BEA WebLogic Enterprise Security™®

Policy Managers Guide

Product Version: 4.2
Revised: September 20, 2005

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Contents

1. About this Document

Audience for this Guide 1-1
Prerequisites for this Guide 1-2
How this Document is Organized 1-2
Product Documentation on the dev2dev Web Site 1-2
Related Information 1-3
Contact Us! 1-4

2. Modeling Policies

Policy Overview 2-1
Security Configuration 2-3
Authentication Policy 2-4
    Users and Groups 2-5
    Identity Attributes 2-6
Role Mapping Policy 2-6
Authorization Policy 2-9
        Defining Resources 2-9
    Resource Attributes 2-10
    Privileges 2-10
    Conditions 2-11
    Delegation 2-12
Auditing Policy 2-12

3. Defining Rules

Overview of Securing Resources 3-1
Designing and Writing Rules 3-3
Rule Structure 3-3
Group Inheritance 3-4
Direct and Indirect Group Membership 3-5
Closed-world Assumption 3-5
Group Hierarchy Examples 3-6
Restricting Rule Inheritance 3-9
Resource Attribute Inheritance 3-10
Constraints 3-10
Declarations 3-11
Type Declarations 3-12
Constant Declarations 3-13
Evaluation Function Declarations 3-15
Attribute Declarations 3-18
Writing Policy for Web Server Web Applications 3-25
Resource Format 3-26
Action Format 3-26
Application Context 3-26
Using Named Keys in the Web Application Policy 3-27
Web Application Context Handler 3-28
Retrieval of Response Attributes 3-28
Translation of URLs to Relative Resource Format 3-28
Designing More Advanced Rules 3-29
Multiple Components 3-30
Rule Constraints 3-30

Comparison Operators 3-30
Regular Expressions 3-31
Constraint Sets 3-33
String Comparisons 3-33
Complex Rule Constraints 3-34
Boolean Operators 3-34
Associativity and Precedence 3-35
Grouping with Parentheses 3-36
Boolean Operators and Constraint Sets 3-37
Using Response Attributes 3-37
report() Function 3-38
report_as() Function 3-39
Report Function Rules Language 3-39
Using Evaluation Plug-ins to Specify Response Attributes 3-40
Using queryResources and grantedResources 3-41

4. Creating Policy Data Files

Policy Data Files 4-2
Policy Element Naming 4-3
Fully Qualified Names 4-4
Policy Element Qualifiers 4-4
Size Restriction on Policy Data 4-5
Character Restrictions in Policy Data 4-7
Data Normalization 4-8
Directory Names 4-11
Logical Name 4-11
Declaration Names 4-11
Special Names and Abbreviations 4-12

Sample Policy Files 4-13
Application Bindings 4-14
Attribute [attr] 4-14
Declarations 4-16
Directories 4-17
Directory Attribute Schemas 4-17
Mutually Exclusive Subject Groups [excl] 4-18
Resources 4-18
Resource Attributes 4-19
Policy Distribution 4-20
Policy Inquiry 4-20
Policy Verification 4-21
Privileges 4-22
Privilege Bindings 4-22
Privilege Groups 4-23
Roles 4-23
Rules 4-23
Security Providers 4-24
Subject Group Membership [member] 4-24
Subjects [subject] 4-25
Resource Discovery 4-26
ARME Request Transformation 4-28
Subject Transformation 4-28
Resource Transformation 4-29
WebLogic Resource Transformation 4-30
Java API Resource Transformation 4-31
Action Transformation 4-32
Attribute Transformations 4-32

What's Next? 4-34

5. Importing Policy Data

Policy Import Tool 5-1
Configuring the Policy Import Tool 5-2
Setting Configuration Parameters 5-3
Username and Password 5-7
Policy Import Parameters 5-7
StopOnError Parameter 5-8
Sample Configuration File 5-8
Running the Policy Importer 5-10
Errors that Halt the Importing Process 5-11

6. Exporting Policy Data

Policy Exporter Tool 6-1
Before You Begin 6-2
Exporting Policy Data on Unix Platforms 6-3
Exporting Policy Data on Windows Platforms 6-4
What's Next 6-5

CHAPTER 1

About this Document

This section covers the following topics:

- "Audience for this Guide" on page 1-1
- "Prerequisites for this Guide" on page 1-2
- "How this Document is Organized" on page 1-2
- "Product Documentation on the dev2dev Web Site" on page 1-2
- "Related Information" on page 1-3
- "Contact Us!" on page 1-4

Audience for this Guide

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

The audiences for this document include:

- Application developers—Application developers should be familiar with the business policies for their company, including access rules and privileges granted or denied to users of the 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 Guide

Prior to reading this guide, you should read the *Introduction to BEA WebLogic Enterprise Security*. This document describes how the product works and provides conceptual information that is helpful to understanding the necessary installation components.

