BEA AquaLogic Enterprise Security™®

Policy Managers Guide

Version 2.1
Revised: December 29, 2005

Copyright

Restricted Rights Legend

Third-Party Software License Agreement

Sun Microsystems, Inc.'s XACML implementation v2.0

This product includes Sun Microsystems, Inc.'s XACML implementation v2.0, which is governed by the following terms:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistribution of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistribution in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Sun Microsystems, Inc. or the names of contributors maybe used to endorse or promote products derived from this software without specific prior written permission.

This software is provided "AS IS," without a warranty of any kind. ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT, ARE HEREBY EXCLUDED. SUN MICROSYSTEMS, INC. ("SUN") AND ITS LICENSORS SHALL NOT BE LIABLE FOR ANY DAMAGES SUFFERED BY LICENSEE AS A RESULT OF USING, MODIFYING OR DISTRIBUTING THIS SOFTWARE OR ITS DERIVATIVES. IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR DIRECT, INDIRECT, SPECIAL,

CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF THE USE OF OR INABILITY TO USE THIS SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

You acknowledge that this software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility.

For all third-party software license agreements, see the 3rd_party_licenses.txt file, which is placed in the \ales21-admin directory when you install the AquaLogic Enterprise Security Administration Server.

Trademarks or Service Marks

Contents

1. Policy Overview

2. Writing Policies

3. Advanced Topics

4. Importing and Exporting Policy Data

Index

About This Document

This section covers the following topics:

Audience for this Document

This document is intended for application developers who are tasked with developing security policies using the BEA AquaLogic Enterprise Security products and security administrators who are implementing the policy. The reader should also be familiar with the policy model used by BEA AquaLogic Enterprise Security and described in the *Introduction to BEA AquaLogic Enterprise Security*.

The audiences for this document include:

- Application developers—Application developers should be familiar with the business policies for their company, including resource access control policies and privileges granted or denied to users of applications and resources.

- Security administrators—Security Administrators have a general knowledge of security concepts. They can identify security events in server and audit logs. They are tasked with implementing security policy, configuring security providers, and implementing authentication, authorization, role mapping, credential mapping, auditing and failover policies.

Prerequisites for this Document

Prior to reading this document, you should read the *Introduction to BEA AquaLogic Enterprise Security*. This document provides an overview of the product and includes conceptual information.

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

How this Document is Organized

This document is organized as follows:

- Chapter 1, "Policy Overview,"describes the different types of policies, describes how to design policies and provides general information about audiences and related information.
- Chapter 2, "Writing Policies," describes how to use the Administration Console to write policies.
- Chapter 3, "Advanced Topics," describes how to write more advanced and complex policies.
- Chapter 4, "Importing and Exporting Policy Data," describes how to create policy data files for importing policy, how to import and export policy data to from and to the policy database.

Product Documentation on the dev2dev Web Site

BEA product documentation, along with other information about BEA software, is available from the BEA dev2dev web site:

`http://dev2dev.bea.com`

To view the documentation for a particular product, select that product from the Product Centers menu on the left side of the screen on the dev2dev page. Select More Product Centers. From the BEA Products list, choose AquaLogic Enterprise Security 2.1. The home page for this product is

displayed. From the Resources menu, choose Documentation 2.1. The home page for the complete documentation set for the product and release you have selected is displayed.

Related Information

The BEA corporate web site provides all documentation for BEA AquaLogic Enterprise Security. Other BEA AquaLogic Enterprise Security documents that may be of interest to the reader include:

- *Introduction to BEA AquaLogic Enterprise Security*—This document provides an overview, and conceptual and architectural information for the BEA AquaLogic Enterprise Security products.
- Administration and Deployment Guide—This document describes tasks associated with deploying and managing AquaLogic Enterprise Security.
- Integrating with the ALES Application Environment—This document describes important tasks associated with integrating AquaLogic Enterprise Security into application environments.
- *Programming Security for Java Applications*—This document describes how to implement security in Java applications. It include descriptions of the security service Application Programming Interfaces (API) and programming instructions for implementing security in Java applications.
- *Programming Security for Web Services*—This document describes how to implement security in web server applications. It include descriptions of the Web Services Application Programming Interfaces (API) and programming instructions for implementing security in web server applications.
- *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*—The Javadocs provide reference material for the Java Application Programming Interfaces (API) that are provided with and supported by this release of BEA AquaLogic Enterprise Security.
- *Wsdldocs for Web Services API*—The Wsdldocs provide reference material for the Web Services API that are provided with and supported by this release of BEA AquaLogic Enterprise Security.

- *Javadocs for the Business Logic Manager API*—The javadocs for the Business Logic Manager (BLM) provide reference material for the BLM API. The BLM can be used to define security policy which can then be deployed using the Administration Console.
- *Javadocs for Security Service Provider Interfaces*—The Javadocs provide reference material for the Security Service Provider Interfaces (SSPI) 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 document 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, 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

CHAPTER 1

Policy 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—**authorization policies, also referred to as access policies, define what users, groups, or roles have what privileges on what 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, role. You can also use delegation policies to assign roles previously granted to the one user to another user or group.

Policy Components

All policies follow a syntax that is similar to English grammar. 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, policy subject, delegator) IF
constraint;
```

A single policy can have multiple privileges, resources, and policy 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 an subject. Deny is used to deny a privilege or a role from an 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 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 **policy subject** can be users, groups, or roles. For authorization policies, policy subjects can be users, groups, or roles. For role mapping policies, policy subjects can be users or groups.
- A **delegator** is a user, or subject, whose privilege or role is being delegated, or assigned, to another policy 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, policy 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.

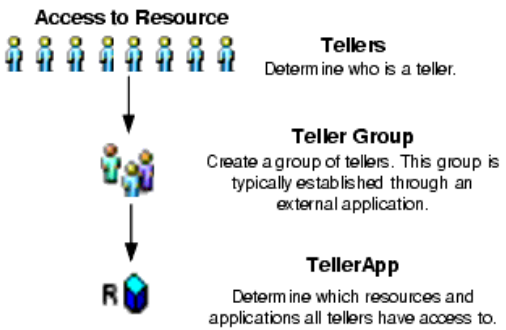
- The delegator portion of a policy is optional. It is only used for delegation policies.
- A delegator must be a user, not a group or role.
- For authorization policies, policy subjects can users, groups, and roles, however, for role mapping policies, policy subjects are limited to users and groups. Roles cannot be used as policy subjects in role mapping policies.
- You may not delegate to a role 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.

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. So defining a resource tree is a critical requirement to enable you to write policies. For example, if you define a resource named `TellerApp` (see Figure 1-1), you can then write an authorization policy that grants or denies access privileges to the `TellerApp` resource.

Figure 1-1 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 (EJB)

- 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 4-25.

The following topics describe resource components:

- "Resource Attributes" on page 1-6
- "Privilege Groups" on page 1-6
- "Privileges" on page 1-6

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

Simply stated, 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.

What actions can a user can perform on a resource?

Privileges are 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, policy 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];
```

With this policy is deployed, only tellers who are assigned the `LeadTellers` role are allowed to used the `TellerApp` to open an account 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 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 policy subject.

