management Tool WAR file by navigating to `<bea_home>\ales26-admin\entitlements\wls9`.
   b. Select `entitlementsadministration.war` and click Next.
   c. Click Next.
   d. Click Next.
   e. Click Finish.
5. In the Change Center of the WebLogic Server Administration Console, click Activate Changes.
6. On the Deployments page, select the `entitlementsadministration` application.
7. Select Start > Servicing all requests and click Yes.

Deploying on WebLogic Server 8.1

1. Start the WebLogic Server Administration Console for the WebLogic Server instance where the ALES Administration Server is installed.
2. In the left panel of the WebLogic Server Administration Console, select Deployments > Web Application Modules.
3. Click Deploy a new Web Application Module.

4. Navigate to `<host>\<bea_home>\ales26-admin\entitlements\wls8`.
5. Select `entitlementsadministration.war` and click Next.
6. Click Deploy.

Deploying on Apache Tomcat

1. Stop the ALES Administration Server if it is running.
2. Copy the WAR file for the Entitlements Management Tool, `entitlementsadministration.war`, from `<bea_home>\ales26-admin\entitlements\tomcat\` to `<bea_home>\ales26-admin\asiDomain\applications\`.
3. Restart the ALES administration server. The Entitlements Management Tool will be loaded.

Configuring the RBAC Model in SSMs

This section describes how to configure a Security Service Module to use the RBAC model.

1. Install the SSM and create the SSM instance that you will use to secure your applications. For more information, see the *Installing the Security Service Module* guide for the type of SSM you are using.
2. Add the RBAC policy. In the ALES Administration Console, add authorization policies for the resources you want to secure like the following:

   grant(any, //app/policy/*resourcename*, //role/Everyone) if rbac_eval_action_resource_in_entitlements(entitlements,sys_privilege);
3. Distribute the policies.
4. Set the metadirectory for the ASIAuthorizer provider. This procedure depends on which SSM you are configuring. For SSMs other than the WLS 9.x SSM:
   a. In the ALES Administration Console, navigate to ASIAuthorizationProvider.
   b. In the **Metadirectory** tab for the ASIAuthorizationProvider, check the **Use Metadirectory** checkbox and set **JDBC URL**, **JDBC Driver**, **Database System**, **Database Login**, and **Database Login Password** to the same values as in the DatabaseAuthenticator configured for the `asiadmin` security configuration.
   c. Distribute the configuration.

   For the WLS 9.x SSM:

a. In the WebLogic Server Administration Console, select **Security Realms** and select the security realm you are configuring.

b. In the **Providers: Authorization** tab, select the ASIAuthorizer.

c. On the **Provider-Specific** tab for the ASIAuthorizer, check the **Use Metadirectory** checkbox and set **JDBC URL**, **JDBC Driver**, **Database System**, **Database Login**, and **Database Login Password** to the same values as in the DatabaseAuthenticator configured for the `asiadmin` security configuration.

d. In the Change Center of WebLogic Server Administration Console, click **Activate Changes**.

5. Restart the application or application server.

Using the Entitlements Management Tool

The Entitlements Management Tool is a browser-based web application. You can access it at this URL:

```
https://<hostname>:7010/entitlementsadministration
```

Like the ALES Administration Console, the Entitlements Management Tool supports only the Microsoft Internet Explorer browser.

Saving and Distributing Changes

When you use the Entitlements Management Tool to make changes, the tool's **Save Changes** and **Revert Changes** buttons become active. The Entitlements Management Tool uses a transactional change model; after you have made all the changes you need to make and click **Save Changes**, all the changes are committed in a single transaction. If the changes require it, they are distributed through the ALES Business Logic Manager.

Security for the Entitlements Management Tool

The Entitlements Management Tool defines the following roles by default that limit access to the tool:

Table 5-1 Default Roles

Role	Description
RBACAdmin	The RBACAdmin can perform all possible operations in the Entitlements Management Tool.
RolesAdmin	The Roles Admin can work only in the roles section of the navigation tree in the Entitlements Management Tool.
SodAdmin	The SoD Admin can work only in the Separation of Duties section of the navigation tree in the Entitlements Management Tool.
PermissionsAdmin	The Permissions Admin can work only in the permissions section of the navigation tree in the Entitlements Management Tool.
ReportingAdmin	The Reporting Admin can only generate reports.

The following users are defined by default, with the following roles:

Table 5-2 Default Users

User Name	Role
system	RBACadmin
rolesadm	RolesAdmin
permadm	PermissionsAdmin
reportingadm	ReportingAdmin

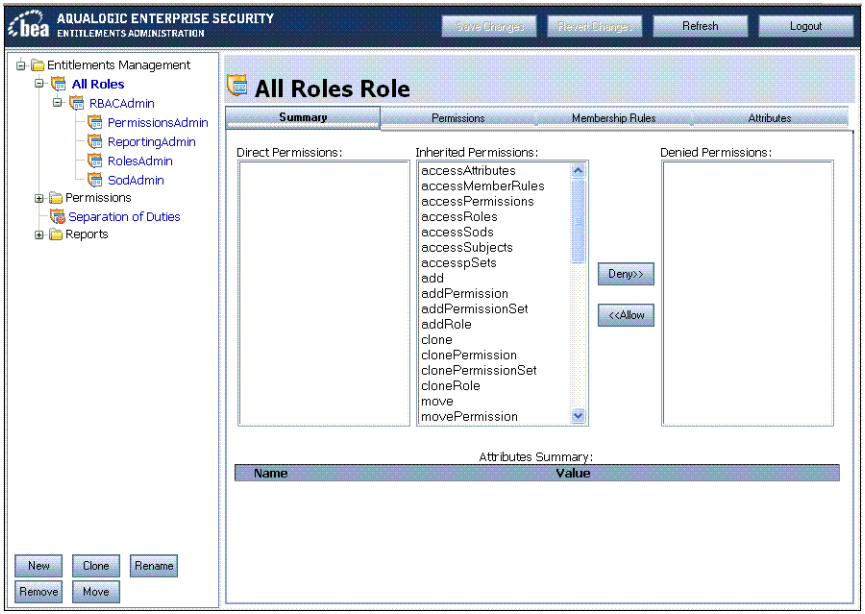
Working with Roles

The Entitlements Management Tool enables you to create hierarchies of roles. In a hierarchical role system, you can create a tree of roles, with each parent node in the tree possessing all the permissions of its child roles.

Viewing Roles

To view all the roles defined in your ALES security realm, in the left panel of the Entitlements Management Tool, expand **Entitlements Management > All Roles**. The Roles summary page displays a role's entitlements, with separate columns indicating which permissions are directly assigned, inherited from child roles, or denied to this role.

Figure 5-2 Roles Summary Page



Creating a New Role

To create a new role:

1. In the left panel of the Entitlements Management Tool, expand **Entitlements Management > All Roles** and then select the parent role under which you want to create a new role.
2. At the bottom of the left panel of the Entitlements Management Tool, click the **Add** button.
3. In the **New Child Node** page, give the new role a name and click OK.
4. Next, assign permissions or permission sets to the new role.

To assign a permission to a role:

a. In the left panel of the Entitlements Management Tool, select the new role and click the **Permissions** tab at the top of the right panel of the Entitlements Management Tool.

b. Click **Modify Permissions**. The **Add Permissions to Role** page appears. Use the **Add** and **Remove** buttons to specify the permissions you want to assign to the role and click OK.

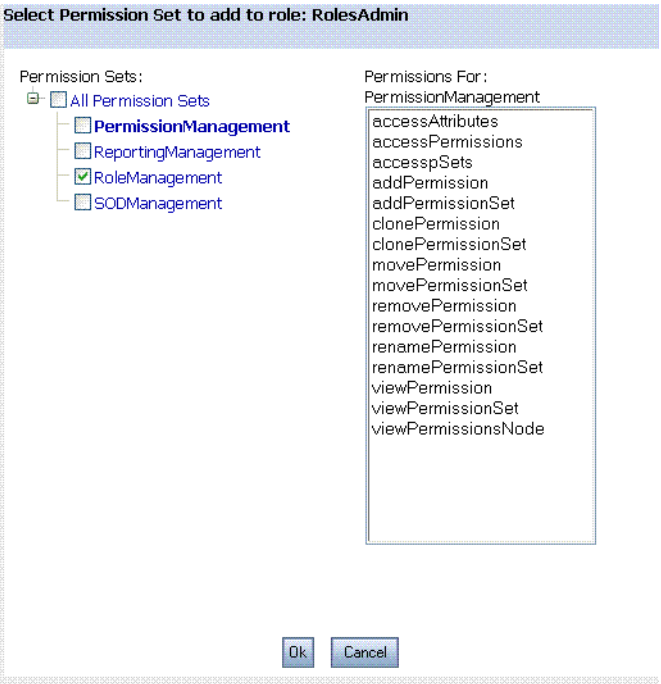
Figure 5-3 Adding Permissions to a Role

To assign a permission set to a role:

a. In the left panel of the Entitlements Management Tool, select the new role and click the **Permissions** tab at the top of the right panel of the Entitlements Management Tool.

b. Click **Modify Sets**. The **Add Permission Sets** page appears. The left column lists the names of the permission sets that have been defined in your installation.

Figure 5-4 Adding Permission Sets to a Role



c. Select a permission set and the permissions contained in that permission set are displayed in the right column.

d. Check the names of the permissions you want to assign to the role and click **Ok**.

5. Define the membership of the new role. Click the **Membership Rules** tab at the top of the right panel of the Entitlements Management Tool and click **New**.

   The New Member Rule page appears. In the **Subject** tab, you can select the users or groups to include (using the Grant option) or exclude (using the Deny option) from the new role. Use the **Add** and **Remove** buttons to specify the users or groups covered by the grant or deny rule and click OK.

6. Further define the role using the **Conditions** tab. You can limit membership in a role based on the presence or absence of attribute values you have previously defined as Declarations in the ALES Administration Console. For information about using declarations, see "Declarations" on page 2-14.

In the **Conditions** field, enter one or more attribute-based conditions, such as:

```
if subject.location = "New York"
```

This member rule will apply only if the subject has the specified attribute name/value pair.

Assigning Role Attributes

You can assign attributes to roles. An attribute is a name/value pair that will be available to the ALES authorization system at run time. Assigning an attribute enables the creation of authorization policies based on the attribute value.

To assign an attribute to a role:

1. In the left panel of the Entitlements Management Tool, expand **Entitlements Management > All Roles** and then select the role.
2. In the right panel, click the **Attributes** tab and click ☠♏◆.
3. In the **Modify Role Attribute** page, enter the name and value of the attribute you want to assign.

Modifying and Removing Roles

You can use the Entitlements Management Tool to move, clone, delete, modify, or rename roles.

To move a role:

1. In the left panel of the Entitlements Management Tool, select the role you want to move and click **Move**.
2. In the **Select the destination node** page, select the role that should be the parent in the roles hierarchy and click **OK**.

   The role you moved, including all its child roles, is now a child of a different parent in the roles hierarchy.

To clone a role:

1. In the left panel of the Entitlements Management Tool, select the role you want to copy and click **Clone**.
2. In the **Select the destination to clone** page, select the role that should be the parent of the cloned role in the roles hierarchy and click **OK**.

   The role you cloned, including all its child roles, is now a child of a different parent in the roles hierarchy, while the original remains in its original place in the roles hierarchy. You

then may want to modify the new cloned role to fit its purpose, including renaming the role.

To delete a role:

1. In the left panel of the Entitlements Management Tool, select the role you want to delete and click **Remove**.
2. In the **Remove Node** page, confirm that you want to delete the role and click **OK**.

   The role you selected, including all its child roles, is deleted.