Additionally, BEA WebLogic 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:

- This chapter provides general information about audiences and available information.
- Chapter 2, "Modeling Policies,"discusses what a policy is and describes how to design your policies.
- Chapter 3, "Defining Rules," describes how to write rules to enforce your policy.
- Chapter 4, "Creating Policy Data Files," describes how to write your policy data.
- Chapter 5, "Importing Policy Data," details how to import policy data into the database.
- Chapter 6, "Exporting Policy Data," details how to export policy data from the 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 WebLogic Enterprise Security 4.2. The home page for this product is

displayed. From the Resources menu, choose Documentation 4.2. 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 WebLogic Enterprise Security. Other BEA WebLogic Enterprise Security documents that may be of interest to the reader include:

- *Introduction to BEA WebLogic Enterprise Security*—This document provides an overview, and conceptual and architectural information for the BEA WEbLogic Enterprise Security products.
- BEA WebLogic Enterprise Security Administration Guide—This document provides a complete overview of the product and includes step-by-step instructions on how to perform various administrative tasks.
- *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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic 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 WebLogic Enterprise Security, or if you have problems installing and running BEA WebLogic 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 2

Modeling Policies

This section covers the following topics:

- "Policy Overview" on page 2-1
- "Security Configuration" on page 2-3
- "Authentication Policy" on page 2-4
- "Role Mapping Policy" on page 2-6
- "Authorization Policy" on page 2-9
- "Auditing Policy" on page 2-12

Policy Overview

Creating policies to use with BEA WebLogic Enterprise Security is a multi-step process and this section contains an overview of the general steps for managing that process. A well-written policy accurately represents the business and security rules for your enterprise, is easy to manage, and is designed for maximum efficiency. It is important to carefully design your policy with these goals in mind.

A simple administration policy is provided for you as an example. This security policy defines some default security roles and rules that protect the Administration Server and Administration Console resources. For a more detailed overview of what this policy is, see Admin Policy in the *BEA WebLogic Enterprise Security Administration Guide*.

You manage your policies based on your specific security needs and business processes. Although your security configurations, resources, users and groups, roles, and rules vary, you can use the following conceptual examples to understand the basic building blocks and how they work together. Managing policy starts with understanding how your business rules are associated with your resources, and how your users interact with those resources.

Sometimes business application users informally define the business rules. Sometimes, formalized business rules are created by a business analyst (or others) who has a more complete understanding of the requirements for an application.

Business rules may be recorded in a spreadsheet, table, or other detailed description and may appear as a series of declarative statements, for example:

Only employees in the sales team may view sales documents

or

Only employees in the sales team who are regional managers can alter discounts

When you define your business rules or evaluate your security risks, you identify resources that you need to protect, the privileges to allow, and the rules and conditions that apply for each user or group of users.

A resource is simply an object used to represent an underlying application or application component, that can be protected from unauthorized access using roles and policies. The number or resources that require protection is always increasing and forever changing. Resources are hierarchical in nature; thus, when you assign a policy to a resource, all child resources inherit that policy. Policies assigned to child resources or attributes override policies assigned to a parent resource.

Developing a policy typically begins by determining which resources you need to protect. You then create roles and rules to define which privileges apply to each resource, and under what specific conditions. Next, policy rules are created that control which users and groups belong in these roles, and under what conditions. Once these policies are deployed, the Security Service Modules apply these policies to the resources they are managing.

BEA WebLogic Enterprise Security can be used to define and implement the following types of security policies:

- Security Configuration—controls what security services are available and configured for each managed environment
- Authentication Policy—controls what authentication services and user stores are used to validate subject identities

- Role Mapping Policy—dynamically determines which users and groups belong to a role, based on certain rules and conditions
- Authorization Policy—dynamically determines resource privileges for one or more roles, (users and groups), based on certain rules and conditions
- Auditing Policy—defines the type of security-related information (authentication or authorization) to capture and store in a designated place

Security Configuration

Once you determine the environments you want to protect, you configure the security services to be used in that environment. Your configuration policy is managed through the Administration Console and each Security Services Module may have a unique or shared configuration.

The configuration policy includes setting up the desired services for each of the supported service types: authentication, authorization, auditing, role mapping, and credential mapping. Each type of service may have one or more security providers configured, depending on your specific needs. For example, you might install a Security Service Module in a web server and authenticate incoming users against a customer database–controlling which parts of the application user interface the user can access based on their role in the organization–and pass their identity to the application server behind the firewall. In this case, you might configure the following providers:

- Database Authentication provider to connect to the customer database
- ASI Authorization provider and ASI Role Mapper provider (these two providers always work together and share the same configuration)
- SAML Credential Mapping provider to map users credentials
- Audit provider to audit transactions in this Security Service Module

BEA WebLogic Enterprise Security includes five service types and each one is optional. In addition, you may have multiple authentication providers configured, for example you can authenticate users against either an LDAP or a database. For detailed information on how each service is implemented, see Security Services in the *Introduction to BEA WebLogic Enterprise Security*.

You can use the providers included with the product or you may use a custom provider developed by a third-party vendor. Custom providers are configured using their own security policy and are generally managed externally, rather than managed through the Administration Console. For detailed information on how to develop a customer provider, see *Developing Security Providers for BEA WebLogic Enterprise Security*.

Authentication Policy

How do we determine that the user is who they claim to be?