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 1-8

"Groups" on page 1-8

"Users" on page 1-9

"Roles" on page 1-9

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 policy 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

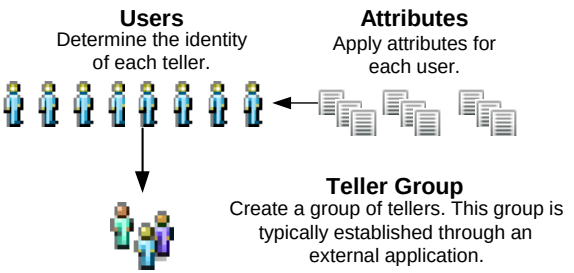
A group is typically a collection of users that have something in common, such as a department, a job function, or a job title. For example, a group named Accounting might contain users in the accounting department. It is important to realize that this does not directly reflect what access rights they 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 a 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 1-2), Tellers are assigned to the Teller Group.

Figure 1-2 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 1-10
- "Authorization Policies" on page 1-11
- "Delegation Policies" on page 1-12
- "Summary of Policy Differences" on page 1-13

Role Mapping Policies

How are user roles assigned?

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

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

Where the `grant|deny` portion is the policy effect and either allows or prohibits the role to the policy subject for the given resource, the `role` defines the role, the `resource` is the application or application component to which the role is scoped, `policy 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/);
```

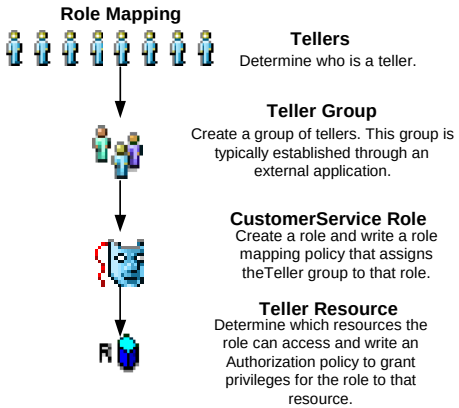
grants the `accountants` role to bill on the `payroll` resource.

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 1-3 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 1-3 Role Mapping Policy



Authorization Policies

What the user is allowed to do to this resource?

The syntax of an authorization policy is as follows:

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

Where the grant|deny portion is the policy effect and either allows or prohibits the privilege to the policy subject for the given resource, the privilege defines the privilege, resource defines the application or application component of the privilege, policy 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. And, 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 policies 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, policy 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.

Summary of Policy Differences

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

Table 1-1 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.
Policy 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.

Table 1-1 Summary of Policy Differences (Continued)

Policy Component	Authorization Policy	Role Mapping Policy
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 1-14
- "Enumerated Types" on page 1-15
- "Attributes" on page 1-15
- "Evaluation Functions" on page 1-15

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, IP address, or you can 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 of false.

AquaLogic Enterprise Security provides a number of pre-defined 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.

CHAPTER 2

Writing Policies

The following topics are covered in this section:

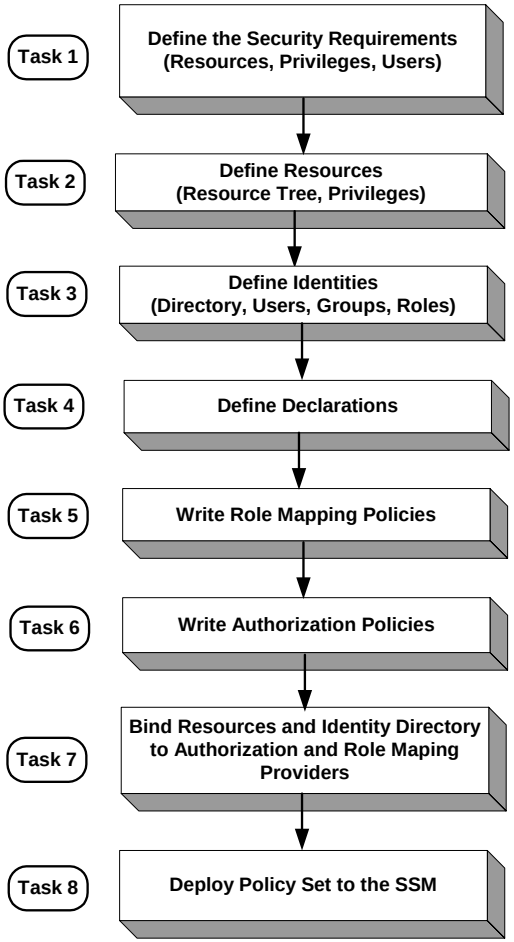
Policy Implementation Tasks

To write and deploy a set of policies, perform the following tasks (see Figure 2-1):

1. **Task 1:** Define security requirements for your business.
2. **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.
3. **Task 3:** Define an identity directory and the identity attributes, users, groups, and roles that are to make up the directory. Optionally, metadirectories can be used to import user identities from remote repositories.
4. **Task 4:** Define declarations to use with the resources and identities and as constraints in authorization policies, role mapping policies, and delegation policies.
5. **Task 5:** Write role mapping policies that control which users and groups have membership in specific roles, under what constraints, and on which resources.

6. **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.
7. **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.
8. **Task 8:** Deploy the set of policies to the SSM. The SSM starts enforcing policies only after they are deployed.

Figure 2-1 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 2-4.
- For instructions for using the BLM, see the Javadocs for the Business Logic Manager.

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.

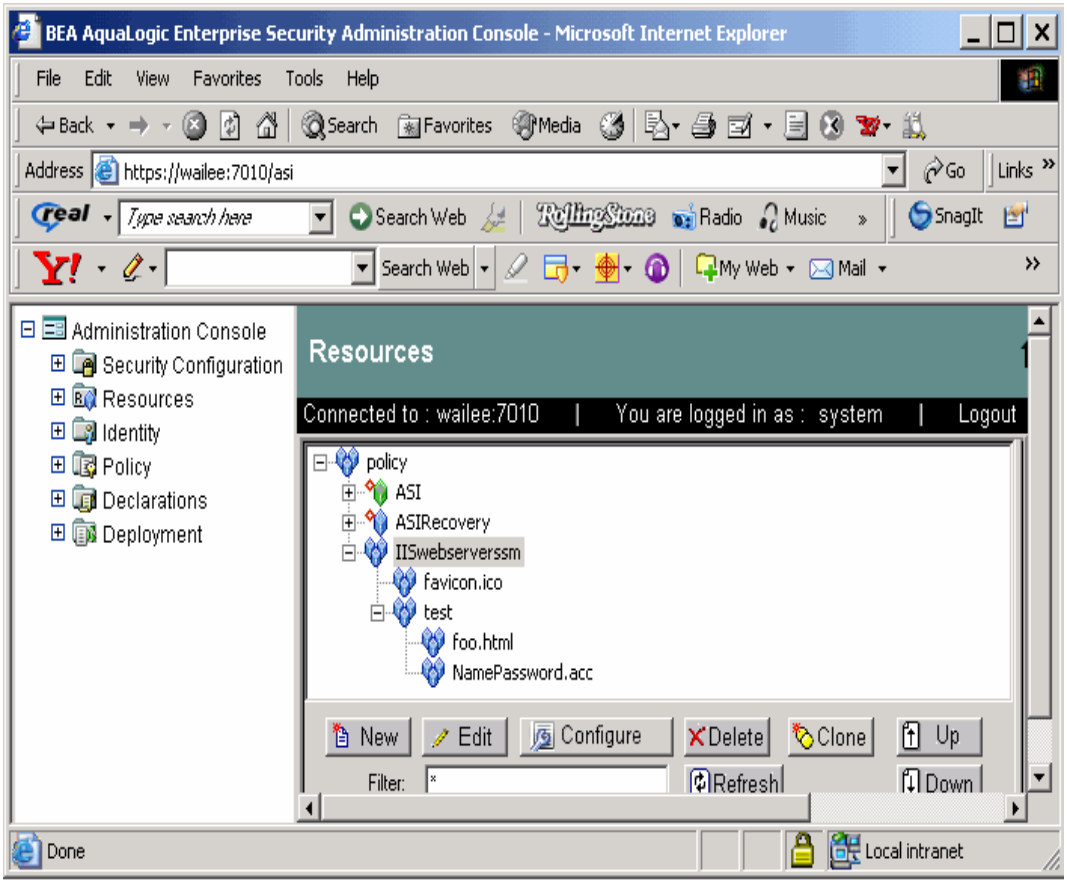
The following sections describe the components of AquaLogic Enterprise Security policies as presented in the Administration Console and how you write security policies using the Administration Console:

Administration Console Overview

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—modeled on your security requirements of your business—that taken together define who has access to which resources.

Figure 2-2 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 2-2 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 2-3 shows the expanded Resource node in the Administration Console.

Figure 2-3 Expanded Resource Node

Clicking Resources displays all of the resources that are defined in the right pane (see Figure 2-4).

Figure 2-4 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.

Individual resources in the hierarchy are also called nodes and the type of node can convey additional information about the resource. Some nodes are organizational because they represent a logical grouping of resources. For example, an organizational node called accounting may represent all resources in the accounting department. A node representing a group of resources is represented by a blue resource icon .

Another type of organizational node is an application node represented by a green resource icon . This icon depicts a collection of resources that provide a specific set of services. However, an application node can also be used to represent an application with a single component, such as a desktop spreadsheet application.

Individual resource nodes are represented by the  icon, and typically represent an individual resource that can be protected. An administrator may define as many resources and levels in the hierarchy as needed to represent data, services and system components within an application.

A special node, called a binding node, is depicted by a resource or application node with a small red diamond  . A binding node is used to associate (or bind) a subset of the resource tree to a particular SSM.