Working with Identities

You can use the Entitlements Management Tool to manage user and group identities in your ALES security realm. The Identity node in the Entitlements Management Tool presents user and group information on three tabs:

- "Users Tab" on page 5-14
- "Groups Tab" on page 5-15
- "Attributes Tab" on page 5-16

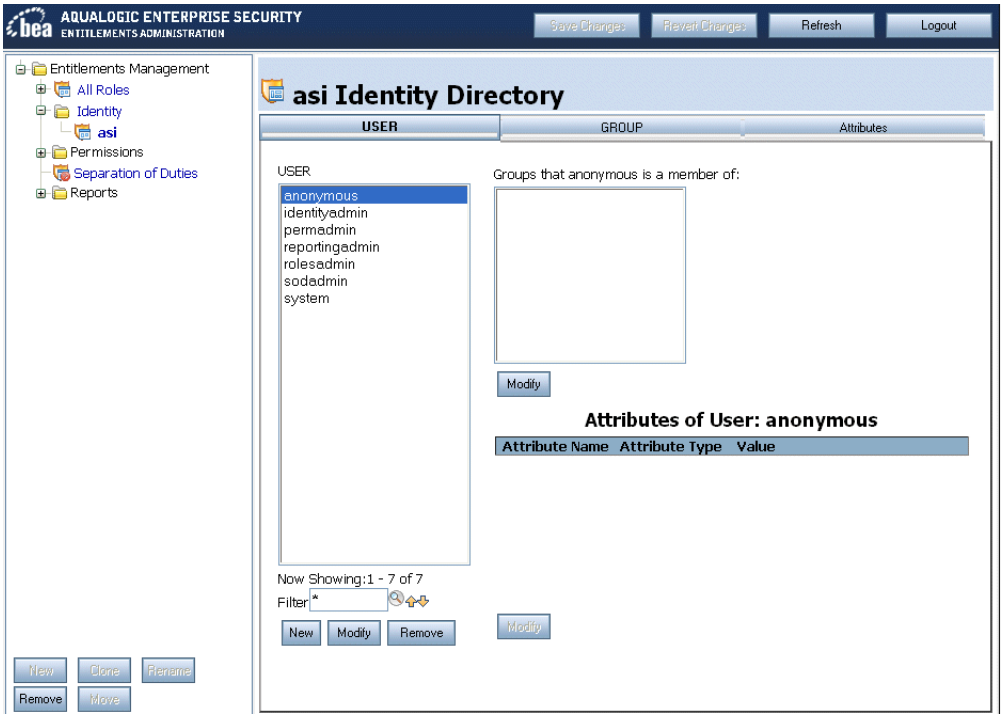
By default, the Identity node has a single child identity directory, named `asi`. You can use the **New** and **Remove** buttons to add or delete other identity directories.

For more information about identities in ALES, see "Identities" on page 2-7.

Users Tab

To view all the users defined in your ALES security realm, in the left panel of the Entitlements Management Tool, expand **Entitlements Management > Identity > asi**. The Users tab displays user names, the groups each user is a member of, and attributes assigned to each user.

Figure 5-5 Users Tab



You can use this page to add or remove users and to specify which groups a user is a member of.

To add a user to a group:

1. In the **User** column, select the username.
2. In the **Groups** column, click **Modify**.
3. In the **Modify group membership** window, use the **Add** and **Remove** buttons to set the user's group memberships and click **Ok**.

Groups Tab

To view all the groups defined in your ALES security realm, in the left panel of the Entitlements Management Tool, expand **Entitlements Management > Identity > asi** and click the **Groups** tab. You can use the Groups tab to add or remove groups and view a group's members and attributes.

Attributes Tab

To view all the identity attributes defined in your ALES security realm, in the left panel of the Entitlements Management Tool, expand **Entitlements Management > Identity > asi** and click the **Attributes** tab. You can use the Attributes tab to define identity attributes, which you can then assign to users. You can also modify or remove attributes.

To define a new attribute:

1. In the left panel of the Entitlements Management Tool, expand **Entitlements Management > Identity > asi** and click the **Attributes** tab.
2. In the Attributes tab, click **New**. The **New Attribute** window opens.
3. In the **New Attribute** window, specify the new attribute's name, type, and default value, and whether the attribute is a list. Note that only list attributes can be group attributes. Click **Ok**.

To modify the value of an attribute for a user or group:

1. Select the name of the user or group.
2. Under Attributes, click **Modify**. The **Modify User Attribute** window opens.
3. Set the new value for the attribute and click **Ok**.

For more information about identity attributes, see "Identity Attributes" on page 2-8 and *Understanding Identity Attributes* in the Administration Console help.

Working with Permissions and Permission Sets

Permissions can be assigned to roles directly or be assigned to permission sets that are then assigned to roles. A permission set is a hierarchical collection of permissions that can be assigned to roles. A child permission set can only have one parent, but a permission set or permission can be assigned to many roles. In addition, permissions can be assigned to many permission sets.

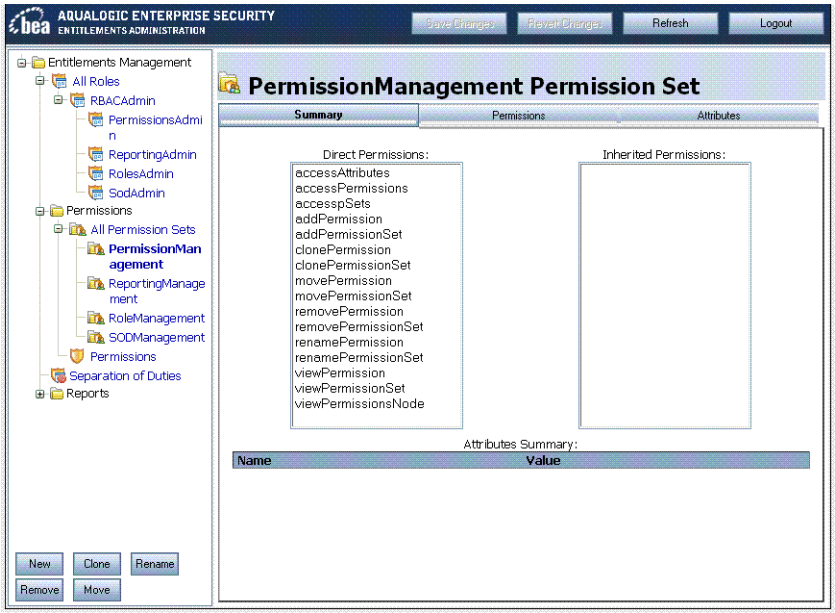
Grouping permissions into permission sets can greatly ease the task of administering roles and policies. Since individual permissions are so specific and small-grained, a given role might require a large number of individual permissions. By grouping permissions into a permission set, it can be easier to create and maintain roles and permissions that correspond to business tasks.

Viewing Permission Sets

To view all the permission sets defined in your ALES security realm, in the left panel of the Entitlements Management Tool, expand Entitlements Management > Permissions > All

Permission Sets. The Permission Sets summary page displays a permission set's entitlements, with separate columns indicating which permissions are directly assigned or inherited from child sets. The Permissions tab enables adding or removing permissions in a permission set, while the Attributes tab enables assigning attributes to a permission set, as described in "Assigning Permission Attributes" on page 5-19.

Figure 5-6 Permission Sets Summary Page



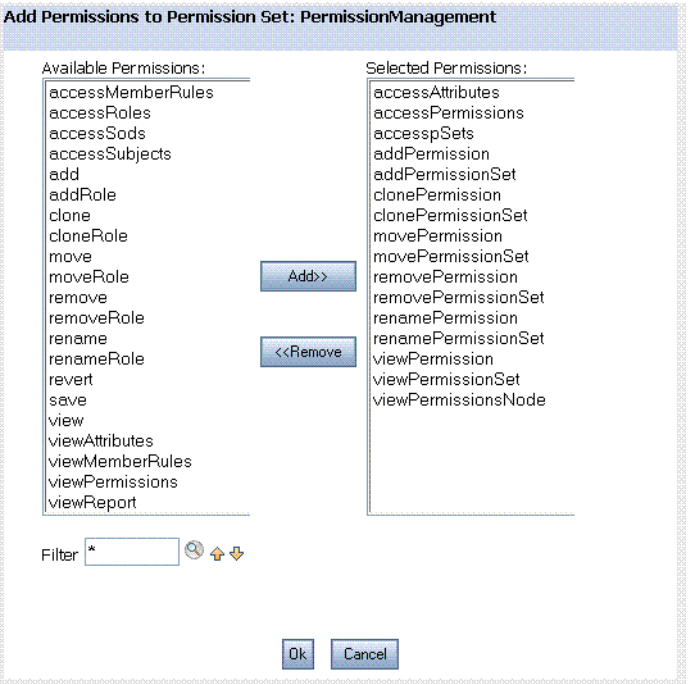
Creating a New Permission Set

To create a new permission set:

1. In the left panel of the Entitlements Management Tool, expand Entitlements Management > Permissions and then select the parent node under which you want to create a new permission set.
2. At the bottom of the left panel of the Entitlements Management Tool, click the Add button.
3. In the New Child Node page, give the new permission set a name and click OK.

4. Next, add permissions to the new permission set. In the left panel of the Entitlements Management Tool, select the new permission set and click the Permissions tab at the top of the right panel of the Entitlements Management Tool.
5. Click Modify. The Add Permissions to Permission Set page appears. Use the Add and Remove buttons to specify the permissions you want to include in your new permission set and click OK.

Figure 5-7 Adding Permissions to a Permission Set



Modifying the Permission Set Hierarchy

By default, the Entitlements Management Tool has no permission sets defined, and the only level of the permission set hierarchy is the All Permission Sets node. After you create one or more permission sets, you can modify the permission set hierarchy in a number of ways:

- Use the Move button to move a permission set and all its children to a different node in the hierarchy.

- Use the Clone button to create a copy of a permission set that you can then modify.

1. In the left panel of the Entitlements Management Tool, expand Entitlements Management > Permissions and then select the permission set you want to move. Click Move.
2. In the Select the destination node page, select the new parent node for the permission set you are moving and click OK.

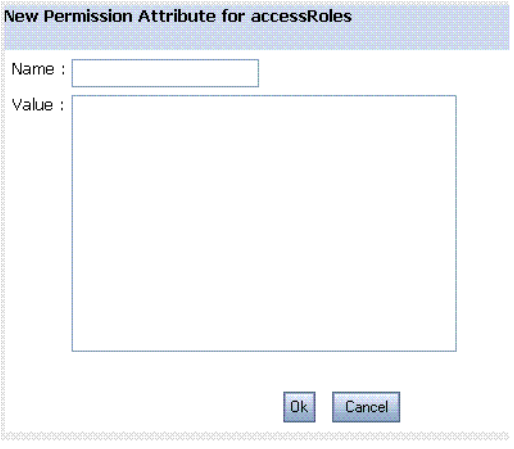
Assigning Permission Attributes

You can assign attributes to permissions and permission sets. An attribute is a name/value pair that will be available to the ALES authorization system at run time. Assigning an attribute enables the creation of authorization policies based on the attribute value.

To assign an attribute to a permission or permission set:

1. In the left panel of the Entitlements Management Tool, expand Entitlements Management > Permissions and then select the permission or permission set.
2. In the right panel, click the Attributes tab and click New.

Figure 5-8 Adding Attributes to Permissions



3. In the Modify Permission Attribute page, enter the name and value of the attribute you want to assign.

Separation of Duties Constraints

Separation of duties constraints are used to prevent conflicts of interest in a role based system. Without separation of duties constraints, the hierarchical role model can give users permissions associated with conflicting roles. For example, suppose you have a trading system with Trader and TradeAuditor roles, both reporting to a VP. Without a separation of duties constraint, users with the VP role would inherit both the Trader and TradeAuditor entitlements. To ensure that the same user could not be the originator of a trade and the approver of the same trade, you can create a separation of duties constraint to that specifies that users with the TradeAuditor role cannot also have the Trader role.