An authentication policy determines whether the subject (user or group) is known to a trusted authentication service. The service may use Active Directory, a database, an LDAP directory, or even a legacy application to access user profile information. The method of authentication you select depends on the type of user directory and authentication method desired or on the form of identity you accept from a trusted external source (see Table 2-1). For example, a web application might require a simple username and password schema, whereas a financial application might require some additional form of authentication.

Table 2-1 Methods of Authentication

Method	Description
Username and password	In username and password authentication, the user provides some form of identification and matching password that the provider (LDAP, Windows NT, or database) checks against the configured metadirectory and validates the identity of the user. If the credentials are trustworthy, the user is authenticated and an internal identity is created and signed.
Certificate	A certificate-based authentication schema is typically used when an SSL or HTTPS client requests access and presents its digital certificate to the server. The server then passes the certificate to the X.509 Identity Assertion provider that verifies the digital certificate and creates a local identity. The digital certificate is issued by a trusted certificate authority and is accepted as a form or user authentication.
Single Sign-On	This form of authentication is typical in a setting where the users are using multiple applications, each of which maintains its own security and requires user identity to control access. These may be within an organization, through an employee portal, or externally on a completely different domain. For example, if the user requires access to separate applications, one in New York with a local LDAP and the other in London that uses a proprietary user database, a SAML identity assertion credential may be passed between these environments, and then either consumed by the remote security system directly or converted to a local identity through credential mapping.

Authentication policy is defined by:

- Configuring a metadirectory to contain your subject data (optionally, containing users, groups, and attributes)
- Configuring one or more authentication providers
- If multiple authentication providers are used, defining the order in which each authentication provider executes

Users, groups and their attributes are typically stored in an external data directory that you access through a metadirectory. For additional information on configuring a metadirectory for use with BEA WebLogic Enterprise Security, see Configuring Directories, as described in *Administration Application Installation*.

Users and Groups

In BEA WebLogic Enterprise Security, the identity is called a subject and refers to the users and groups incorporated into your policies, through roles. User and group data, along with their attributes, are typically stored and managed in external directories and databases, and are gathered into the subject at the time of authentication. The metadirectory in which your users and groups are stored, is referred to simply as a directory from within the Administration Console. You can also create additional directories, users and groups through the Administration Console. A user or group is represented using the directory name, as follows:

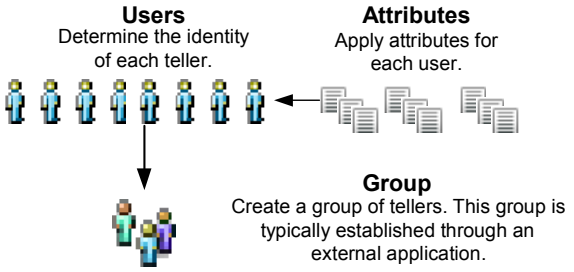
- `//user/dir/name`
- `//sgrp/dir/name`

where `dir` represents the name of the directory to which the user (`user`) or group (`sgrp`) belongs.

While user data changes less frequently, group membership changes even less frequently. Typically, for ease-of-management, you want to design a policy that applies to a large community, as establishing policies for each and every user can be very time consuming and does not scale well. Organizational structure, in the form of groups, corresponds well to typical business policies and is very useful for determining role membership. Typically, a group hierarchy is based on an organization model of the company, although this is not necessary. For example, the source of your user data might be an employee database, where a user belongs 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 (or groups of groups), whose tasks are related and for whom the policy enforcement is the same. In the following example (see Figure 2-1), Tellers are assigned to the Teller Group.

Figure 2-1 Users and Groups



Identity Attributes

A user or group can contain attributes that further describe their characteristic—who they are and what they can do; these are referred to as identity attributes. You can use these identity attributes to define dynamic conditions 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 could write a role rule that only allowed access to the application if the customer were in a specific role.

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

This role rule allows customers who belong to the `premierbanking` role, to access the resource called `bankapp`, if they have an `accountbalance` of over $100,000. The `accountbalance` is included for each customer in the `bankusers` group (`sgrp`) as an attribute.

Role Mapping Policy

How are user roles used in policy management?

User privilege is also based on the roles the user may hold at the time an access request is made. Unlike groups, which are relatively static and persist for the session, roles are highly dynamic and are assigned to users by processing role policies. Role mapping significantly reduces the number of policies required and makes features like delegation easy to manage.

Role policies are combined with authorization requests (this is done automatically for you), or are directly accessed to support uses like application or portal personalization. If you configure the role mapping service, role policies can also use the same conditions that are available for the authorization service.

The basic format of the role rule looks like this:

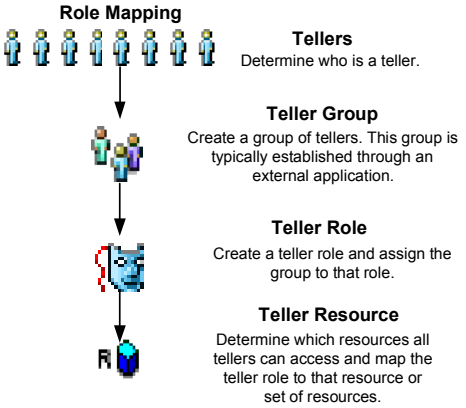
```
grant|deny(role, resource, group|user|role) IF condition
```

Where `grant` or `deny` is the privilege, `role` defines the role that applies, `resource` defines the application or application component to protect, `group` specifies what users and groups belong to the role, and `condition` applies any constraints to apply to the role rule.