Any resource at or above a binding node can be marked as a policy distribution point. When a policy distribution 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 from the existing distribution points. Only updates that were made at or below the selected distribution points 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 help pane, and click Creating a Resource.

The following topics provide more information on configuring resources:

- "Resource Attributes" on page 2-7
- "Privileges" on page 2-8
- "Privilege Groups" on page 2-9

Resource Attributes

A resource attribute is represented by the  icon in the Administration Console (see Figure 2-3). All resources can have attributes, which store information about the resources to which they belong. Common resource attributes might be a file type, resource owner, or the

creation date. 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.

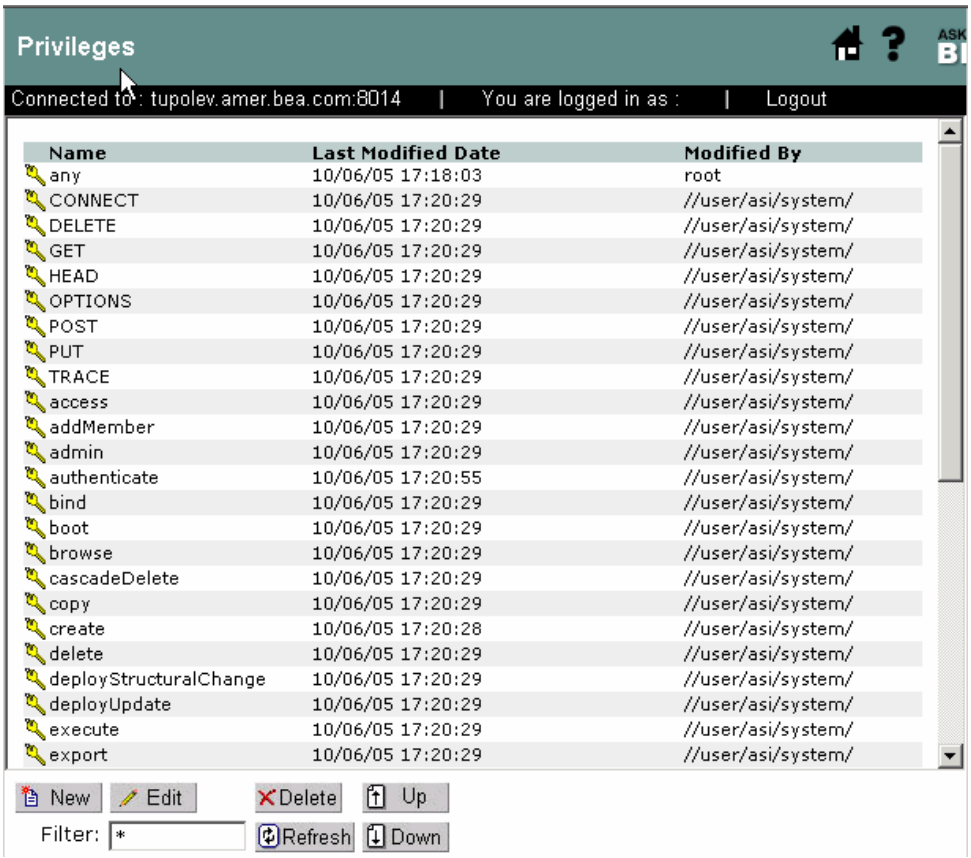
Privileges

A privilege is represented by the key  icon in the Administration Console (see Figure 2-3) 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 2-5 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 2-5 Privileges Representation in the Administration Console

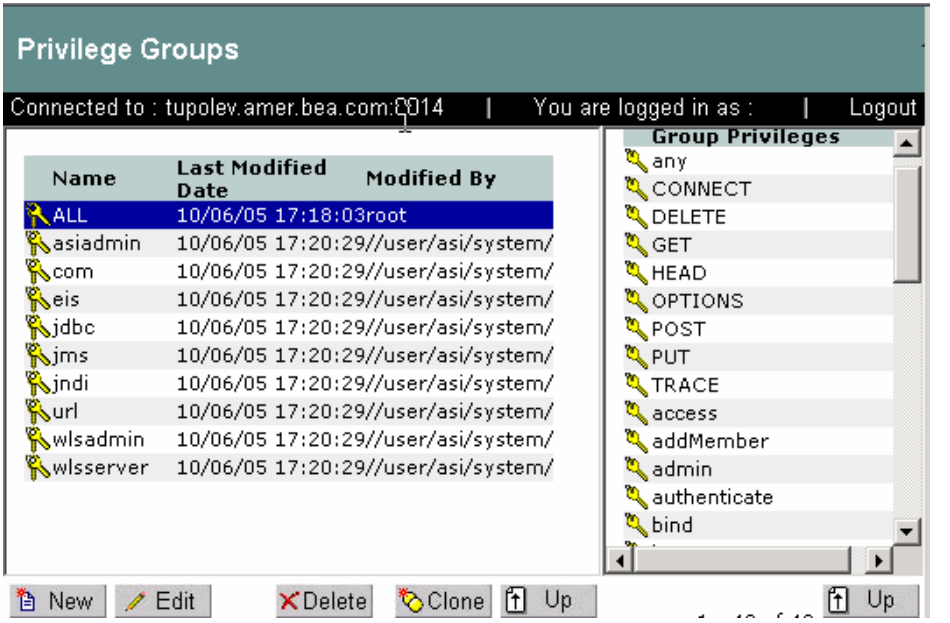


Privilege Groups

A privilege group is represented by the keys  icon in the Administration Console (see Figure 2-3) 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 2-6 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 2-6 Privilege Groups Representation in the Administration Console



Defining Identities

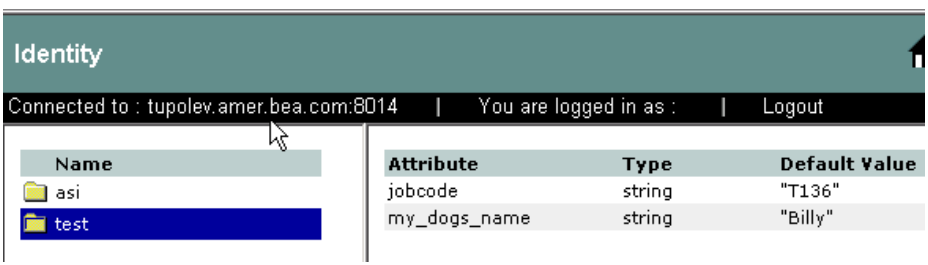
The Identity icon represents your directories and user communities in the Administration Console (see Figure 2-3). Although BEA AquaLogic Enterprise Security provides tools to manage users and groups locally, they are typically managed through an external repository, such as a Lightweight Directory Access Protocol (LDAP) directory server or a network database. A virtual view of identity data can be created through the replication and synchronization of the data using the metadirectory tools. User and group information, along with any attributes, is then stored as metadata in the policy database and is then available for viewing directly through the Administration Console.

Figure 2-7 shows the expanded Identity node in the Administration Console.

Figure 2-7 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 2-8 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 2-8 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 2-12
- “Groups” on page 2-12
- “Users” on page 2-14
- “Roles” on page 2-14
- “Metadirectory” on page 2-15

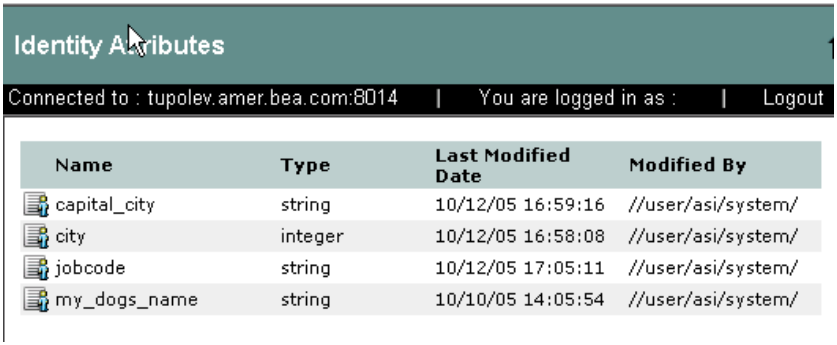
Identity Attributes

Identity attributes are represented by the  icon in the Administration Console (see Figure 2-7). 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 2-9 shows how identity attributes are represented in the Administration Console.

Figure 2-9 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 represented by the  icon in the Administration Console (see Figure 2-7) and 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

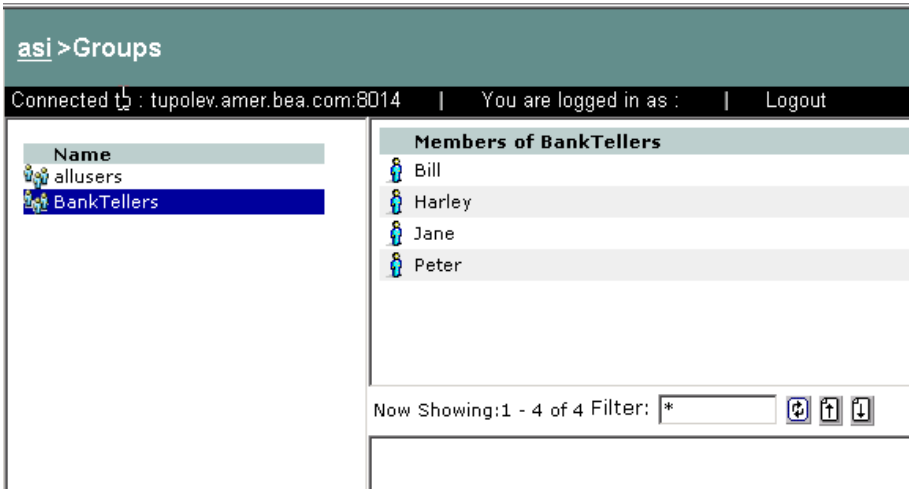
Representatives and Sales Managers, because they want their sales personnel to have different levels of access to resources depending on their job functions.

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 2-10 shows how groups are represented in the Administration Console. Notice that the `BankTellers` group contains four members.

Figure 2-10 Group Representation in the Administration Console



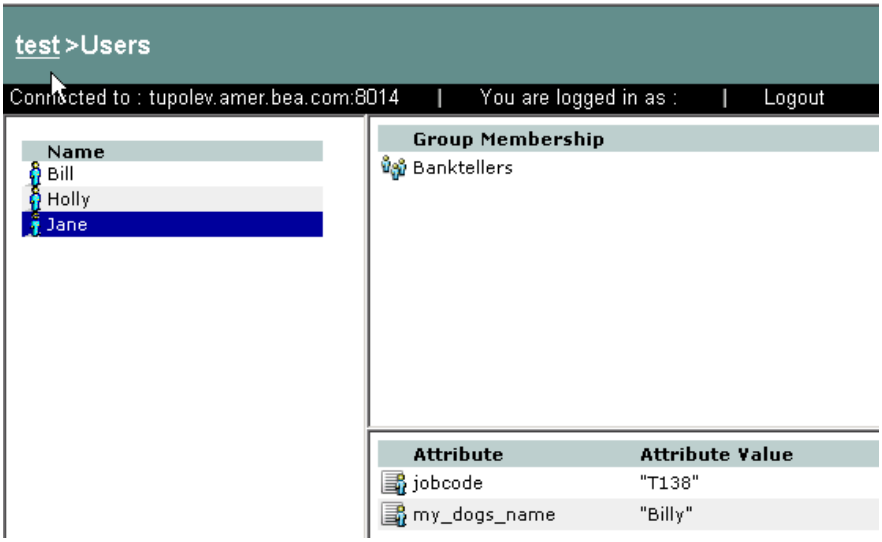
For instructions on how to create groups, log into the Administration Console, access the console Help, find Identity-->Groups in the left help pane, and click Creating a Group.

Users

A user is represented by the  icon in the Administration Console (see Figure 2-7) and 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 2-11 shows an example of a user representation with identity attributes.

Figure 2-11 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

Roles are represented by the  icon in the Administration Console (see Figure 2-7). 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 2-12 shows an example of a roles representation.

Figure 2-12 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.

Metadirectory

Metadirectories are represented by the  icon in the Administration Console (see Figure 2-7). You can use a metadirectory to extract user data from your user repository (for example, an LDAP server, an Active Directory, a database server, or NT Domain directory) and import that data into the policy database. As a result, the user, group, roles, and attributes are available and synchronized, and can be used to enforce dynamic security policies in your applications through the ASI Authorization and ASI Role Mapping services.

For instructions on how to configure metadirectories, log into the Administration Console, access the console Help, find Identity-->Metadirectories in the left help pane, and click Configuring a Metadirectory.

For more information on configuring a metadirectory, see "Configuring Metadirectories" in the *Integrating ALES with Application Environments.*

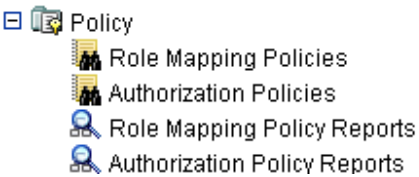
Writing Authorization and Role Mapping Policies

The policy node is represented by the  icon in the Administration Console (see Figure 2-3). 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 policy 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 policy 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 2-13 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 2-13 Expanded Policy Node



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

- “Role Mapping Policies” on page 2-16
- “Authorization Policies” on page 2-17
- “Role Mapping Policy Reports” on page 2-17
- “Authorization Policy Reports” on page 2-18

Role Mapping Policies

Role mapping policies are represented by the  icon in the Administration Console (see Figure 2-13) and displays the policies that 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, policy 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.

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 are represented by the  icon in the Administration Console (see Figure 2-13) and displays the policies that 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 represented by a blue check mark  icon in the Administration Console. An authorization policy can define privileges, resources, policy 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.