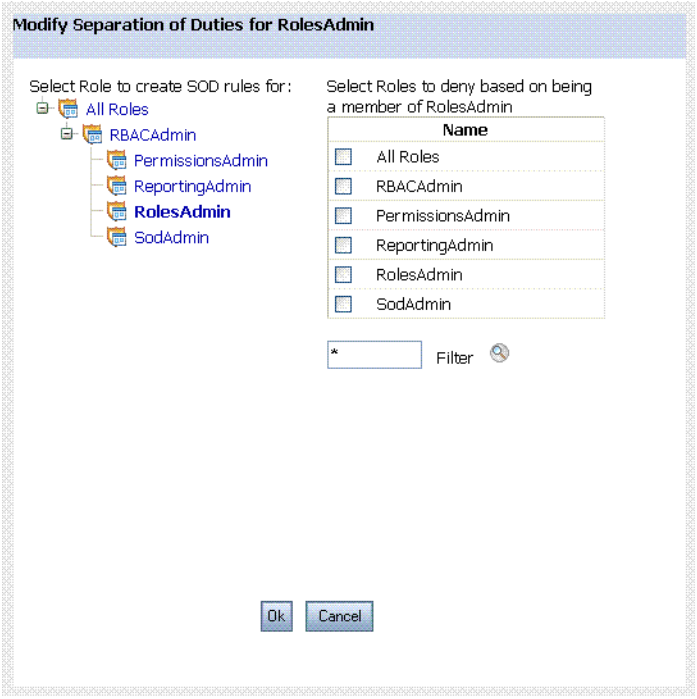
Separation of duties places constraints on the assignment of users to roles so that membership in one role can preclude membership in another. The Separation of Duties node in the Entitlements Management Tool enables you to select a defined role (which we will refer to as “the constrained role”) and then specify which other roles are denied to subjects who have the constrained role.

To create a new separation of duty constraint:

1. In the left panel of the Entitlements Management Tool, select **Separation of Duty**.
2. In the Separation of Duty page, click **New**.

   The **Modify Separation of Duty** page displays.

Figure 5-9 Adding Separation of Duties Constraints



3. From the tree of roles in the left column of the **Modify Separation of Duty** page, select the role for which you want to define a constraint.
4. From the list of roles in the right column of the **Modify Separation of Duty** page, check the roles that should be denied to subjects in the constrained role.
5. Click **OK**.

The **Find Conflict** button allows you to find any conflicting role assignment policies in the system based on the separation of duties defined. If any conflicting roles exist, you need to modify your role assignments to eliminate the conflicts.

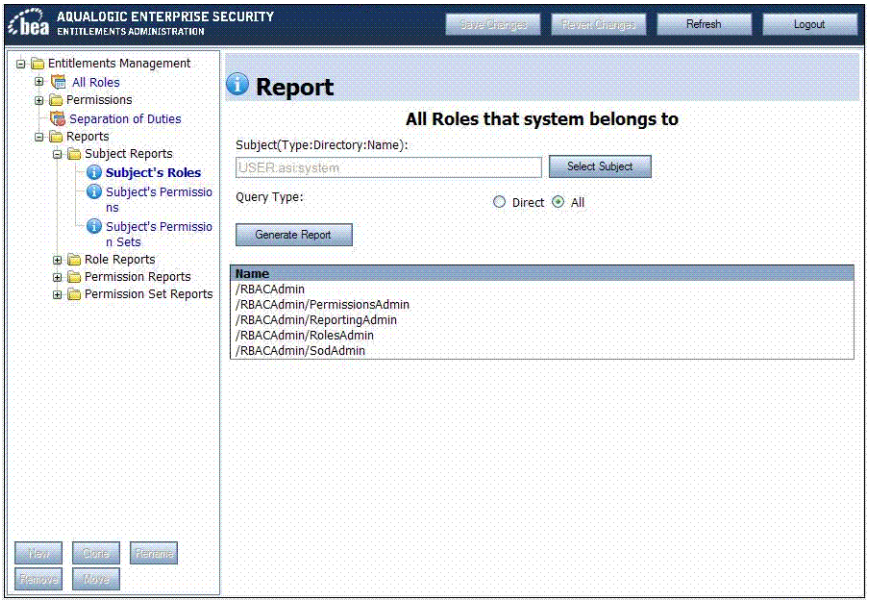
Generating Reports

The Reports node of the Entitlement Management Tool enables you to generate reports about subjects, roles, permissions, and permission sets in your security realm. The following reports are available:

Table 5-3 Available Entitlement Reports

Report	Description
Subject's Roles	Which roles does this subject have?
Subject's Permissions	Which permissions does this subject have?
Subject's Permission Sets	Which permission sets does this subject have?
Role's Subjects	Which subjects have this role?
Role's Permissions	Which permissions does this role have?
Role's Permission Sets	Which permission sets does this role have?
Permission's Permission Sets	Which permission sets does this permission belong to?
Permission's Roles	Which roles have this permission?
Permission's Subjects	Which subjects have this permission?
Permission Set's Permissions	Which permissions are members of this permission set?
Permission Set's Roles	Which roles have this permission set?
Permission Set's Subjects	Which subjects have this permission set?

Figure 5-10 Reports in the Entitlements Management Tool



To generate a Subject's Roles report:

1. In the left panel of the Entitlements Management Tool, expand **Reports > Subject** and select **Subject's Roles**.

   The **Report** page appears in the right panel.

2. Use the **Select Subject** button to browse for the name of a subject. The subject can be either a user or a group.

3. Select a query type, one of:

   – **Direct** - returns only the Subject's directly-assigned roles

   – **All** - returns both directly-assigned and inherited roles of the Subject

4. Click **Generate Report**.

   The **Reports** page lists all the roles your selected Subject has.

Generating any of the other reports is essentially the same.

CHAPTER 6

Extending the Entitlements Management Tool

ALES version 2.6 makes it possible for you to extend the Entitlements Management Tool to manage custom objects.

The Entitlements Management Tool provides a powerful and flexible way of managing roles and permissions. It now allows you to manage custom objects by extending it.

You extend the Entitlements Management Tool by adding new objects to the navigation tree. You can then add custom JSPs to manage any data associated with the objects. The custom objects and their attributes can be part of an ALES policy and used to affect an access decision.

The ability to extend the Entitlements Management Tool is not limited to the model used in the current hierarchal implementation: you can extend the tool to support customer-specific entitlements modeling. For example, you might incorporate a custom modeling implementation with the existing Entitlements Management Tool role, permission set, and permission modeling.

This section describes how to extend the Entitlements Management Tool. The following topics are described:

Why Might You Want to Extend the UI?

The Entitlements Management Tool is very powerful and flexible, but it implements a particular modeling scheme. You might find it more convenient to instead reuse the tool and JSF framework for custom modeling.

As an example, consider the case where a publishing company wants to manage a subscription model using the Entitlements Management Tool. (This scenario is the basis for the complete example described in "Example of Extending the Entitlement UI" on page 6-34.)

Assume that the customer service representatives who manage subscription data need to be able to do the following:

- Manage a list of magazines to which users can subscribe
- Manage user subscriptions, where a subscription can contain multiple magazines with a duration for which the subscription is valid.
- Manage user licenses, where each license consists of a subscription and a start date.

The sections that follow illustrate at a very high level how you might extend the Entitlements Management Tool to handle this scenario.

"Extending the Entitlements Management Tool: Main Steps" on page 6-13 describes how to extend the Entitlements Management Tool in detail.

Managing a Subscription Model: Step 1

The first step is to design your UI model. Decide what objects you need in the navigation tree, the relationships between those objects, the policies you need, and so forth.

You must be familiar with the following concepts:

- The RBAC model, as described in "Understanding the RBAC Model" on page 5-1.
- Java Server Faces (JSF) is a Java framework for building user interfaces for web applications. It is the foundation of the Entitlements UI.

  See Developing Web Applications with JavaServer Faces for more information.

- Attribute Inheritance Mode operates in two modes, as described in "Create a metaobject_mappings.properties Configuration File Under WEB-INF/config" on page 6-13:
  - `BOTTOM_TO_TOP` – Parent to child inheritance
  - `TOP_TO_BOTTOM` – Child to parent inheritance

Managing a Subscription Model: Step 2

Configure a new Custom Tree node in the Entitlements Management Tool tree.

When extending the Entitlements Management Tool, you cannot add to an existing root node in the hierarchy. You can only create a new root node, and then add children to it through the UI. To do this, you extend the `EUIMetaObjectNode` object. `EUIMetaObjectNode` is a generic hierarchal object that facilitates the creation of a selectable tree node.

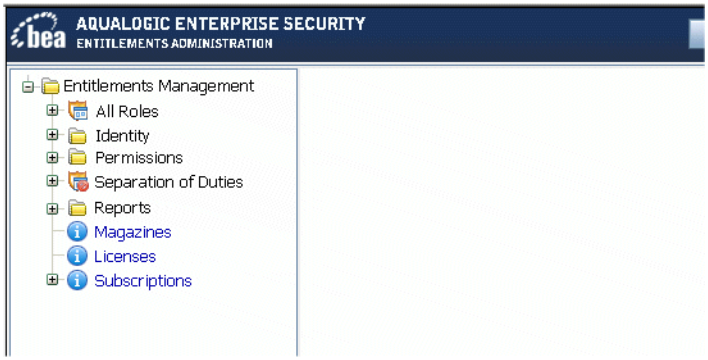
The key parameter when extending `EUIMetaObjectNode` is the node type. When you create a new node, you overload the node type in all of the constructors so that the Entitlements Management Tool will render the desired content on the main page. You then create a mapping file and update two JSP files, navigation.jsp and main.jsp, to display the object.

The concept of extending `EUIMetaObjectNode` and how the node type relates to the JSP files is described in detail in "Extending the Entitlements Management Tool: Main Steps" on page 6-13.

Add the following three new high-level nodes to the navigation tree, as shown in Figure 6-1.

- Magazines
- Licenses
- Subscriptions

Figure 6-1 Managing a Subscription Model: Step 2



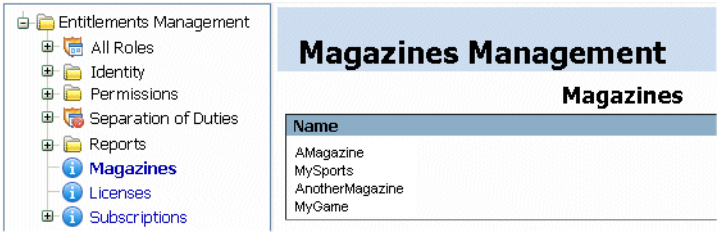
The administrator can now add and delete objects of each type. For this example, we use a flat list of licenses, magazines, and subscriptions.

New objects to be managed could also be hierarchical, such as by organizational structure, geographical hierarchy, product categories, and so forth.

Managing a Subscription Model: Step 3

Add a custom JSP to allow the administrator to add and delete publications, as shown in Figure 6-2.

Figure 6-2 Managing a Subscription Model: Step 3

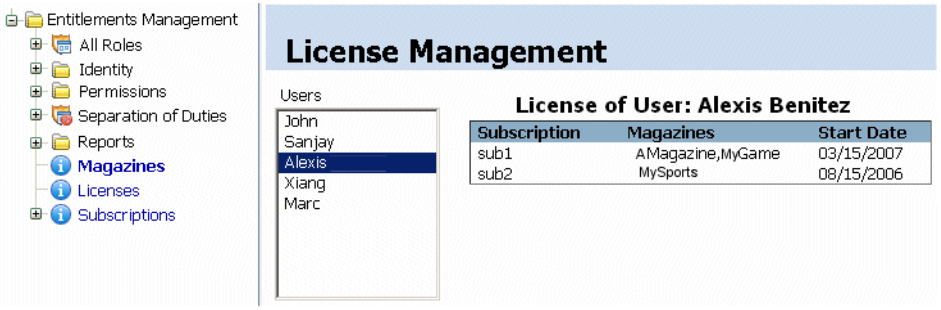


In this case we have a flat space of magazine names managed as a list. Magazines could also be managed as objects in the tree with additional data associated with each.