After you determine who the authenticated users are, you delineate their responsibilities by determining what role they play and the groups to which they belong. A role may apply to one or more users and usually refers to some group of related tasks. For example, a group of bank tellers might have access to the same group of applications (resources) to perform specific banking tasks; thus, you might have a role called Teller Role and assign your group of tellers to that role. Figure 2-2 shows a group of tellers who belong to a teller group. The tasks they perform in your organization can then be grouped into roles.

For example, individuals or groups may be assigned the Teller Role, to which anyone who is a member can access specific resources and perform the same specific tasks. You can apply restrictions and conditions to access of the resource by defining the rules that apply to a role. The privileges or actions allowed on that resource are defined by an authorization policy. Now, anyone who does not have a Teller Role does not have access to the `TellerApp`.

Figure 2-2 Role Mapping Policy



A role mapping policy is designed by:

- Defining roles
- Configuring a role mapping provider
- Defining a set of resources
- Defining rules for users and groups.
- Assigning users and groups to roles

Authorization Policy

How do we control what the user is allowed to do?

An authorization service manages the interactions between users and resources, by defining the type of action the user may take. This action is referred to as a privilege. An authorization policy defines the tasks the requesting user can do with the requested resource, optionally based on conditions.

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 roles and policies. Resources are often hierarchical in nature; thus, when you assign a policy to a resource, all child resources inherit that policy. Policies assigned to child resources override policies assigned to the parent resource. An authorization policy is designed by:

- Configuring an authorization provider
- Defining a resource
- Defining privileges and privilege groups
- Assigning privileges to roles
- Defining rules and conditions to apply

You can assign policies to any of the defined resources, or to attributes or operations of a particular instance of a resource. Authorization policies are specific to the Authorization provider configured.

Defining Resources

A resource represents anything to which you can grant, deny or delegate roles or privileges, and any conditions that apply.

What roles can access the resource?

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

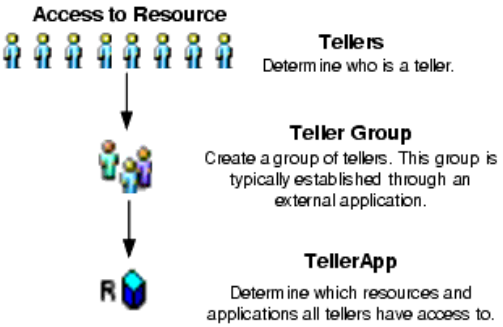
- An application, an application window, or a dialog box
- Specific business transactions, such as a money transfer or security trade
- Application controls, such as buttons and menu selections
- Database or directory server structures

- Web pages, servlets, and Enterprise Java Beans (EJB)
- Products or services in a 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 4-26.

In our example (see Figure 2-3), the teller group is granted (or denied) access to the `TellerApp` resource.

Figure 2-3 Resource Mapping



Resource Attributes

Resources can contain attributes that provide additional data about what that resource is and who may access it. 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`.

Privileges

A privilege represents an action or task in your business policy.

What actions can a user take on a resource or what privileges does a user have?

Each rule is structured using the following general syntax:

```
grant(privilege, resource, subject) IF condition1, and condition2, and
condition3;
```

For example, if you have the policy rule "*Only Managers can open an account*," "*OpenAccount*" might be one of your privileges, while Managers represents the role (comprised of users and/or groups of users). Remember, the target of the operation is the resource. Now, to add the privilege for `LeadTellers` (the role) to open an account (the privilege), the rule might look like this:

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

where the account can only be opened (`OpenAccount` is the privilege), using the TellerApp (`TellerApp` is the resource), by anyone belonging to the specified role (`LeadTellers` is the role), between the time of 9:00 AM and 5:00 PM (a time condition), on Monday through Friday (a date condition).

There are many types of conditions included with the ASI Authorization and ASI Role Mapping providers. You can use any of the built-in conditions in your rules. For detailed information on designing rules and using conditions, see "Defining Rules" on page 3-1.

Conditions

When can the user or group access the resource?

You typically want to apply conditions to define the time of day or the days of the week when a user or group can access the resource and perform a task. Conditions may contain one or more comparative statements joined using a Boolean operator (for example, `AND`, `OR`, or `NOT`). If the result of the entire expression evaluates to true, then the policy statement is true for the selected privileges, resources, and policy subjects.

Once you determine the conditions for the rule, you can define conditions, such as time, date, etc. For example, the following rule might be generated for our authorization policy:

```
GRANT(any, //app/policy/TellerApp, //sgrp/directory/tellers/)
if time24 in [900..1700] AND
if dayofweek in [Monday..Friday] ;
```

This policy rule grants access to all users belonging to the group called tellers to the resource called `TellerApp`, from 9:00 AM to 5:00 PM, Monday through Friday.