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

Role Mapping policy reports are represented by the  icon in the Administration Console (see Figure 2-13). Using this function you can 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 simply 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.

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

Authorization policy reports are represented by the  icon in the Administration Console (see Figure 2-13). Using this function you can 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 simply 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, policy subjects (users, groups, and roles) and delegators to the inquiry definition.

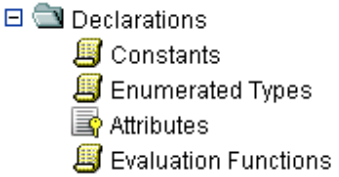
For instructions on how to create authorization policy reports, log into the Administration Console, access the console Help, find Policy-->Authorization Policy Reports in the left help pane, and click Creating an Authorization Policy Report Inquiry.

Defining Declarations

Declaration are represented by the  icon. 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). To help you design efficient policies, various built-in declarations are provided for your use.

Figure 2-14 shows the expanded Declarations node in the Administration Console.

Figure 2-14 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 3-17.
- **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 3-21.

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

To use the Administration Console to bind the Security policy 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.

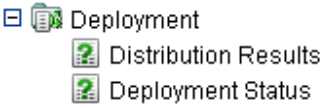
5. Click the Details tab, enter identity directory you defined when you define the user identities and the application deployment parent (`//app/resource`) you defined the top-level resource, and click Save.

Deploying Policies

Deployment is represented by the  icon in the Administration Console. Once you have designed and tested your security policy and configuration, you need to deploy so the policy and configuration take effect in your environment. When you distribute policy, you choose the distribution point for the policy before you actually distribute it. The distribution point identifies what portions of the policy updates are made active in your environment. After the distribution, you can view the results of the policy distribution. You also distribute security configuration data. After a configuration update, you must restart the SSM to make use of the new configuration.

Figure 2-15 shows the expanded Deployment node in the Administration Console.

Figure 2-15 Expanded Deployment Node



The Deployment Status page allows you to gather information regarding policy and configuration deployments. The page shows whether any components are out of synchronization with the Administration Application.

For instructions on how to distribute policy, 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

CHAPTER 3

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, policy 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:

- "Policy Inheritance" on page 3-24

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.

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 3-10).

Note: Once a grant result is determined at runtime by the Authorization and Role Mapping Engine (ARME), 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.

The following topics provide more information on constraints:

Comparison Operators

Constraints support the comparison operators listed in Table 3-1.

Table 3-1 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_resource 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 3-2 lists the three Boolean operators allowed.

Table 3-2 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, the order in which keywords are evaluated; and, associativity, 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 a constant ("Constant Declarations" on page 3-11). 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 3-11
- "Enumerated Type Declarations" on page 3-13
- "Attribute Declarations" on page 3-14
- "Evaluation Function Declarations" on page 3-21

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 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 pre-defined, 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 called `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 four 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 the 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 3-15
- "Identity Attributes" on page 3-16
- "Static Attributes" on page 3-16
- "Dynamic Attributes" on page 3-17
- "Time and Date Attributes" on page 3-17
- "Request Attributes" on page 3-19

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 an 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 3-3 lists some examples of user attributes.

Table 3-3 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 3-4 lists the built-in time and date attributes provided for you to use.

Table 3-4 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 3-5 describes these attributes and provides and example of each one.

Table 3-5 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 an ARME 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 3-5 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 3-5 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	The action referenced in the context of a role mapping request.
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 pre-defined 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 the Performance and Caching, in *Integrating ALES with Application Environments*.

Note: By default, authorization caching is turned on.

Table 3-6 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 3-6 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.
`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.

Table 3-6 Expiration Functions for Authorization Cache (Continued)

Function	Argument Type	Description
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.
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, if you had the following authorization policy:

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

With authorization caching is enabled (it is enabled by default), you could write the following authorization policy:

```
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. Not calling `valid_until_hour` expiration function results in caching the grant decision until the next policy distribution. Therefore, with authorization caching enabled, it is important to update your time dependent policies appropriately.

Closed-world Security Environment

The policy evaluation strategy imposes a closed-world security environment. This means that before you specifically create a authorization policy granting access privileges to specific resources, users, groups, and roles have no privileges. You must grant privileges with a 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 Inheritance

Inheritance reduces the number of policies the have to written to protect a set of resources. The following topics describe how inheritance works:

- "Group Inheritance" on page 3-25
- "Direct and Indirect Group Membership" on page 3-25

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.

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 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.

Writing Policy for Web Server Web Applications

This section discusses writing security policy for web applications that are protected using either of the Web Server Security Service Modules (SSMs): the IIS Web Server SSM or the Apache Web Server SSM.

In the context of a web application, a policy enforces security to protect your web application from unauthorized access. A security policy defines who can do what, where, and when.

The resource that the web application policy protects is a URL. The resource is expressed as a BEA AquaLogic Enterprise Security representation, which separates the resource and the action. The action is the method: GET, HEAD, POST, PUT.

When writing security policies for web applications hosted on web servers you must take the resource format, the action format, and the application context (information relevant to the request environment) into consideration.

For more information, see the following topics:

Resource Format

The resource format passed to the provider is a portion of the full URL. The fully qualified URL is available in the context as "`url`". The resource format is as follows:

```
/path/to/directory/page.html
```

Using this resource format allows the Web Server SSM to handle many virtual servers, each server with their own separate security policy.

Action Format

The action passed through to the provider is the method name from the HTTP protocol. GET, POST, HEAD, PUT, or some other custom action (if the web server supports it).

Application Context

The application context provides a mechanism to write authorization policy based on request attributes such as query parameters, cookies, or header information.

An application context is passed through to the Authorization and Role Mapping providers and is associated with any audit records logged. This context contains values relevant to the request environment at the time the provider processes the call. The values in the context are not prefixed with key names that segment the context into related values. If a query string name and HTTP header name collide, only one will be represented in the application context. The order in which names are added from the HTTP request is undefined.

The Web Server SSM supports the following context keys:

- HTTP Headers
- Cookies

- URL Query Strings

Note: All context keys are returned as strings, including `date`, which is normally a type.