Managing a Subscription Model: Step 4

Add a custom JSP to allow the administrator to manage licenses for each customer, as shown in Figure 6-3.

Figure 6-3 Managing a Subscription Model: Step 4

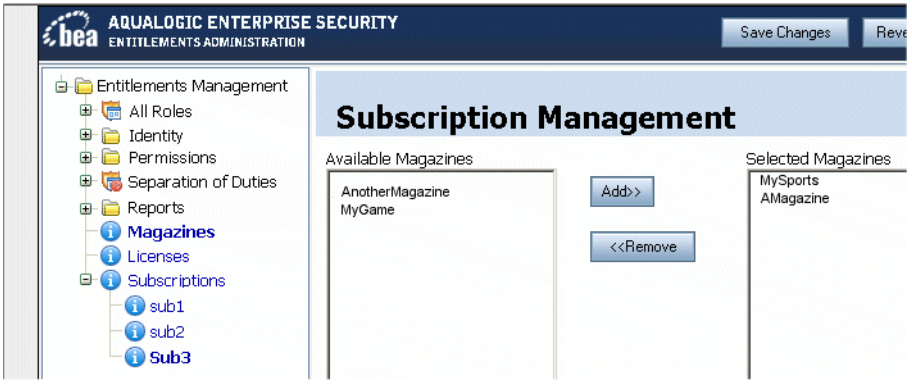


Each license can have multiple magazines, and each license has a start date.

Managing a Subscription Model: Step 5

Add a JSP to allow the administrator to manage each subscription, as shown in Figure 6-4.

Figure 6-4 Managing a Subscription Model: Step 5



Subscriptions can be added or deleted from the navigation tree, and magazines can be added to a subscription. A duration is also set for each subscription.

By extending the EUI, the publishing company's applications can now use the data in ALES policies to:

- Control access to online content for each publication
- Start and stop subscriptions
- Send renewal or billing notifications, and so forth

The custom data is stored in the ALES database. It can be used in ALES policy through the use of a custom attribute retriever or a custom evaluation function.

Components of the Entitlements Management Tool

The Entitlements Management Tool is delivered as an archived web application. There are three major components:

- Presentation Layer – JSP files using javascript and tags that dictate how the UI is displayed.

  For the ALES 2.6 release, the presentation layer is publicly accessible. You can modify the layout of the UI and its look and feel by directly modifying the existing set of JSP files.

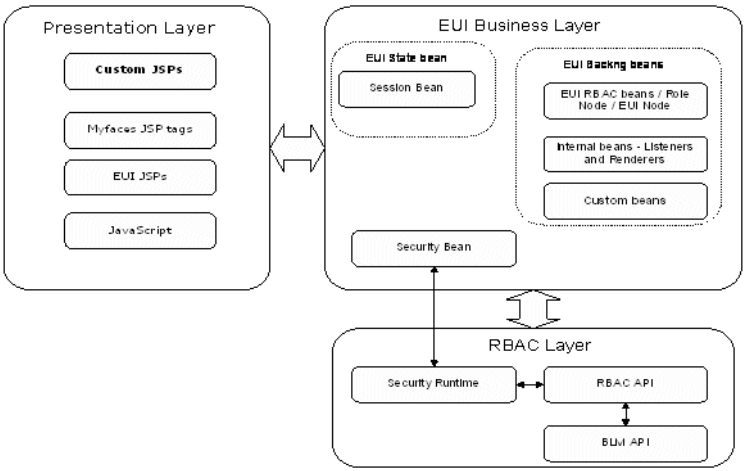
- Business Logic – Backing Beans that provide state and caching functionality for managing user entitlements.

  The backing Beans provide a public management interface for referencing data presented in the Entitlements Management Tool.

- RBAC_API - A management API built on top of the existing BLM_API that provides an interface for managing RBAC components (roles, permissions and permission sets). The RBAC_API is available only through the Bean interface.

Figure 6-5 shows how the Entitlements UI components work together.

Figure 6-5 RBAC API Architecture



This figure illustrates how all the components in the Entitlements UI interact with each other:

- The Myfaces JSP tags reference code that is executed in the Myfaces tag library implementation.
- The tag library's components used by the entitlements UI were extended to call `isAccessAllowed` prior to being rendered to allow for component-level redaction of the UI via the security bean.
- Once authorization is complete, the tags then call into the session beans to get the appropriate information to be rendered on the JSP page.
- The session beans are responsible for maintaining state about a user's session in the entitlements UI.
- The backing (helper) beans are wrappers around the RBAC API. The helper beans provide all required caching logic.
- The RBAC API is built on top of the BLM API. Access to the RBAC API is through the helper beans; you do not call the RBAC API directly. The RBAC API is responsible for providing a clean interface to manage entitlements based on the NIST hierarchical RBAC model. It currently supports role management, permission management, permission set management, and provides reporting functionality based on the hierarchical rbac model.

Entitlements UI Application Objects

There are five packages in the Entitlements UI:

- `com.bea.security.entitlements.admin.beans`
- `com.bea.security.entitlements.admin.rbac`
- `com.bea.security.entitlements.admin.resources`
- `com.bea.security.entitlements.admin.util`
- `com.bea.ales.persistence`

These packages are described in the sections that follow.

Entitlements UI Beans Package

The beans package is responsible for providing application state and session management functionality.

Table 6-1 Entitlements UI Beans

Class Name	Description
Distributable	Mandates the distribution contract for the EUI functionality, so that the JSP or the external clients invoke methods to control distribution.
Transactable	Mandates transaction contract for the EUI functionality, so that the JSP or the external clients invoke transaction functionality. Changes stored in the current session for a given transaction are persisted or rolled back based on the user operations.
SecurityBean	Provides hooks for authorization checks. These hooks append resource information that provides the context in which a resource resides. For example, the `canViewResource` method appends the `componentId` to the end of the resource: <the currently selected node>/<the name of the tab selected>

Table 6-1 Entitlements UI Beans

Class Name	Description
SelectedPermissionSets	Used to provide state information for the permission set popup dialogue box. This object handles the special functionality: when a parent check box is checked, all child check boxes are checked, and if a child checkbox is unchecked, the parent is also unchecked.
SessionState	Main monolithic application state bean. This class provides references to the current set of selected elements in the entitlements UI. It's also responsible for main application logic like constructing the navigation tree and policy distribution.

Entitlements UI RBAC Package

The backing (helper) beans are wrappers around the RBAC API. The helper beans provide all required caching logic.

Table 6-2 RBAC Package

Class Name	Description
AllSODConflictsReport	Provides implementation to provide a comprehensive report to all conflicting SODs (separation of duties).
AttributableEntitlementElement	Provides base functionality for elements with attributes requirements.
AttributableEntitlementNode	Provides core functionality for all nodes that could have attributes associated to its definition.
AttributeElement	This is a helper class used to represent an instance of an attribute
BaseTableElement	Provides base functionality for an EUI table element.
BaseTreeNode	Provides base functionality for most tree nodes.
DataTableElement	This is an instance of a row in a data table. It's used to support making a data table selectable
EUIMetaObjectNode	Facilitates custom object creation of a selectable tree node. MetaObjects to extend the EUI functionality are configured via metaobject_mappings.properties. These files would be loaded during startup time.

Table 6-2 RBAC Package

Class Name	Description
GroupItem	Represents a selected group associated with current identity directory node in EUI tree. The class will be used by the entitlements front end for group management.
IdentityDirectoryNode	Represents an identity directory node in EUI tree. The node will wrap RBAC Identity Directory Node used by the entitlements front end.
IdentityNode	Extends BaseTreeNode.
MembershipRuleElement	Represents an instance of a membership rule. Wraps the RBAC_MembershipRule object.
NavigationTreeNode	Base node for navigation tree elements.
PermissionElement	Represents an instance of a permission object. Wraps the RBAC_Permission object.
PermissionNode	Represents the Permission Node on the nav tree. It's what is used to build the permissions table. Contains a list of permissionElement objects.
PermissionPermissionSet MembershipReport	Provides reporting on what permissions are a member of a permission set.
PermissionSetElement	Represents an instance of a permission set. This object just contains data about Permission Sets
PermissionSetMembership Report	Accessor class to retrieve the permission set membership report.
PermissionSetNode	Represents a Permission Set node on the navigation tree. Wraps the RBAC_PermissionSet object.
ReportingNode	The base reporting object. All reporting elements are based on this class.
ResourceNode	Represents a node on the resource tree used for adding resources to permissions.
RoleElement	Represents an instance of a role's data for display in the UI.
RoleNode	Represents a role on the navigation tree. Wraps the RBAC_Role object.
RolePermissionSetReport	Used to facilitate the All Permission Sets assigned to a role report.
RolePermissionsReport	Used to facilitate all Permissions assigned to a role report.

Table 6-2 RBAC Package

Class Name	Description
RoleSubjectMembersReport	Used to facilitate all subjects that are a member of a role report.
RolesWithPermissionReport	Used to facilitate all roles that have a particular permission report.
RolesWithPermissionSetReport	Used to facilitate all roles that have a particular permission set report.
SODElement	Used to represent the instance of an SOD rule. Has a reference to the rolenode the SOD element is based on
SODNode	represents the SOD node on the nav tree.
SubjectComparitor	Used to compare subjects for sorting.
SubjectElement	Represents an instance of a subject for UI display purposes.
SubjectPermissionSetsReport	Used to report all subjects associated with a permission set.
SubjectPermissionsReport	Used to report all subjects associated with a permission
SubjectRoleMembershipReport	Used to report the set of roles a subject has.
SubjectsWithPermissionReport	Used to report the set of permissions a subject has.
SubjectsWithPermissionSetReport	Used to report the set of permission sets a subject has.
TreeNodeWithSelection	Base TreeNode class used to support the permission set selection popup window.
UserItem	Represents a selected user associated with current identity directory node in EUI tree. The class will be used by the entitlements front end for user management.

Utils Package

The Utils package contains helper classes for converting objects needed by JSF to all the wrapper classes around the RBAC_API.

Table 6-3 Utils Package

Class Name	Description
DirectoryKeyConverter	This is used to convert a directory object into a string – used for by select many list boxes and drop down lists.
IdentityGroupItemKeyConverter	Converts JSF faces objects to ALES format.
IdentityUserItemKeyConverter	Converts JSF faces objects to ALES format.
PermissionKeyConverter	Responsible for converting permission objects to strings and back to permission objects.
SubjectKeyConverter	Used to convert a subject to a string and back to a subject.
Utils	This is a generic helper class that provides functionality like looking up a bean in the user's session.

Persistence Package

This API provides access to group objects and their attributes.

Table 6-4 Persistence Package

Interface/Class	Description
MetaObject	This interface is for getting information about an object in a hierarchy. The object implemented by this interface is a node in a hierarchy. It may have a parent, children, and attributes.
MetaObjectFactory	The factory class to get MetaObject.
MetaObjectImpl	Implementation of interface MetaObject. It will be mainly used to manipulate common object.
Permission	Permission object that is associated with privilege and resource directly.

Table 6-4 Persistence Package

Interface/Class	Description
PermissionSet	The permission set object, which may contain other permission set and permissions.
Role	The Role object. Besides having other Roles as its member, it can also have Permission and PermissionSet as its member.

Extending the Entitlements Management Tool: Main Steps

Un-jar Entitlements Management Tool Web Archive File

Use the regular jar command to un-jar the *BEA_HOME*\ales26-admin\entitlements*<container>*\entilementsadministration.war file to a directory of your choice, in a manner similar to the following:

```
jar –xvf entilementsadministration.war destination_dir
```

Create a metaobject_mappings.properties Configuration File Under WEB-INF/config

When you unjar the `entilementsadministration.war` file, a sample *<destination_dir>/WEB-INF/config/metaobject_mappings.properties* file is created.