Many types of conditions may exist when a rule is applied. Often, these include conditions based on contextual data available from the application. For example, a transaction amount passed in as a parameter to an EJB method. In some cases, a rule could require dynamic data not available in the application and the provider would need to retrieve it from another source or compute it from another piece of data. This requires more advanced application development, but can be

accomplished through the use of ARME Plug-ins. For additional information, see "Provider Extensions" in the *Administration Guide*.

Delegation

Policy can also be applied by delegating a role or privilege from one user or group to another user or group. This is called delegation. For example, a manager may delegate the ability to approve expense reports to his assistant manager.

```
DELEGATE (any, //app/policy/TellerApp, //user/TellerApp/john/,
//user/TellerApp/amy/)
```

delegates privileges related to the `TellerApp` application from `amy` to `john`.

Auditing Policy

The auditing service makes a vast amount of information available from all the services, but it is the auditing provider that determines what parts of this information are captured, where the information is directed, and the format of the information. An auditing policy defines what types of transactions you record from all the security services.

The log4j Audit Channel provider features allow you to output the results to: an output stream, a Unix Syslog daemon, a Windows NT Event Logger, or any other reporting facility in a variety of formats. It also allows you to select the level of auditing required.

Note: A negative effect on performance occurs whenever auditing features are used. The more information audited, the more significant the effect on performance.

To define auditing policy, you must first determine which events you want to record. The most commonly recorded events might include each successful and unsuccessful event related to various components. For additional information on what type of information can be audited see, Audit Events in the *BEA WebLogic Enterprise Security Administration Guide*.

The Log4j auditing features support the standard Log4j features provided by the Apache Jakarta Project. For complete information on Log4j and how to configure auditing features, see http://jakarta.apache.org/log4j/docs/.

CHAPTER 3

Defining Rules

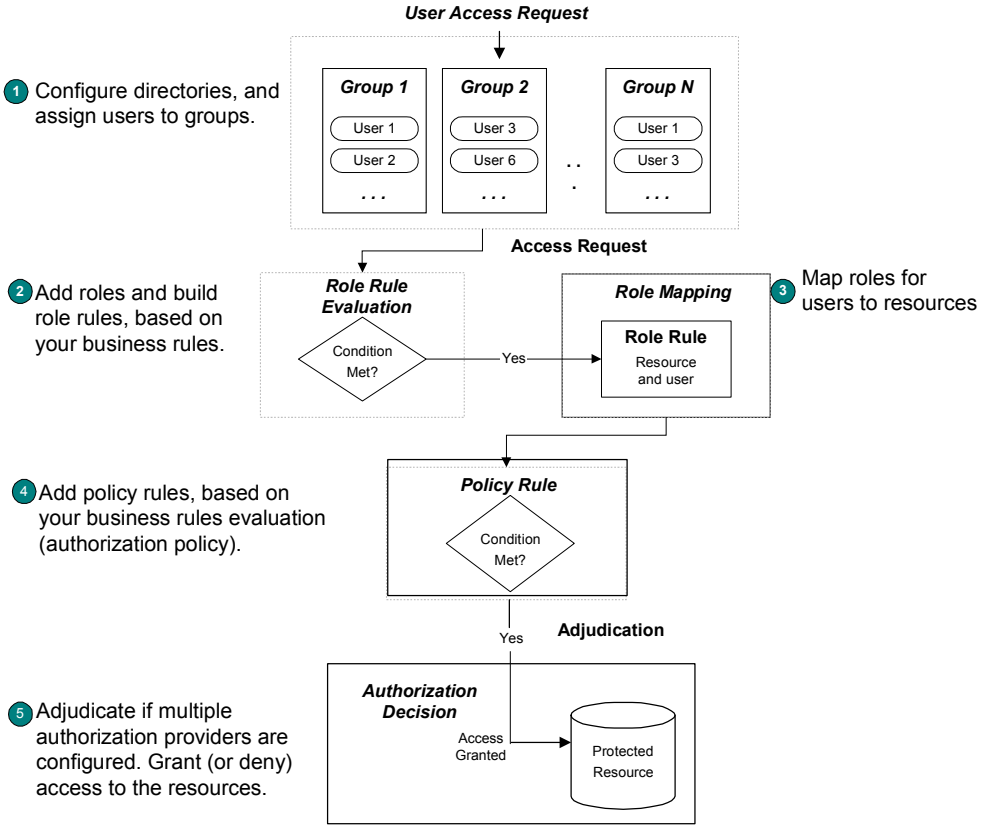
This topic describes the basic steps to defining rules. Before you begin designing rules, however, you should know which resources you want to protect with your rules. The following sections describe key concepts and tasks for securing resources, and discusses how to write rules to protect those resources:

- "Overview of Securing Resources" on page 3-1
- "Designing and Writing Rules" on page 3-3
- "Designing More Advanced Rules" on page 3-29
- "Using Response Attributes" on page 3-37
- "Using queryResources and grantedResources" on page 3-41

Overview of Securing Resources

A resource represents an underlying entity that can be protected from unauthorized access using security roles and security policies. Examples of resources include Enterprise Applications and Web Applications. Figure 3-1 illustrates the overall process for securing resources.