When you write security policies for web applications that are protected by a Web Server SSM, you add the context key value pair (name/value) to the constraint. For example the policy:

```
grant ( //priv/any, //app/policy, //sgrp/allusers/ ) if accept-language like
"*us_en*";
```

where `accept-language` is the name and `"*us_en*"` is the value.

This policy grants any privilege on any resource to all users whose browser is configured to accept United States English.

The authorization policy:

```
grant ( //priv/any, //app/policy, //sgrp/allusers/ ) if session.accesscount
< 100;
```

grants any privilege on any resource to all users 100 times within the session. On the 101st evaluation of this policy within the session access will be denied.

Header Context Key (HEADERNAME)

Header context keys return values in the HTTP request header. This key is returned as a string. Any value in the header can be retrieved and so there is no hard-coded set of keys in this context. The `HEADERNAME` component of this context key is case sensitive, and specifically linked to the HTTP protocol. Examples of keys often available are: `"date"`, `"if-modified-since"`, `"referrer"`, or `"user-agent"`.

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

Query Context Key (VARNAME)

The Query context key returns values that are on the URL query-string. This key is returned as a string. The `VARNAME` portion of this key is case sensitive and refers to the query string variable encoded within the request. For example, if the URL includes a query such as `?test=endcoded%20char`, the security policy query is:

```
"if query.test= "encoded char"
```

Cookie Context Key (COOKIENAME)

The cookie context key returns values passed to the web server as cookies. This key is returned as a string. The `COOKIENAME` portion of this key is case sensitive and refers to the name of the

cookie passed to this request from the browser. The value of the cookie returned is application specific and may need further decoding. For example, if you are using the ALES cookie, an example context key is:

```
"ALESIdentityAssertion"
```

Using Named Keys in the Web Application Policy

In addition to using elements of the resource to map the resource to the web application resource hierarchy in the administration server, the named values themselves can be referenced directly in policy constraints. For example, to write a policy on `policy2.asp` so that a policy only applies to the file if there is a mode=view argument, you would write an authorization policy for the `policy2.asp` resource with a mode=view constraint as follows:

```
grant( //priv/any, //app/policy/mywebapp/policy2.asp, //sgrp/allusers/ ) if
if mode="view";
```

This policy applies the constraint to access to the `policy2.asp` file if `mode` is either an HTTP header or a query string argument.

Web Application Context Handler

The Web Server SSM implements a context handler that provides contextual information from the HTTP request to the security framework. This information includes HTTP header, query arguments, and cookies. All information is in a single namespace. Therefore, there is the possibility of name collisions between the name/value pairs.

Retrieval of Response Attributes

The Web Server SSM retrieves response attributes during the authorization process and provides them in a form that a layered web application can use. For more information on response attributes, see “Using Response Attributes” on page 3-29

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);
```

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 3-31
- “report_as() Function” on page 3-31

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.

Note: 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 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 3-1 shows an example of how to use a plug-in to implement this function.

Listing 3-1 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 is`AccessAllowed` 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 ARME 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 ARME 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.

CHAPTER 4

Importing and Exporting Policy Data

This section covers the following topics:

Introduction

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.

Creating Policy Data Files for Importing

In addition writing policy using the Administration Console or the Business Logic Manager, you may also write policy by creating and importing policy data files. To use policy data files to import policy, you must enter the policy data in a text file. When you view policy data in Administration Console, you are viewing its external representation. However, internally, policy data names are represented with certain prefixes or suffixes. You must use this internal representation when you create policy data files.

This following topics describe how to create policy data files:

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 4-3 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 4-1 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 4-5).

Table 4-1 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/`
role	`//role/roleName`
logical name	`//ln/ShortHandForResource`

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

- "Fully Qualified Names" on page 4-3
- "Policy Element Qualifiers" on page 4-4
- "Size Restriction on Policy Data" on page 4-5
- "Character Restrictions in Policy Data" on page 4-6
- "Special Names and Abbreviations" on page 4-11

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 either 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 4-2 lists the built-in qualifiers.

Table 4-2 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 4-11.

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 4-3 summarizes the limit for different policy data and different databases.

Table 4-3 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

Table 4-3 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
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
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 4-4 describes the extended character set restrictions.

- Because AquaLogic Enterprise Security does not support internationalization, the character set used in policy data is ISO 8859-1 (Latin-1), Western European eight-bit character set.
- 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 name, privilege

group name, role name, resource name, user name, subject group name, directory name, logical name, security provider name, declaration, and attribute name.

- All names, except user and subject group name, must start with an alpha character or underscore. Numeric characters are not allowed.

Table 4-4 Policy Data Character Restrictions

Policy data	Extra characters allowed
Resource name	Pound sign (#), apostrophe (`), dash (-), period (.), colon (:), at (@), tilda (~), ampersand (&).
User name or Subject group name	The name can include any other printable characters, such as space, period, and dollar sign, etc. A forward slash (/) in the name must be escaped by a backward slash (\), because a forward slash (/) is used as field separator.
String typed attribute value	All printable characters are 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 4-7
- "Directory Names" on page 4-10
- "Logical Name" on page 4-10
- "Declaration Names" on page 4-11

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 4-5 apply to the following elements: resource, attribute, privilege, role, and directory names.

Additionally, any non printable character is translated into the numeric hexadecimal equivalent; for example, the ASCII character code 1 (a smiley face) is represented as __0x1_. Table 4-5 shows the characters that are normalized and the character substitution applied at runtime. When writing policy, you must substitute these characters.

Table 4-5 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_

Table 4-5 Character Substitution (Continued)

Character	Character Substitution
#	__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_
+	__PLUS_
,	__COMMA_
/	__FSLSH_
;	__SCLN_
<	__LT_
=	__EQ_
>	__GT_
?	__QTM_
[	__OSQB_
\	__BSLSH_
]	__CSQB_
‘	__CSQUOT_
{	__OCRL_
|	__PIPE_
}	__CCRL_
&	__AMP_ Applies only to attr, priv, role, dir

Table 4-5 Character Substitution (Continued)

Character	Character Substitution
-	__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. Directory names are 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 4-6 describes the rules for using keywords.

Table 4-6 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`
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

This section provides examples of each policy file. A policy file is a text file that lists the relevant policy elements using their fully qualified names. Sample files for each policy element are provided with the product and are installed in the following directory:

```
BEA_HOME\ales21-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]” on page 4-13
- “Attribute [attr]” on page 4-14

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 4-1, the user attribute values and subject attribute value are defined as shown in Listing 4-2 and Listing 4-3.

Listing 4-1 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 4-2 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 4-3 Sample Subject Group Attribute

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

Declarations [dec]

BEA 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 4-4 shows policy inquiry examples.

Listing 4-4 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 4-5 shows policy verification examples:

Listing 4-5 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 Authorization and Role Mapping Engine (ARME) 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/myAuthorizationProvider ARME
```

```
//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 4-6.

Listing 4-6 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 4-7.