The `metaobject_mapping.properties` is located in the `WEB-INF/config` directory by default. Alternatively, you can override this with a system property setting similar to the following. (This has to be directory path only; not a path with the file name.)

```
-Deui.metaobject.home=<fully qualified directory of the
metaobject_mapping.properties file>
```

You must create an instance of this file for **each** top-level node you create when extending the Entitlements Management Tool.

When extending the Entitlements Management Tool, you cannot add to an existing root node in the hierarchy, you can only create a new root node and then add to it. To do this, you extend the `EUIMetaObjectNode` object. The class you create becomes the link between your Java application and the Entitlements Management Tool.

A key parameter when extending `EUIMetaObjectNode` is the node type. When you create a new node, you overload the node type in all of the constructors so that the Entitlements Management Tool can render the desired content on the main page.

In this mapping file, you reference those node types in the `MetaObjectType` field, as well as your custom classes that implement `EUIMetaObjectNode` and the names of the root nodes.

Note: The values you specify in the mapping file must be an exact, case-sensitive match to those specified in your Java application.

Consider the example shown in Listing 6-1 and note the comments in **bold**.

Listing 6-1 Sample Metaobject_mappings.properties File

```
# Define custom type
MetaObjectType1=Subscription
# Configure custom class implementing EUIMetaObjectNode
Subscription.MetaObjectImpl=com.metanode.test.SubscriptionNode
# Configure Name of the root node
Subscription.MetaObjectRootName=All_Subscriptions


MetaObjectType2=Magazine
Magazine.MetaObjectImpl=com.metanode.test.MagazineNode
Magazine.MetaObjectRootName=Magazine_Management


MetaObjectType3=License
License.MetaObjectImpl=com.metanode.test.LicenseNode
License.MetaObjectRootName=License_Management
```

An InheritanceModel value of 0 uses TOP_TO_BOTTOM (parent to child) inheritance. A value of 1 use BOTTOM_TO_TOP (child to parent) inheritance. It defaults to 0.

```
# Configure Inheritance Model
```

```
#Inheritance model 0 implies TOP_TO_BOTTOM
#Inheritance model 1 implies BOTTOM_TO_TOP
License.MetaObjectInheritanceModel=0
```

Create Custom Implementation Node to Extend EUIMetaObjectNode

Create one or more custom implementation nodes that extend the `EUIMetaObjectNode` class. Package these classes under *<destination_dir>*`/WEB-INF/classes` as a fully-qualified class, or in *<destination_dir>*`/WEB-INF/lib` as a jar.

When you create a new node, you overload the node type in all of the constructors so that the Entitlements Management Tool can render the desired content on the main page.

Note: The values you specify in your Java Application must be an exact, case-sensitive match to those specified in your mapping file.

Consider the example shown in Listing 6-2 from `SubscriptionNodejava`.

Listing 6-2 Extending EUIMetaObjectNode

```
package com.metanode.test;
:
public class SubscriptionNode extends EUIMetaObjectNode {
       public static final String SUBSCRIPTION_NODE_TYPE = "Subscription";
       public SubscriptionNode (MetaObject pset, Boolean isLeaf) {
              super (SUBSCRIPTION_NODE_TYPE, pset, isLeaf.booleanValue());
       }
       public SubscriptionNode (TreeNodeWithSelection parentNode,
                                          MetaObject pset,
                                          Boolean isLeaf)  {
              super (SUBSCRIPTION_NODE_TYPE, parentNode, pset,
isLeaf.booleanValue ());
       }
```

```
        public String getNodeType () {
                return SUBSCRIPTION_NODE_TYPE;
        }
}
```

In the Subscription example described in "Example of Extending the Entitlement UI" on page 6-34, there are three custom implementation nodes that extend the `EUIMetaObjectNode` class: `LicenseNode`, `MagazineNode`, and `SubscriptionNode`.

The example utilizes three node files (`LicenseNode.java`, `MagazineNode.java`, and `SubscriptionNode.java`) and three backing bean files (`LicenseBean.java`, `MagazineBean.java`, and `SubscriptionBean.java`).

The example is designed such that the backing bean code handles the actions of subscription management (for example, `getSelectedOwnedMagazines()` in `SubscriptionBean.java`) and the node code (for example, `SubscriptionNode.java)` contains mostly the constructors.

`SubscriptionNode.java` does include one other method, `getAttribute()` to get the attribute element of a specific attribute. You could instead create a `SubscriptionNode.getMagazines()` or `subScription.getDuration()` method to make the use more intuitive.

Create Custom JSPs

Create new JSPs to represent your custom entitlements model. Consider the sample section of the `subscriptions.jsp` file shown in Listing 6-3.

Pay particular attention to the code section in **`bold`**.

Listing 6-3 subscriptions.jsp

```
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f" %>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h" %>
<%@ taglib uri="http://myfaces.apache.org/tomahawk" prefix="t" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%--
```

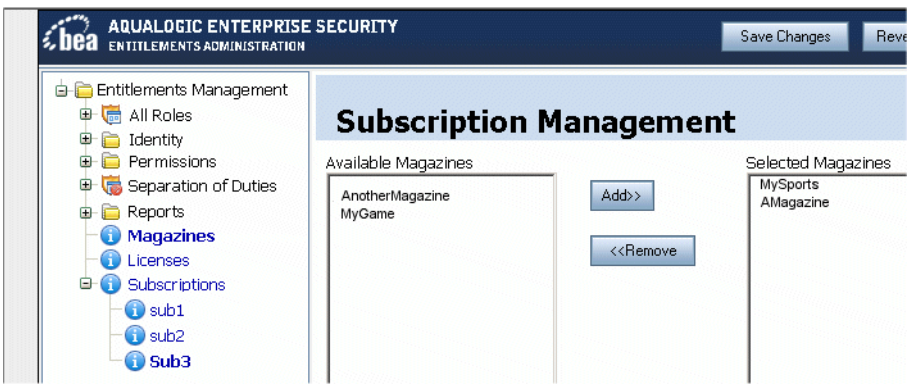
```
    This page shows all permisions of the selected role, includes direct,
inherited and denied

--%>

<f:loadBundle
basename="com.bea.security.entitlements.admin.resources.messages"
var="bundle"/>

<f:subview id="subscriptions">

<c:if test="${sessionState.selectedNode.description!='All
Subscriptions'}">

<h:form id="subscriptionForm">

       <t:panelGrid width="100%" columns="1"
styleClass="main-content-panel" rowClasses="twoRowLayout-50_50"
align="center">

              <t:panelGroup>

                     <t:panelGrid width="100%"
columnClasses="agrolesides,agrolesides,agrolecenter,agrolesides"
columns="3" align="center">

                           <t:panelGroup>

                                  <h:outputText
value="#{bundle.AvailableMagazines}"/>

                                  <f:verbatim><br/>

                                  </f:verbatim>

                                  <%-- get all available magazines and
show in list box --%>

                                  <t:selectManyListbox id="allmagazines"
value="#{subscriptionBean.selectedAvailableMagazines}"
styleClass="selectBox" size="17"
converter="com.metanode.test.MagazineConverter">

                                         <f:selectItems
value="#{subscriptionBean.availableMagazines}"/>

                                  </t:selectManyListbox>
```

```
                                    </t:panelGroup>
                                    <t:panelGroup>
:
:
```

Consider how the **bold** code maps to Figure 6-6:

- `AnotherMagazine` and `MyGame` represent the list of available magazines (**`subscriptionBean.availableMagazines)`**.
- If the user were to select `AnotherMagazine`, it would comprise the list of selected, available magazines (**`subscriptionBean.selectedAvailableMagazines).`**

Figure 6-6 Available and Selected-Available Magazines



Modify Existing Navigation and Main JSP Files

Modify the Entitlement Management Tool `<destination_dir>/navigation.jsp` file to add a new facet for **each** custom node. This page defines the navigation tree, and all of the supported node type should have a facet here.

Inside a facet, two panelGroup objects should be defined. One will display when `sessionState.disableMainButtons` is false. The other panelGroup will display when `sessionState.disableMainButtons` is true.

Listing 6-4 shows the Subscription facet for the modified `navigation.jsp` file.

Listing 6-4 Subscription Facet in navigation.jsp

```
<f:facet name="Subscription">
      <t:panelGrid id="c" columns="2" cellpadding="2" cellspacing="0"
width="100%" styleClass="treeTable">
          <t:graphicImage value="../images/wlp-info-16.gif"/>
          <t:panelGroup
rendered="#{sessionState.disableMainButtons==false}">
            <t:commandLink
actionListener="#{sessionState.processSelectAction}"
                value="#{node.description}"
                styleClass="bold" rendered="#{t.nodeSelected == true}"/>
           <t:commandLink
actionListener="#{sessionState.processSelectAction}"
                value="#{node.description}"
                rendered="#{t.nodeSelected == false}"/>
          </t:panelGroup>
          <t:panelGroup
rendered="#{sessionState.disableMainButtons==true}">
               <h:outputText value="#{node.description}" styleClass="bold"
rendered="#{t.nodeSelected == true}"/>
               <h:outputText value="#{node.description}"
rendered="#{t.nodeSelected == false}"/>
          </t:panelGroup>
      </t:panelGrid>
</f:facet>
```

Modifying main.jsp

Modify the Entitlement Management Tool *<destination_dir>*/main.jsp file. This page is the main page of entitlements administration, and includes the JSPs to which the Tool will navigate.

Make the following changes:

1. When you created a new node, you overloaded the node type in all of the constructors so that the Entitlements Management Tool could render the desired content on the main page. You supply this nodeType in main.jsp.

```
<h:outputText value="#{bundle.SubscriptionManagement}"
rendered="#{sessionState.selectedNode!=null &&
sessionState.selectedNode.nodeType == 'Subscription'}"
styleClass="RoleName"/>
```

2. The header facet displays the title and image associated with a given main page. Update as needed in main.jsp

3. Add details for the custom node type.

```
<%-- Add details jsp here for custom node type --%>

<%-- defines the detail page for magazine, subscription and license --%>

<h:panelGroup id="Magazine">

       <jsp:include page="magazines.jsp" />

</h:panelGroup>

<h:panelGroup id="Subscription">

       <jsp:include page="subscriptions.jsp" />

</h:panelGroup>

<h:panelGroup id="License">

       <jsp:include page="licenses.jsp" />

</h:panelGroup>
```

Modify the JSF Configuration File

As described in Developing Web Applications with JavaServer Faces, an application configuration resource file, `faces-config.xml`, is used to define your managed beans, validators, converters, and navigation rules.

Modify the Entitlement Management Tool *<destination_dir>*`/WEB-INF/config/faces-config.xml` file to add the relevant information for your application.

For example, Listing 6-5 shows the changes made to *<destination_dir>*`/WEB-INF/config/faces-config.xml` in support of the extension example.

Listing 6-5 Modified faces-config.xml File

```
Define converters
<converter>
        <converter-id>com.metanode.test.MagazineConverter</converter-id>
        <converter-class>com.metanode.test.MagazineConverter</converter-cl
ass>
</converter>


<converter>
        <converter-id>com.metanode.test.LicenseUserConverter</converter-id
>
        <converter-class>com.metanode.test.LicenseUserConverter</converter
-class>
</converter>
:
Define Managed Beans
<managed-bean>
        <managed-bean-name>magazineBean</managed-bean-name>
```

```
        <managed-bean-class>com.metanode.test.MagazineBean</managed-bean-c
lass>
        <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
        <managed-bean-name>subscriptionBean</managed-bean-name>
        <managed-bean-class>com.metanode.test.SubscriptionBean</managed-be
an-class>
       <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
<managed-bean>
       <managed-bean-name>licenseBean</managed-bean-name>

<managed-bean-class>com.metanode.test.LicenseBean</managed-bean-class>
       <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

Define Navigation Rules

```
<navigation-rule>
       <from-view-id>/pages/newmetaattribute.jsp</from-view-id>
       <navigation-case>
              <from-outcome>Success</from-outcome>
              <to-view-id>/pages/modifymetaattribute.jsp</to-view-id>
       </navigation-case>
</navigation-rule>
<navigation-rule>
       <from-view-id>/pages/modifymetaattribute.jsp</from-view-id>
       <navigation-case>
              <from-outcome>close</from-outcome>
```

```
                <to-view-id>/pages/closepopup.jsp</to-view-id>
        </navigation-case>
        </navigation-rule>

<navigation-rule>
        <from-view-id>/pages/newMagazine.jsp</from-view-id>
        <navigation-case>
                <from-outcome>close</from-outcome>
                <to-view-id>/pages/closepopup.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
<navigation-rule>
        <from-view-id>/pages/subscriptions.jsp</from-view-id>
        <navigation-case>
                <from-outcome>success</from-outcome>
                <to-view-id>/pages/subscriptions.jsp</to-view-id>
        </navigation-case>
        <navigation-case>
                <from-outcome>failed</from-outcome>
                <to-view-id>/pages/subscriptions.jsp</to-view-id>
        </navigation-case>
</navigation-rule>
<navigation-rule>
        <from-view-id>/pages/newLicense.jsp</from-view-id>
        <navigation-case>
                <from-outcome>close</from-outcome>
                <to-view-id>/pages/closepopup.jsp</to-view-id>
```

```
        </navigation-case>
    </navigation-rule>
```

Re-jar Entitlements Management Tool Web Archive

Use a command similar to the following to re-jar the Entitlements Management Tool web archive with your changes from the `destination_dir` you specified in "Un-jar Entitlements Management Tool Web Archive File" on page 6-13.

```
jar -cvfM entilementsadministration.war *.*
```

Redeploy Entitlements Management Tool Web Archive on Admin Server

To redeploy the Entitlement Management Tool on the ALES Admin server, use the tool to overwrite the existing `entitlementsadministration.war` file. Click redeploy under Deployments->WebApplications->Redeploy.

Using Custom Data for Access Control

Adding custom data to the ALES Entitlement Management tool involves two steps:

- Adding the new objects to the navigation tree, as described in "Create Custom Implementation Node to Extend EUIMetaObjectNode" on page 6-15.
- Adding custom JSPs to manage data for the new objects, as described in "Create Custom JSPs" on page 6-16.

Once these steps are complete, the custom data can be managed in the ALES Entitlements Management tool and stored in the ALES database.

There are two options to tie the custom data to an ALES policy:

- Use an attribute retriever to get a custom data value that will be used in the constraint portion of an ALES policy, as described in "Using an Attribute Retriever to Get a Custom Data Value" on page 6-25.
- Use an evaluation function to execute logic using the custom data that returns a true or false result, as described in "Using an Evaluation Function" on page 6-29.

Using an Attribute Retriever to Get a Custom Data Value

As described in Attribute Retriever, attribute retrievers are used by ASI Authorization and ASI Role Mapping providers to retrieve attributes for use by AquaLogic Enterprise Security authorization and role mapping providers.

For example, using the subscription model we can write the following ALES policy:

```
grant(any, //onlineContent/Newsmagazine, //role/Everyone) if (Newsmagazine
in magazines)
```

where `magazines` is a custom attribute retriever that returns a list of magazines by examining the user's current subscriptions.

The attribute retriever uses the "Persistence Package" on page 6-12 to get custom runtime information.

Note: See the Readme for important information about enabling metadirectory support.

Consider the example attribute retriever shown in Listing 6-6. Pay particular attention to the code in **`bold`**.

Listing 6-6 MagazinesAttributeRetriever

```
package com.bea.security.providers.authorization.asi;


import java.text.SimpleDateFormat;

import java.util.ArrayList;

import java.util.Calendar;

import java.util.Date;

import java.util.HashMap;

import java.util.Iterator;

import java.util.Map;

import java.util.Set;


import javax.security.auth.Subject;
```

```
import org.apache.log4j.Logger;

import weblogic.security.service.ContextHandler;
import weblogic.security.spi.Resource;

import com.bea.ales.persistence.MetaObject;
import com.bea.ales.persistence.MetaObjectFactory;
import com.bea.ales.rbac.AttributeElement;
import com.bea.ales.rbac.InvalidTypeException;
import com.bea.ales.rbac.ObjectNotFoundException;
import com.bea.ales.rbac.QueryType;
import
com.bea.security.providers.authorization.asi.ARME.evaluator.RequestHandle;

/**
 * The custom attribute retriever of subscription model sample
 *
 */
public class MagazinesAttributeRetriever implements AttributeRetrieverV2,
InitializationShutdownFunction {
    static final String ATTRIBUTE_NAME = "subscriptions";
    private static final String LICENSE = "license";
    private static final String SUBSCRIPTION = "Subscription";
    private static final String MAGAZINES = "magazines";
    private static final String DURATION = "duration";
    private static final String DATE_PATTERN = "MM/dd/yyyy";
    private static final String COMMA = ",";
    private static final String LEFT_SQUARE_BRACKETS = "\"";
```

```
    private static final String RIGHT_SQUARE_BRACKETS = "\"";
    private static final String LICENSE_PREFIX = "\"";
    private static final String LICENSE_POSTFIX = "\"";

    private static Logger LOGGER =
Logger.getLogger(MagazinesAttributeRetriever.class);

    private MetaObject subScriptionAll = null;
    private Map metaProperties = null;

    /**
     * The method defined in the interface
     * ARME will call this method to initilize this retriever
     */
    public void init(Map map){
        Set keys = map.keySet();
        Iterator keysit = keys.iterator();
        while(keysit.hasNext()) {
        Object key = keysit.next();
        LOGGER.debug("init() Key=" + key + ", value=" + map.get(key));
        }
        //Use the map to initilize the metaobject factory so that it will
connect the database
        this.initMetaProperties(map);
        MetaObjectFactory.initilize(map);
        try {
            subScriptionAll =
MetaObjectFactory.getInstance().getRootObject(SUBSCRIPTION);
```

```
            } catch (InvalidTypeException e) {
                LOGGER.error(e.getMessage(), e);
                subScriptionAll = null;
            }
        }
:
:
/**
     * Add a subscription to the result.
     * @param entry An entry of license
     * @param result The result list to return to ARME
     */
    private void addSubScription(Map.Entry entry, ArrayList result){
        try {
        //Get the 'magazine' and 'duration' attribute of the subscription
            MetaObject subScription =
subScriptionAll.getObjectByName((String)entry.getKey(), SUBSCRIPTION);
         AttributeElement magazines = subScription.getAttribute(MAGAZINES,
QueryType.DIRECT);
           AttributeElement duration = subScription.getAttribute(DURATION,
QueryType.DIRECT);
            if (LOGGER.isDebugEnabled()){
                LOGGER.debug("Subscription: "+subScription.getName());
                LOGGER.debug("magazines: "+magazines);
                LOGGER.debug("duration: "+duration);
```

Note: As an alternative to calling `subScription.getAttribute(MAGAZINES, QueryType.DIRECT)` or `subScription.getAttribute(MAGAZINES, QueryType.DIRECT)`, you could instead create a `subScription.getMagazines()` or `subScription.getDuration()` method to make the use more intuitive.

Using an Evaluation Function

As described in evaluation function, you can write an evaluation function to make additional authorization request data available, and therefore allow a more complex attribute evaluation to be performed. The method is invoked while the policy contains a custom evaluation function with a matching name. For example:

```
grant(any, //onlineContent/Newsmagazine, //role/Everyone) if
rbac_eval_magazine(session,args,subject,roles,resource,contextHandler);
```

where `rbac_eval_magazine`() is the custom evaluation function name. You must register one evaluation class that includes the `rbac_eval_magazine()` method.

Consider the evaluation function shown in Listing 6-7.

Note: This function uses the attribute retriever shown in Listing 6-6.

Listing 6-7 MagazinesEvaluator Function

```
import com.wles.util.AttributeElement;


/**
 * The custom evaluation function of subscription model sample
 *
 */cd
public class MagazinesEvaluator {


    private static final String REQUESTED_RES = "sys_obj_q";
    //private static final String MAGAZINES = "subscription_magazines";
    private static final char SLASH = '/';
```

```
    private static Logger LOGGER =
Logger.getLogger(MagazinesEvaluator.class);
    /**
     * The method to evaluate whether a request to some magazine is allowed
     * @param session
     * @param args
     * @param subject
     * @param roles
     * @param resource
     * @param contextHandler
     * @return
     * @throws MissingAttributeException
     */
   public boolean rbac_eval_magazine(RequestHandle session, Object[] args,
Subject subject, Map roles, Resource resource,
          ContextHandler contextHandler) throws MissingAttributeException {

        LOGGER.debug("Entering rbac_eval_magazine...");
        String currentMagazine = null;
        AttributeElement magazines = null;
        try
        {
        //Get the resource that are requested
            AttributeElement resAttr =
session.getAttribute(REQUESTED_RES,false);
            String res = (String)resAttr.getValueAs(String.class);
            if (res.charAt(res.length()-1) == SLASH){
                res = res.substring(0, res.length()-1);
```

```
            }
            //get the last part of request url which is the magazine name
            currentMagazine = res.substring(res.lastIndexOf('/')+1);
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(),e);
            throw new MissingAttributeException("missing attribute:
"+REQUESTED_RES,REQUESTED_RES);
        }
        try
        {
          //Check if the requested magazine is in the list which are allowed
        //to be accessed according to the license
        magazines =
session.getAttribute(MagazinesAttributeRetriever.ATTRIBUTE_NAME,false);
            if (magazines != null){
                ArrayList mags =
(ArrayList)magazines.getValueAs(ArrayList.class);
                Iterator iter = mags.iterator();
                while (iter.hasNext()){
                    String magazine = (String)iter.next();
                    if (currentMagazine.equalsIgnoreCase(magazine)){
                        LOGGER.debug("Exiting rbac_eval_magazine with the
result true");
                        return true;
                    }
                }
```

```
            }
        }
        catch (Exception e)
        {
            LOGGER.error(e.getMessage(),e);
            throw new MissingAttributeException("Missing attribute:
"+MagazinesAttributeRetriever.ATTRIBUTE_NAME,
MagazinesAttributeRetriever.ATTRIBUTE_NAME);
        }
        LOGGER.debug("Exiting rbac_eval_magazine with the result false");
        return false;
    }
}
```

Clone and Move Operation for Custom Node

You can modify the existing *<destination_dir>*/pages/cloneNode.jsp and moveNode.jsp files to support the clone and move operations, as show in Listing 6-8.

Note: You would typically need to change only the nodeType and the Facet in this example for each custom node defined in "Modify Existing Navigation and Main JSP Files" on page 6-18.

Listing 6-8 Adding Clone and Move Operations to JSP

```
<c:if test='${sessionState.selectedNode != null && sessionState.nodeType ==
"Subscription"}'>
<t:tree2 id="nTree" value="#{sessionState.rootMetaTree}" var="node"
clientSideToggle="false" preserveToggle="false" varNodeToggler="t">
       <f:facet name="Subscription">
```

```
            <t:panelGrid id="c" columns="2" cellpadding="2"
cellspacing="0" width="100%" styleClass="treeTable">

            <t:graphicImage
value="../images/wlp-folder-rolemapper-16.gif"/>

            <t:panelGroup>

            <t:commandLink onmousedown="cancel=false;"
actionListener="#{sessionState.processSelectDestinationNodeAction}"
value="#{node.description}" immediate="true" styleClass="bold"
rendered="#{t.nodeSelected == true}"/>