Listing 4-7 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/
```

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.discovery
mode=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 4-7 lists the files and their default locations for each type of SSM.

Table 4-7 Setting System Properties for Discovery Mode

Security Service Module Type	File Name	File Default Location
Apache	`set-env.sh` (UNIX only)	*BEA_HOME*`\ales21-ssm\apache-ssm\instance\<`*instancename*`>\bin`
IIS Web Server	`set-env.bat` (Windows only)	*BEA_HOME*`\ales21-ssm\iis-ssm\instance\<`*instancename*`>\bin`
Java	`set-env.bat` (`.sh` for UNIX)	*BEA_HOME*`\ales21-ssm\java-ssm\instance\<`*instancename*`>\bin`
Web Services	`wlesws.wrapper.conf`	*BEA_HOME*`\ales21-ssm\webservice-ssm\instance\<`*instancename*`>\config`
WebLogic Server 8.1	`set-wls-env.bat` (`.sh` for UNIX)	*BEA_HOME*`\ales21-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 4-33.

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 ARME 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 4-6 for further details.

The following sections describe how each element is represented and transformed.

Subject Transformation

A subject representation uses the standard *javax.security.auth.Subject* class that contains a set of *java.security.Principal* objects.

A subject consists of a directory name, a user name, and a set of group names. The user and groups must exist within the directory specified for the provider. The directory name is part of the provider configuration and can be modified using the 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.

Resource Transformation

A resource representation varies based on the resource type. Each WebLogic J2EE resource has a special representation based on a common resource base class. The Java API uses a standard resource representation (see Java API and Programming Security for Java Applications).

To effectively handle each of these resource representations the ASI Authorization and Role Mapping providers support a plug-in mechanism for registering code to handle the transformation to the native resource format. See Resource Converter for details on how to use the resource converter plug-in.

The ARME represents a resource as a node within a tree and the node is referenced using a path-like expression. The nodes are delimited by the forward slash (/) character and are rooted at a node named `//app/policy`. Typically, an application is not located directly below the root node, but is nested, based on an organizational structure. The application root node is referred to as the Application Deployment Parent and is a configuration property for the ASI Authorization and ASI Role Mapping providers. The plug-in defines the part of the path expression below the Application Deployment Parent.

A typical resource path may look like:

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

In this case, the Application Deployment Parent is:

```
//app/policy/MyWebServer
```

AquaLogic Enterprise Security provides resource converter plug-ins for the standard WebLogic J2EE resource types. The same transformation scheme is used for each resource and consists of the following elements:

- Name of the application with which the resource is associated
- Resources type
- Resource hierarchy that identifies the parents of the resource
- Resources name

These elements are used to define the resource path expression. The hierarchy is typically divided into a number of individual units.

Resources that have no associated application are considered to be shared and exist under a configurable shared node. Typically, this shared node is called “shared” and exists just below the Application Deployment Parent.

The following example shows a URL resource, describing its component parts. A typical URL resource looks like this:

```
http://localhost/helloworld/HelloWorld.jsp
```

It may also contain the following data elements:

- Application - `HelloWorldApp`
- Context Path Argument - /`helloworld`
- URI - `HelloWorld.jsp`

The URL Resource Converter performs the following transformation:

1. The application maps to an application
2. The resource type is: `url`.
3. The resource hierarchy is built from the context path and the URI.
4. The resource name is the last element of the URI.
5. The context path and URI are hierarchal components and are parsed based on the URL delimiter (/) and reassembled into the ARME resource.

   The resulting ARME resource is represented as:

   ```
   HelloWorldApp/url/helloworld/HelloWorld.jsp
   ```

   with an Application Deployment Parent of:

   ```
   //app/policy/MyWebServer
   ```

   a fully qualified ARME resource is thus named:

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

WebLogic Resource Transformation

The same basic pattern is used for all other WebLogic resources. Table 4-8 describes the elements of each resource class that are used to construct each resource name. An understanding of the WebLogic Server resource representation may help you have a better understanding the data included in the resource. For detailed descriptions of each resource class, see the Javadocs for the `weblogic.security.service` package in the WebLogic Server 8.1 Javadocs located at: `http://edocs.bea.com/wls/docs81/javadocs/index.html`.

Table 4-8 Standard WebLogic Server Resources

Resource Class	Application	Type	Hierarchy	Name
ApplicationResource	Application	“app”	N/A	N/A
URLResource	Application	“url”	contextPath	URI
AdminResource	shared	“adm”	Category	realm
COMResource	Absolute parent’s application name	“com”		Class name
EJBResource	Application	“ejb”	Module name	EJB name
JDBCResource	Application typically *Shared*	“jdbc”	Module name + resource type	Resource name
JNDIResource	Application typically *Shared*	“jndi”	JNDI Path	
ServerResource	Application typically *Shared*	“svr”		Server name
EISResource	Application	“eis”	Module name	EIS name
JMSResource	Application typically *Shared*	“jms”	Destination type	Resource Name
WebResource	Application name	“Web”	URI of the web resource	Resource name
WebServiceResource	application	“webservices”	Context path	Web service name
wlp	Enterprise application name	“wlp”	Web application name	Resource name

Java API Resource Transformation

The BEA AquaLogic Enterprise Security for Java 2.1 API Reference provides a more abstracted resource interface. Resources of all types are represented as a *ResourceActionBundle*. This class provides methods for extracting deep enumerations describing each resource. The conversion for the *ResourceActionBundle* resource type is simple. The first non-null, non-empty element in the enumeration is considered to be the application, and the last non-null, non-empty element is

considered to be the name. Basically the elements are joined in order using the ARME forwardslash (/) delimiter.

Action Transformation

The ASI Authorization and ASI Role Mapping providers represent an action as a simple string identifier. The Security Framework encodes the action as part of the resource object. The resource converter extracts an action similar to how it extracts the resource path. Some resources do not have the notion of an action, in which case the default action `access` is used. Table 4-9 lists the WebLogic resources and associated action transformations.

Table 4-9 WebLogic Resource Action Transformations

Resource	Attribute
URLResource	http method name
AdminResource	Admin action name
COMResource	“access”
EJBResource	EJB method name
JDBCResource	JDBC action name
JNDIResource	JNDI action name
ServerResource	Server action name
EISResource	“access”
JMSResource	JMS action name
WebServiceResource	WebService method name

With the Java API resource transformation, actions are represented as a deep enumeration. If the enumeration contains more then a single item, they are joined together using an underscore (_) delimiter. The ASI Authorization and ASI Role Mapping providers treat these as an atomic unit.

Attribute Transformations

The ASI Authorization and ASI Role Mapping providers attempt to make as much contextual information available as possible for use in making authorization decisions. This includes both data from the `ContextHandler` as well as data embedded in the resource object.

The data from the `ContextHandler` is directly available by name; that is, if the `ContextHandler` contains an element named `balance`, it can be directly referenced from the authorization policy. The data supplied in the `ContextHandler` is application specific.

The resource object used in the authorization query can contain more data than was used in constructing the ARME resource and action. This data is provided to the ARME as attributes. The specific attributes available depend on the resource type (see Table 4-10). Again, the resource converter retrieves resource specific data whenever requested.

Table 4-10 Attributes and Resources Class

Resource Class	Attributes
URLResource	application, contextpath, uri, httpmethod, transporttype, authtype, pathInfo, pathtranslated, querystring, remoteuser, requestedsessionid, requesturi, requesturl, servletpath, characterencoding, contenttype, locale, protocol, remoteaddr, remotehost, scheme, servername, serverport, issecure **Note:** Additionally, servlet parameters, attributes, headers, and cookies are all available by name. These items are specific to the request and thus cannot be enumerated here.
AdminResource	category, realm, action
COMResource	application, class
EJBResource	application, module, ejb, method, methodinterface, signature
JDBCResource	application, module, category, resource, action
JNDIResource	application, path, action
ServerResource	application, server, action
EISResource	application, module, eis
JMSResource	application, destinationtype, resource, action
WebServiceResource	application, contextpath, webservice, method, signature
ResourceActionBundle	All elements in the resource and action enumerations are available by name. These names are application specific.

What's Next?

After you create the policy data files, you can import those files into the policy store using the Policy Import tool.

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

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 "Creating Policy Data Files for Importing" on page 4-2.

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 "Sample Policy Files" on page 4-12.

Note: If you want to store users, groups, and their attributes in an external store, you can use the Metadata Directory tool, as described in "Configuring Metadirectories" in *Integrating ALES with Application Environments.*

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 4-11 lists the parameters you need to configure for the Policy Import tool.

To create the configuration file (see Listing 4-8 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 4-11. This is not the order in which they are listed in the `policy_loader_sample.conf` file.

Table 4-11 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.

Table 4-11 Configuration Parameters (Continued)

Parameter	Description
`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`.
`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`.

Table 4-11 Configuration Parameters (Continued)

Parameter	Description
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 asipasswd 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.
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 asipasswd utility. Default: ../ssl/password.key.
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.
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

Table 4-11 Configuration Parameters (Continued)

Parameter	Description
`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 4-38
- "Policy Import Parameters" on page 4-38

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 4-8 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*`/ales21-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 4-8 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 asipasswd 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 "Creating Policy Data Files for Importing" on page 4-2 or you can use files that you have exported from your policy database as described in "Exporting Policy Data" on page 4-42.

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 4-39.

3. Run the Policy Import tool.

   On a Microsoft Windows platform, run

   ```
   policyloader.bat
   ```

   On a UNIX platforms, 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. By either updating the mode in the configuration file to RECOVER or adding 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 Exporter 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 4-33. 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 either the Oracle or Sybase client installed and configured to connect to the policy database. Make sure all the environment

settings are correct as discussed in Database Setup in the *Administration Application Installation Guide*.

Make sure you can access the policy database using the **isql** (Sybase) or **sqlplus** (Oracle) command. 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.

Before you begin, make sure you have the information listed in Table 4-12:

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\ales21-admin\bin`.
2. Ensure that the current path (.) is included your `PATH`. Also, ensure that the Sybase or Oracle client environment is set up as discussed in Database Setup in the Installing the *Administration Sever.*
3. At the command prompt, do one of the following:

   For Oracle, type the following command, and then press <Enter>:

   ```
   export_policy_oracle.bat server policyowner login password directory
   ```

   For Sybase, type the following command, and then press <Enter>:

   ```
   export_policy_sybase.bat server databasename policyowner login
   password directory
   ```

   where *server*, *databasename*, *policyowner*, *login*, and *directory* are as defined in Table 4-12.

Table 4-12 Information Required to Export Policy Data

Information	Description
server	Name of your database server (Sybase) or the service name of the database server instance (Oracle).
databasename	Name of your database (Sybase only).
policyowner	Name of the owner of the policy database. This is usually the same as the username used to access your Sybase or Oracle database. Do not confused this with the database owner for Sybase.
login	Username used to access your Sybase or Oracle database.
password	Password used to access your Sybase or Oracle database.
directory	The target directory for the exported policy files. Be sure to include the full path of the directory. This directory cannot contain white space.

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 Borne shell.

Before you begin, make sure you have the information listed in Table 4-12.

To export the policy data on a UNIX platform, perform the following steps:

1. Open a command window and change to `BEA_HOME/ales21-admin/bin` directory.

2. Ensure that all the *_oracle.sh (for Oracle) or *_sybase.sh files (for Sybase) have execution permission and that the current path (.) is included in the `PATH` environment variable. Also, ensure that the Oracle or Sybase client is set up as described in Database Setup in the *Administration Application Installation Guide*.

3. From the command line, enter the following command:

   For Oracle: `export_policy_oracle.sh`<Enter>

   For Sybase: `export_policy_sybase.sh`<Enter>

4. When the script prompts you to continue, type Y, and then press <Enter>.

5. 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>.

6. When the script prompts you for your database server, type the name of your database server (Service Name in Oracle), and then press <Enter>.

7. With Sybase, the script may prompt you for your database name, type the name of your database, and then press <Enter>.

8. When the script prompts you for the policyowner, type the name of the database user who owns the policy schema, and then press <Enter>.

   The policy owner is the owner of the policy database, (for example, the database schema owner). Do not confuse the database owner (dbo) with the policy owner in Sybase.

9. When the script prompts you for your Oracle or Sybase login ID, type your database username, and then press <Enter>.

10. When the script prompts you for your login password, type your database password, 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 4-33.

Upgrading an Administration Server to AquaLogic Enterprise Security 2.1

If you are upgrading your Administration Server from WebLogic Enterprise Security 4.2 to AquaLogic Enterprise Security 2.1 and you want to save any changes you may have made to the policies and security configuration provided with the product, you must export the policies, modify the policy file as instructed in this procedure, upgrade the Administration Server to ALES 2.1, and import the policies into the newly installed server.

The following changes were made to the default policy provided with the ALES 2.1 product:

- The names of the following resources were changed:
  - `//app/policy/WLES` changed to `//app/policy/ASI`.
  - `//app/policy/WLESRecovery` changed to `//app/policy/ASIRecovery`.
- The name of the default identity directory was changed from `wles` to `asi`. This is the directory where the user system resides.

To save your changes, perform the following steps:

1. Use the Export tool to export you policy data from the Administration Server. For instructions, see "Exporting Policy Data" on page 4-42.
2. Open the exported data files and make the following changes:
   a. Change all occurrences of `//app/policy/WLES` to `//app/policy/ASI`.
   b. Change all occurrences of `//app/policy/WLESRecovery` to `//app/policy/ASIRecovery`.
   c. Change all occurrences of `//dir/wles` to `//dir/asi`.

d. Change all occurrences of `//user/wles` to `//user/asi`.

e. Change all occurrences of `//sgrp/wles` to `//sgrp/asi`.

f. Change all occurrences of `//bind/wlesadmin` to `//bind/asiadmin`.

g. Change all occurrences of `//app/config/ssm/wlesadmin` to `//app/config/ssm/asiadmin`.

h. Change `config__identity_scope S wles` to `config__identity_scope S` to `asi`.

i. Change `config__name S wlesadmin` to `config__name S asiadmin`.

3. If it is necessary to save the Policy Queries Migration file (piquery), merge any multiple items into one line. Listing 4-9 shows an example.

Listing 4-9 Example of Merging Lines in the piquery File

```
change:
O
//app/policy/MgrtOrg/MgrtBindingApp/MgrtBindingAppRes deny
//user/wles/system/
QueryAllUser
to:
O //app/policy/MgrtOrg/MgrtBindingApp/MgrtBindingAppRes deny
//user/wles/system/QueryAllUser
```

4. Install the AquaLogic Enterprise Security 2.1 Administration Server.
5. Import the policy data into the newly installed Administration Server. For instructions, see "Importing Policy Data" on page 4-33.
6. Check to see if there are any versioned files (*.1) in the policy data directory. If so, open the *.1 files and check whether any of the data is related to the asi admin configuration data and whether it is different from the current configuration defined in the latest installation.

Index

E

G

I

K

L

N

O

P

Q

R

S

U