          <t:commandLink onmousedown="cancel=false;"
actionListener="#{sessionState.processSelectDestinationNodeAction}"value="
#{node.description}" immediate="true" rendered="#{t.nodeSelected ==
false}"/>

            </t:panelGroup>

            </t:panelGrid>

     </f:facet>

      </t:tree2>

</c:if>
```

Debugging Techniques and Problem Isolation

If you are having trouble with your Entitlement Management Tool extension, please consider the following possible problem scenarios:

Table 6-5 Problem Isolation

Problem	Solution
Custom nodes fail to load	Verify the meta object properties files entries. Implementation class available under Web-INF/classes or WEB-INF/lib as a jar. Type returned by the custom classes representing the node match MetaObject property file should be available via either the system property or under /WEB-INF/config directory. Check Weblogic server console for NoClassDefFoundError
Blank display pages on the Entitlements Management Tool console.	There could be an issue with the scripts. May need to turn on JSF debugging.
Any authentication and authorization failures	Use the log4j debugger as for other ALES components.

Example of Extending the Entitlement UI

ALES 2.6 includes an example that shows how to extend the Entitlements UI in a custom entitlement model. The example shows how to extend a generic object, how to create custom JSP pages and backing beans, and how to integrate them with the entitlements UI.

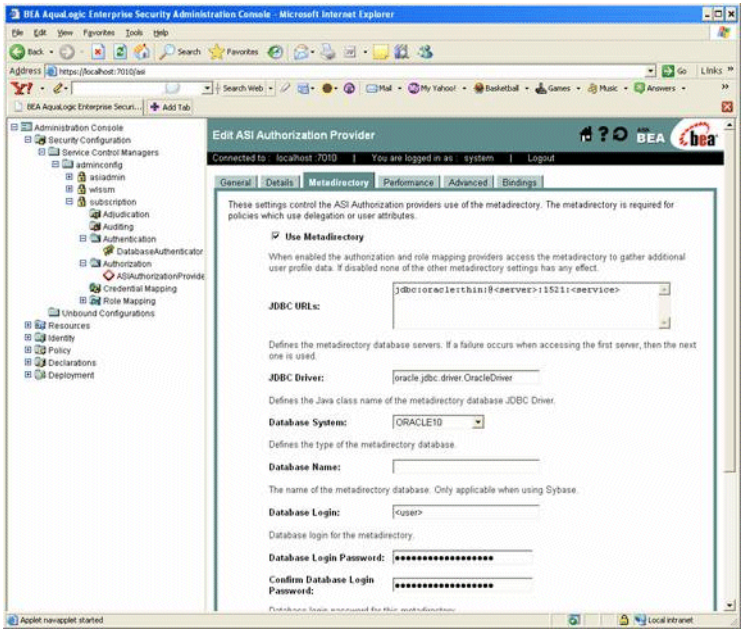
You can use this example as a guide for extending entitlements UI.

Follow the Instructions in the Readme

The example is available in *BEA_HOME*\bea\ales26-admin\entitlements\example and includes an extensive Readme file. The Readme describes how to build the example and deploy the web application.

The Readme includes instructions on how to use the ALES Admin console to set the Authentication provider database properties and how to set the metadirectory for ASIAuthorizer, as shown in Figure 6-7.

Figure 6-7 ASI Authorization Provider use of Metadirectory



Importing and Exporting Policy Data

The AquaLogic Enterprise Security Administration Server includes two tools to assist you in managing the contents of the policy store: an import tool and an export tool. Using these tools you can perform the following tasks:

- Define your policy data in text files that are external to the Administration Server and import those files to a policy store on any Administration Server.
- Export policy data from an existing policy store on an Administration Server and import that policy data to a policy store on any Administration Server.
- Export policy data from an existing policy store on an Administration Server, install a newer version of the server software, and re-import the policy data into the upgraded server.

For information about writing policy files, see "Advanced Topics" on page 4-1. The following sections describe how to use the policy import and export tools:

Importing Policy Data

This section provides instructions and information on how to import policy data to the policy store. It covers the following topics:

Policy Import Tool

Note: As of AquaLogic Enterprise Security version 2.5, policy loading is now transactional: all policies are loaded, or none. In addition, the BLMContextManager API has been updated to include transactional methods.

The Policy Import tool is a Java utility that provides an alternate method of entering policy data (rather than through the Administration Console). The main purpose of using this tool is to reduce the amount of manual data entry required. The Policy Import tool lets you load policy data into the database, distribute that policy, and remove policy data from the database. The Policy Import tool reads and imports policy data that is stored as text using non-XML, easy to read format. Each policy element is stored in a separate file, referred to as a policy file. For information on the specific format of these policy elements, see Chapter 4, "Advanced Topics."

The Policy Import tool has the following features:

- Multi-threaded architecture–Allows for more efficient policy loading.
- Separation of policy elements–Loads multiple files with each file corresponding to one policy element.
- Optimized–Fast import of large policies during initial import.
- Policy Distribution–After importing, use the Policy Import tool to distribute the policy.

  Note: Before you can use the Policy Import tool to distribute policy, you must configure the distribution file and enable the policy distribution feature in the distribution configuration file of the policy loader.

- Removing Policy–You can also use the Policy Import tool to remove policy elements from the database.

Note: When running the Policy Import tool on a large policy, the number of records processed may not be synchronized. If multiple threads are used to import the data, when one thread completes before the other cannot be determined. If the threads are set too high, a message may appear indicating that the number of records processed is not synchronized. This is normal and is not a problem for the Policy Import tool.

For a description of the content of policy files, see Chapter 4, "Advanced Topics."

When exporting the policy, the configuration resources are saved to the following files: `object_config` and `objattr_config`. These two files are not loaded by the policy loader by default. If you want to load the configuration resources, you need to create a directory and copy `object_config`, `objattr_config`, and `binding` into that directory. Rename `object_config`

to `object` and `objattr_config` to `objattr`. Then you can configure the policy loader to load these files into this new directory.

Configuring the Policy Import Tool

The Policy Import tool relies on the configuration file for information on how to load the policy files. You only need to modify the configuration file if you the change the location of the policy files or you want to change some configuration options. The `Domain` parameter is required for successful import. The Policy Import tool uses default values for the other parameters, which are all optional.

This section covers the following topics:

Setting Configuration Parameters

Each configuration parameter has the following format:

`<Parameter> <Value>`

The file paths in the configuration file depend on the directory from which you run the Policy Import tool. You may use the full path filename to avoid directory dependency. Spaces are allowed between parameters and between new lines. Parameter names are case insensitive. Table 7-1 lists the parameters you need to configure for the Policy Import tool.

To create the configuration file (see Listing 7-1 for a complete sample), you need a text editor such as Notepad. Create the file by entering the necessary parameters and parameter values. The following sections describe the contents of a sample configuration file, with a detailed explanation of each parameter and its default value.

Enter the following parts of the configuration file in the format described. These are only sample entries. Your entries depend on the names you create and where your files are stored. An *italics* font is used here to represent variables that you replace with your own parameter names. You do not need to list the parameters in the configuration file in this order.

There is a sample of a Policy Import configuration file named `policy_loader_sample.conf` located in the .../`examples/policy` directory. You can modify this file for your own use. BEA recommends that you use this file as a template and customize it for your particular needs.

Note: The configuration parameters are listed in alphabetical order in Table 7-1. This is not the order in which they are listed in the `policy_loader_sample.conf` file.

Table 7-1 Configuration Parameters

Parameter	Description
Action	Indicates the Action that the Policy Import tool will perform. Supported values are LOAD and REMOVE (case insensitive). REMOVE = Unloads the specific policy from the database. ADD = Loads the specific policy data into the database.
ApplicationNode	Specifies the application node that holds the administration policy. If this parameter is commented out, the default value of admin is used.
BLMContext Retries	Specifies the number of times retries should take place. If the ALES Administration Console server is still starting up, then you need to retry the BLM API Authentication. In most cases the ALES Administration Console server is always running. Default: 100.
BLMContext Interval_ms	Specifies the amount of time (in milliseconds) to wait between context retries. DEFAULT: 100ms.
BulkSize	Specifies the number of records to send at one time in a thread. Default: 200. **Note:** When there are multiple threads importing policy data, each processing a number of records, the number of records processed may result in an "out-of-sync" message. However, it does not harm the data when importing the policy. The policy import tool switches to single thread when importing some policy elements, such as resources and declarations, as the later records have dependency on earlier records.
ConsoleDisplay	Specifies whether to hide console interaction or not (yes/no). If you want to run the policy loader in the background as a batch process, set to no. Default: yes no = Error messages are not displayed on the console and the user is requested to enter their Username and Password if they are missing in the configuration file. yes = Error messages are displayed on the console. This parameter must be enabled if you want to type in your password on the command prompt, rather then use the one specified in the password.xml or in the configuration file.
Debug	Specifies whether you want to log debug information. Default: 0 0 = Does not log debug information. 1 = Sends debug information to the file defined by: ErrorLogFile.

Table 7-1 Configuration Parameters (Continued)

Parameter	Description
`Domain`	Specifies the Enterprise domain name, as assigned during the installation of the Administration Application. Default: `asidomain`. This parameter is required.
`ErrorLogFile`	Specifies the name of error log file. This file is produced if the Importing Tools fails while attempting to load a set of policy files. It contains error messages that describe the failures to assist you in correcting the errors. Default: `error.log`.
`Mode`	Specifies the mode of operation the Policy Import tool. Values are `INITIAL` or `RECOVER` (case insensitive). Use INITIAL mode the first time you run the Import Policy Tool to load a set of policy files. If you encounter errors in the initial load attempt, check the `ErrorLogFile` for a description of the error, correct the errors in the generated error file(s) (an error file is produced for each policy file that fails), and rerun the Import Policy Tool again, but this time in the RECOVER mode. This way the tool only attempts to load the generated error files. If the tool fails again, fix the errors, and run it again in RECOVER mode. Repeat until no errors are encountered. **Note:** This parameter can also be passed in as a command-line parameter `-recover` or `-initial`. Values for this parameter on the command line override values specified in the configuration file.
`PasswordFile`	Specifies an encrypted password file. To set up a password file, use the `asipassword` utility. This utility prompts you for the alias (username) and the password of the user trying to import the policy and then saves the encrypted password in the `password.xml` file. Default: `../ssl/password.xml`. For more information, see asipassword in the *Administration Reference*.
`PasswordKey File`	Specifies a private key used to decrypt the password stored and encrypted in the `password.xml` file. To set up a password file, use the `asipassword` utility. Default: `../ssl/password.key`. For more information, see asipassword in the *Administration Reference*.
`Policy DirectoryPath`	Specifies the directory path from which to import policy files. For example: `../examples/policy`. The path may be relative. Default: "`.`" (for relative)
`Policy Distribution`	Specifies whether the Policy Import tool will distribute policy. If the distribution file is in policy distribution path and PolicyDistribution parameter is set to yes, the policy will be distributed. Supports `YES` or `NO` setting. Default: `YES`. `YES` = The Policy Import tool distributes policy data. `NO` = The Policy Import tool does not distribute data. It only imports it into the database. The Administration Console can then be used to distribute data.

Table 7-1 Configuration Parameters (Continued)

Parameter	Description
`requestTimeout`	Specifies the time (in milliseconds) to wait for the server to respond. Should be longer for loading large files. May set to infinite (`ASI.INFINITE`) for very large files. Default: 600000
`RunningThread`	Number of threads running concurrently to process the policy import. The value depends on the capacity of the database server. Commonly the optimal value is 2 - 4 or be larger for a high capacity database server. Default: 2.
`Username`	Specifies the username for the administrator (optional). The `username` is case sensitive. If the `username` is not specified in the configuration file and the `ConsoleDisplay` parameter is enabled, then you are prompted to enter one. Default: `system`. **Note:** This user must have the privilege to import policy.

For more information on the configuration parameters, refer to the following topics:

- "Username and Password" on page 7-6
- "Policy Import Parameters" on page 7-6

Username and Password

Including the password in the configuration file is optional and is *not* recommended because it could be viewed by others who are not authorized to import policy. The password can be encrypted and stored in the `password.xml` file. You should set the `PasswordFile` and `PasswordKeyFile` for the policy to automatically retrieve the password using the alias as the username specified in the configuration file. If you do not include these parameters and the console display is enabled (the default setting), you are prompted to enter their values when you run the Policy Import tool. If one of the two parameters is not included in the configuration file and the console display is disabled, the Policy Import tool logs an error and terminates. When entered, the password is not displayed for security reasons.

Policy Import Parameters

This section of the configuration file specifies parameters that the Policy Import tool uses to import policy data. There are three policy import parameters: `PolicyDirectoryPath`, `RunningThread` and `BulkSize`.

The `PolicyDirectoryPath` parameter specifies the directory path for the policy files. When you start the Policy Import tool, it looks in the directory pointed by `PolicyDirectoryPath` for valid files. The directory path is either a relative or full path. If the value is left empty or the value is a period (.), the current directory of the Policy Import tool is assumed. For example:

```
PolicyDirectoryPath ../examples/policy
```

The `RunningThread` parameter specifies the number of running threads and depends on the hardware configuration of the database server. The default number is 3. For most database servers, you want to use a value from 2 to 4. For a high-capacity database server, where a high CPU speed and large memory size are allocated, increase this number to improve import performance. If you set this value too high, it may hinder the performance of the Policy Import tool. If this is the case, you can observe `database busy` warning messages in the server log file.

The `BulkSize` parameter denotes the size of each bulk load data block per thread in the Policy Import tool; that is, the number of entries imported in a single load using a single connection between server and the database. Increase the parameter value to lessen the time to initiate a connection. If you enter too high a value, the import process slows, which in turn requires higher `RequestTimeout` and `ConnectionTimeout` values. The optimal value is between 50 and 300.

Sample Configuration File

Use the sample file shown in Listing 7-1 to guide you through the process of creating your configuration file. Each parameter description includes comments, indicated by the # symbol. The sample configuration file assumes that all of your policy files are located in the directory specified by *BEA_HOME*`/ales26-admin/examples/policy`.

Note: Be sure to use forward slashes (/) when specifying the policy file directory path.

The sample configuration file also assumes that no policy distribution is performed.

Listing 7-1 Sample Configuration File

```
# Required

## In addition to this file, asi.properties is read in from the ALES_HOME/config
## directory.  Any parameters set here will override values defined there.

#### policy domain name, as set in policy database during database installation
Domain asidomain

# Optional
```

```
#### A ALES administrator user id and password.
#### If either Username or password is not provided, they can be
#### entered at prompt (case sensitive).
#### They should be same as stored in database.
#Username system

#### Encrypted password file
#### To set up a password file, use asipassword utility tool
PasswordFile ../ssl/password.xml

#### Password key file
PasswordKeyFile ../ssl/password.key

#### This is the application node that holds the administration policy.
#### If commented out it assumes the dafult value of "admin".
ApplicationNode admin

#### Number of Threads Running concurrently
#### The value depends on the capacity of the database server
#### commonly the optimal value is 2 - 4, or could be larger for high capacity
#### DB server
RunningThread 2

#### If ALES Admin console server is still coming up then you need to retry
#### the BLM API Authentication. In most cases the ALES Admin console server will
#### always be running.
#### Configure the number of times retries should take place (DEFAULT 100)
BLMContextRetries 2

#### Configure the the amount of time in milli seconds to wait between context
#### retries (DEFAULT 100ms)
BLMContextInterval_ms 100

#### Size for each bulk load. I.e. number of entries loaded in a
#### single load(200 here)
BulkSize 200

#### Loading directory value for loading policy files, value is the
#### directory from which the files will be loaded.
#### Directory path may be a relative path
PolicyDirectoryPath .

#### To indicate whether to distribute policy in same operation.
#### If distribution file is in policyDistribution path and
#### PolicyDistribution parameter is not set to no the policy WILL be
#### distributed.
#### Parameter takes either yes or no (case insensitive). Default = YES
PolicyDistribution yes

#### File where all error messages are logged.
ErrorLogFile policyImporter.log
```

```
#### To indicate the Action that the Policy Import tool will perform.
#### Values are LOAD or REMOVE (case insensitive). Default = LOAD
#Action REMOVE

#### To indicate the Mode the Policy Import tool will be in
#### Values are INITIAL or RECOVER (case insensitive). Default = INITIAL
#### This parameter can also be passed in as a commandline parameter -recover or
#### -initial.
#### Values on the command line will override values specified in the
#### configuration file.
#Mode RECOVER

#### uncomment if you want to see debug information, Default = 0 (no debug)
#Debug 1

#### uncomment if you want to hide console interaction (yes/no), default = yes
#### If you want to run loader in background/in batch process, set this to no
ConsoleDisplay yes
```

Running the Policy Import Tool

After you complete the configuration file, you can run the Policy Import tool and import your policy files.

To run the Policy Import tool:

1. Prepare your policy data files.

   You can create your own policy data files as described in Chapter 4, “Advanced Topics.” or you can use files that you have exported from your policy database as described in “Exporting Policy Data” on page 7-11.

2. Create a configuration file to define your policy load.

   You can use the `../examples/policy/policy_loader_sample.conf` file as a template for your configuration file. Additionally, for a sample configuration file, see “Sample Configuration File” on page 7-7.

3. Run the Policy Import tool.

   On a Microsoft Windows platform, run

   ```
   policyloader.bat
   ```

   On a UNIX platform, run:

   ```
   policyloader.sh
   ```

4. Check for errors in log file.

Note: If an error occurs, the Policy Loader terminates; you must restart the Policy Import tool. The name of the error file is defined in the your Policy Import tool configuration file by the `ErrorLogFile` parameter. In addition, to distribute policy you need distribution privileges granted to you.

Also, because the Policy Import tool is multi-threaded and each thread writes out to the log when it is complete, you cannot guarantee the order in which each load completes.

The Policy Import tool processes policy files according to a predefined order, and if the policy file is not found, it tries to load the next policy file in the proper order. Records imported successfully are committed to the database. After the import process begins, you cannot go back within the same process and edit changes you have made. If you want to change what you have done, you have to start a new import process. After the import process is complete, you may run the removal operation to reverse the import process.

Understanding How the Policy Loader Works

When an `Object Exists Error` occurs—indicating that you created a duplicate policy entry—the import process does not stop. When the Policy Import tool encounters an error other than the `Object Exists Error`, it generates a file named `<filename>.<version>` (for example, `object.1` , `object.2`) and the error message is logged in the configured error file.

Once the policy loader has finished, you need to check to see if there are any versioned files. If there are such files, this indicates that there were errors in certain files and only the problematic lines from those files have been placed in the versioned files. You can now correct the mistakes in the versioned files and re-run the policy loader in the recover mode. You can do this in two ways. Either:

- update the mode in the configuration file to RECOVER or
- add an extra command line argument (`-recover`) when running the policy loader again.

Now the loader will only try to load the highest version files that has not already been previously loaded. If you corrected `priv.1` and there are still problems, then the loader will now generate `priv.2` with just the lines that filed. You now have to make the fix in `priv.2` and rerun the policy loader in the recover mode. You need to keep doing this until the policy loader does not generate any new version files and the error log file does not have any errors listed in it for the last run.

Policy unloading works similar to policy loading except the order in which the files are read is reversed, and the policy is removed from the database instead of being added.

Exporting Policy Data

This section provides instructions and information on how to export policy data from the policy store. It covers the following topics:

Policy Export Tool

Policy exporting allows you to output data from the policy database to text files called policy files. These policy files can be imported back to the same or another policy database using the Policy Import tool, as described in "Importing Policy Data" on page 7-1. This tool allows you to transfer your policy data easily to a production environment.

To perform policy exporting, you need access to the policy database. In general, you can access the policy database when you are the policy owner or the database administrator.

All the files that are exported by the Policy Export tool are supported by the Policy Import tool. All the files are created even though some files may not contain any records. There are two other files exported: `object_config`, and `objattr_config`, that contain the data for SSM configuration. These files also get loaded and are similar to `object` and `objattr` respectively in format. These files are split so as to differentiate policy elements from configuration elements. However, the `object_config` and `objattr_config` files can be merged into `object` and `objattr` respectively, if needed.

Before You Begin

Before you begin, perform the following tasks:

1. Locate or create a target directory in which to store the policy files.

   Ensure that the directory is not write-protected. The free space that the export requires depends on the size of your existing policy. If your export fails because of insufficient disk space, add more space before attempting the export again. In addition, ensure that the full directory path contains no white space.

2. Ensure that the database client is installed and configured, and that you have access to the database.

   Depending on the database system, you need to have the database client installed and configured to connect to the policy database. Make sure all the environment settings are correct.

   Make sure you can access the policy database. For example, for Sybase use the `isql` command or use the `sqlplus` command for Oracle. You must be the policy owner or database administrator to run the export tool. When exporting, you are asked to provide the information for policy owner, your database login id and password.

3. Ensure that you can run the tools from the `/bin` subdirectory for the product installation.

   You must run the exporting scripts in this directory because the scripts need to locate some files relative to this directory.

   On a Microsoft Windows platform, you can open a DOS command prompt window and change to this directory.

Exporting Policy Data on Windows Platforms

This procedure exports your policy from the database into formatted text files. You perform this export using the export tool included as part of the Administration Application.

To export the policy data on a Windows platform, perform the following steps:

1. Open a command window and change to the `\bin` directory in the product installation. By default, this directory location is `C:\bea\ales26-admin\bin`.

2. Ensure that the current path (.) is included your `PATH`. Also, ensure that the client environment is set up properly.

3. At the command prompt, type the following command, and then press <Enter>:

   ```
   policyexporter.bat directory
   ```

   where *`directory`* is the target directory for the exported policy files. Be sure to include the full path of the directory. This directory cannot contain white spaces.

When exporting the policy, the configuration resources are saved to the following files: `object_config` and `objattr_config`. The Policy Import tool does not import these two files by default. If you want to import the configuration resources, you need to create a directory, and copy `object_config`, `objattr_config`, and `binding` into that directory. Rename

object_config to object and objattr_config to objattr. Then you can configure the Policy Import tool to import these to file in this new directory.

Exporting Policy Data on UNIX Platforms

This procedure exports your policy from the database into formatted text files. You perform this export using the Policy Export tool included as part of the Administration Server.

Running the Policy Export tool on Sun Solaris requires the use of a shell script. If you do not normally use this shell or have difficulty running the tool, check with your UNIX system administrator to determine if it is available in your environment. For Linux, you can run this script from a Bourne shell.

To export the policy data on a UNIX platform, perform the following steps:

1. Open a command window and change to BEA_HOME/ales26-admin/bin directory.
2. From the command line, enter the following command:

   ```
   policyexporter.sh
   ```

3. When the script prompts you for the directory in which to save the policy files, type the full path directory name, and then press <Enter>.

   When the script completes, a successful message appears.

When exporting the policy, the configuration resources are saved to the following files: object_config and objattr_config. The Policy Import tool does not import these two files by default. If you want to import the configuration resources, you need to create a directory, and copy object_config, objattr_config, and binding into that directory. Rename object_config to object and objattr_config to objattr. Then you can configure the Policy Import tool to import these to file in this new directory.

What's Next

Once you have exported the policy data, you can import the exported policy into policy database using the Policy Import tool. The exported policy files are in the format required by the Policy Import tool; however, you need to configure the tool to point to the exported file directory. You also need to create a policy distribution file distribution if you want the policy to be automatically distributed after the import completes. For additional information, see "Importing Policy Data" on page 7-1.