Figure 3-1 Securing Resources



1. Administrators typically assign users to groups based on an organizational hierarchy, however this is not necessary. You can use the Administration Application to configure an external user repository or you can add directories, users, and groups using the console. A user can be a member of multiple groups; and, a group may be a member one or more other groups. Figure 3-1 shows three groups with two users each. User 1 and User 3 are members of multiple groups. BEA recommends assigning users to groups to increase efficiency for administrators who work with many users, since groups change less frequently than users.

2. Administrators create roles based on established business procedures. The security role consists of one or more role rules, each of which may include one or more constraints. A constraint specifies the circumstances under which a particular group is granted or denied the role on the resource.
3. At runtime, the security service compares the user profile against the role rule to determine whether users in the group are dynamically granted a role. This process is referred to as role mapping. In Figure 3-1, Group 2 is the only group that is granted a security role. Individual users can also be granted a role, but this is a less typical practice. BEA recommends assigning privileges to roles (rather than users or groups) to secure resources, because doing so promotes separation of duty and increases efficiency for administrators who work with many users.
4. Administrators create a security policy based on established business procedures. The security policy consists of one or more policy rules, each of which may include one or more constraints. The policy statement specifies the circumstances under which a particular role is granted access to a user or group for a protected resource.
5. The security service uses the security policy, user profile and resource to determine whether or not to grant access to the protected resource. Only users who are granted the security role (either directly or through group membership) can access the resource. In Figure 3-1, User 3 and User 6 can access the protected resource because they are members of Group 2 and Group 2 is granted the necessary security role.

Designing and Writing Rules

A rule is a statement of what action a user or group can perform on a resource. To design and write rules, follow the instructions and guidelines presented in the following topics:

- "Rule Structure" on page 3-3
- "Writing Policy for Web Server Web Applications" on page 3-25

Rule Structure

Security policies use rules to grant or deny users permission or roles on resources. A rule is a structured statement of what action a user or group can perform on a resource. The general rule (policy or role) structure is:

```
effect(right, resource, subject[user]) IF constraint [AND/OR/NOT]
constraint;
```

Where:

`effect` is `GRANT`, `DENY`, or `DELEGATE`

`right` is a privilege or role

`resource` is the entity being protected by the rule

`subject` is a user, group, or role

`constraint` is one or more conditions (for example., `AND, OR, NOT`) under which the rule applies.

A `right`, `resource`, or `subject` can also be a list.

These rules follow a grammar that is similar to English grammar. The structure of a rule is like a sentence and the policy elements can be thought of as parts of that sentence, for example:

```
GRANT (action, resource, subject, delegator) IF constraint;
```

The `GRANT` portion of the rule represents the effect and designates permission for the user, group or role (`subject`) to access the specified `resource`. For instance, the rule:

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

grants `any` privileges related to the `acme` payroll application to the user `agarcia` in the `acme` directory.

All access to resources is considered denied until explicitly granted with a rule. Once an entitlement is granted, you must explicitly deny it with a `DENY` rule to revoke that right (or you can simply remove the `GRANT` rule). Explicit denies are very powerful and cannot be overruled. Sometimes you may want to explicitly deny an entitlement to ensure that user or group can never gain access to a specific resource. Similarly:

```
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 called `receptionist` in the `acme` directory.

A delegation rule allows you to share the role or privileges of one user with another. In addition, you might add a constraint that restricts this sharing to a certain time of day or date range, for example:

```
DELEGATE (//priv/any, //app/policy/acme, //user/acme/joe/,
//user/acme/larry/);
```

delegates `any` privileges related to the `acme` application from `larry` to `joe`.

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 rules apply to all members.

Note: BEA recommends that you define your role granting policy using groups, rather than individual users. Rules written directly on users should be used for exceptions or short-lived rules 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. An indirect member appears to either side of the parent group, rather than directly below. The collection of all groups available, either directly or through inheritance, is referred to as group closure.

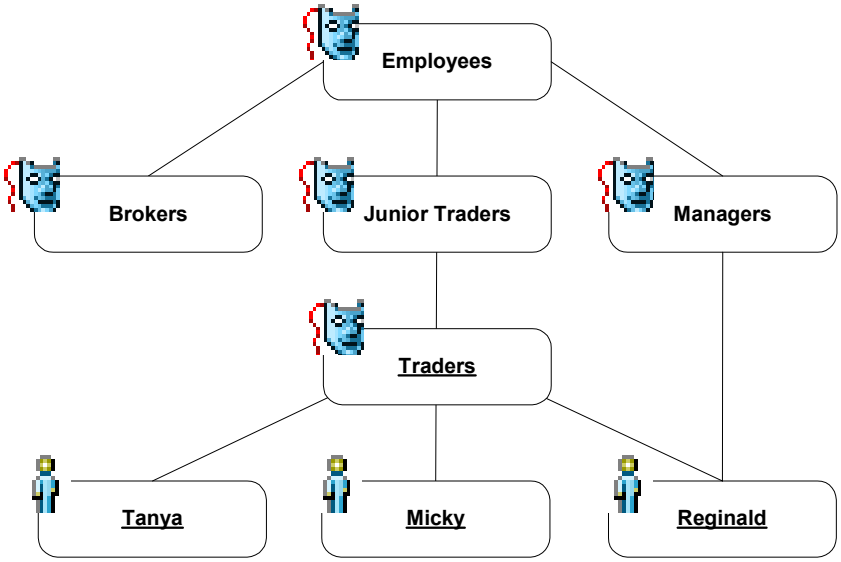
Closed-world Assumption

The policy evaluation strategy imposes a closed-world assumption. This means that before you specifically create a rule granting rights, users and groups have no rights. You must grant rights with a rule before a user can do anything. Because of this closed-world assumption, all rights to are implicitly denied until rules grant specific rights as needed.

With the closed-world assumption, a group initially has no rights granted. The closed-world assumption has the powerful advantage of having your application error on the side of caution. That is, if you forget to apply a rule, someone may be denied access rather than be granted access to something to which they should not have access. Thus, a denied user usually will let you know there is a problem (and, if they do not, that is probably okay).

Without the closed-world assumption (or if you grant all rights to every user), a user with unintended rights may never tell you they have access, which may have disastrous consequences. Once you grant a right, you must explicitly deny it to revoke that right. Explicit `DENY` rules cannot be overruled. Figure 3-2 shows an example of a typical group hierarchy.

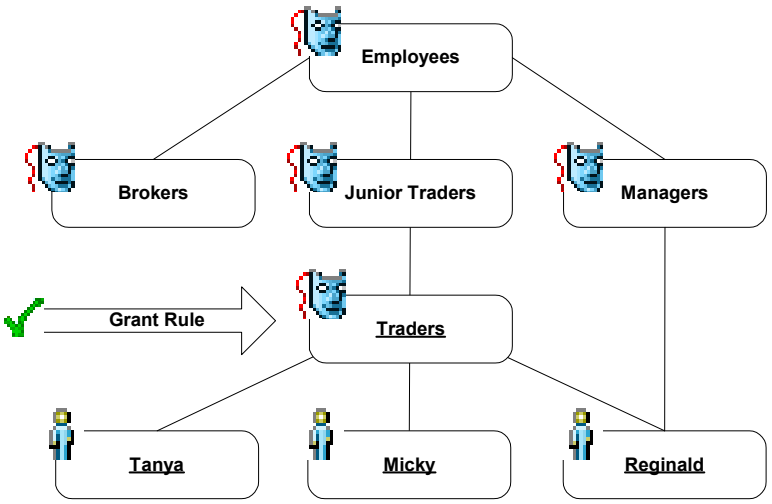
Figure 3-2 Typical Group Hierarchy



Group Hierarchy Examples

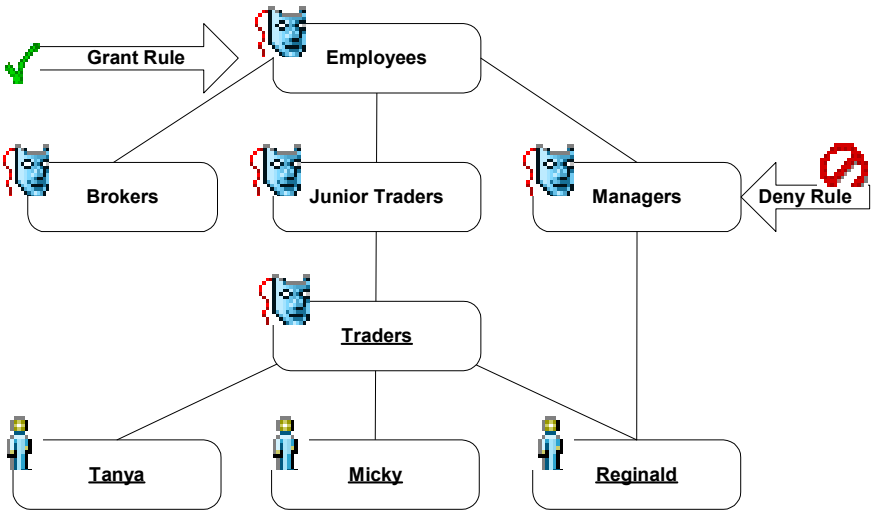
The following two examples demonstrate the application of rules to a group hierarchy. They are both based on the same group hierarchy as Figure 3-2. Explicit DENY rules always take precedence over GRANT rules. However, the implied DENY of the closed-world assumption does not. For example, in Figure 3-3, Reginald receives the grant through the Traders role even though there is an implied deny on the Managers role.

Figure 3-3 Application of a Grant Rule to a Group Structure



In Figure 3-4, all the groups are granted the right through their Employees parent and then the Managers group is specifically denied the right with a DENY rule. Notice that Reginald is denied the right despite being a descendent of the Employees group.

Figure 3-4 Application of a Deny Rule to a Group Structure



After seeing how groups work, you might wonder why you can clone a user in the Administration Console, when you can create a user with the same qualities by creating a group for both users, and then assign that group to any parent groups, rather than assigning individual users. This method does create two users with the same permissions, but there are two potential drawbacks to this approach:

- First, you may not want to clutter up your hierarchy with another group if this pairing of users is an isolated event.
- Second, the two users are locked together from then on with all the same privileges because their common parent group is inheriting the groups for both of them. That is, you cannot easily individualize the groups for each user after the initial assignment of common group memberships; however, you can by applying rules to their member users, which usually is not a good practice.

Restricting Rule Inheritance

Rules are inherited in a number of ways:

- Rules written on a resource apply to the resources descendants
- Rules written on a Group apply to all members of that Group
- Rules written on a Role apply to all members of that Role
- Rules written on the "any" privilege apply to all privileges

You can restrict rule inheritance by limiting its applicability. For instance, if you limit the applicability of the original `GRANT`, then you do not need to create a `DENY` rule. This can prevent you from getting into scenarios where `DENY` is too restrictive. However, you can use these same techniques to prevent `DENY` rules from restricting more than you would like them to. For example, this rule is applicable to any resources under the "protected" resource:

```
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. For more information on the system attributes (which begin with the prefix `sys_`), see "Request Attributes" on page 3-22. The constraint in this rule keeps it from being applicable to the descendants of the `protected` resource, thus blocking rule inheritance. In this case, you need to add this rule to grant managers the admin role to financial, assuming financial is virtual. If it is defined, you can just write the second rule:

```
GRANT(//priv/read, //app/policy/www.myserver.com/protected,
//sgrp/acme/manager/) if sys_obj_q =
www.myserver.com/protected/financial;
```

or

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

The next rule restricts the rule to apply only to users who are members of both the manager and controller groups.

```
GRANT(//role/admin, //app/policy/www.myserver.com/protected,
//sgrp/acme/manager/) if sys_obj_q //sgrp/acme/controller/ IN
sys_subjectgroups;
```

If you have lots of resources with the same leaf name, you can also write the policy like this:

```
GRANT(//role/admin, //app/policywww.myserver.com, //sgrp/acme/manager/)
if sys_obj = protected;
GRANT(//role/admin, //app/policywww.myserver.com, //sgrp/acme/manager/)
if sys_obj = financial;
```

or using pattern matching:

```
GRANT(//role/admin, //app/policywww.myserver.com, //sgrp/acme/manager/)
if sys_obj like "*/protected"
GRANT(//role/admin, //app/policywww.myserver.com, //sgrp/acme/manager/)
if sys_obj like "*/financial"
```

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.

It is recommended 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. Rules written directly for child resources should be used for exceptions or short-lived rules to handle unusual circumstances.

Constraints

Constraints limit when or under what circumstances the rule applies. All constraints start with the keyword `IF` and end with a semicolon (;). Simple constraints usually contain two values separated by an operator. The following example shows a rule with a constraint:

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

In this rule, 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 your dynamic application, data, or business environment. 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 3-11).

Note: Once a grant result is determined, the rest of the applicable GRANT rules are ignored. If your business logic requires the evaluation of multiple constraints, you must combine them into a complex constraint using an AND operator.

Declarations

Declarations allow administrators 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:

- Type – Defines the structure of the other declarations.
- Constant – States one definition for a value that is used over and over.
- Evaluation Function – Returns a true or false value from a plug-in.
- Attribute – 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.

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.

Type Declarations

A type declaration defines a class or group of values from which you can create constants and attributes. A type declaration is a template for constants and attributes. For example, an attribute of the built-in type integer 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. You could think of a type declaration as the cookie cutter and attributes as the cookies.

Several types are provided for you to use:

- `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.
- `dayofweek_type` – An enumerated type that lists the days of the week. The values are ordered from Monday to Sunday. Because `dayofweek_type` is an enumerated type, you can use it in ordered comparisons and ranges (such as Monday..Friday).
- `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.
- `month_type` – An enumerated type that lists the months in the standard western calendar. The values are ordered from January to December. Because it is an enumerated type, you can use `month_type` in ordered comparisons and ranges (such as January..March).
- `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 your own 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 rules. 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 rule.

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`, rules can then refer to the constant `Rate` rather than using its literal value, `12`. Constants are used to:

- Make rules 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 rules, 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 rule 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. Using our earlier example, without the 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 could 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.

There are a few rules for defining constant lists:

- 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 rules 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`, this test is invalid.

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

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

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 rule 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 rule. 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. The cache is not smart enough to know when authorizations 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 rules that reference dynamic data. For additional information on caching, see the Performance and Caching, in the *BEA WebLogic Enterprise Security Administration Guide*.

Note: By default, authorization caching is turned off.

Table 3-1 lists the expiration functions for the authorization cache that let you set an expiration time for the 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 3-1 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.

Table 3-1 Expiration Functions for Authorization Cache (Continued)

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

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

And if authorization caching is enabled, you could write the following rule:

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

With authorization caching, the result of this rule is cached until 11:00 AM, at which time it expires.

Not calling `valid_until_hour` results in caching the grant decision until the next policy distribution. So you can see that if you are using authorization caching it is important to update your time dependent rules appropriately.

Attribute Declarations

An attribute is a variable that you can use in your rules. 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 rules 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 rule, such as `timeofday`. Attributes contain values for your input data that your rules can manipulate. That is, they can serve as variables, for example, in our earlier example, `account_balance` could be used as an attribute.

There are four ways to use attributes:

- Identity Attribute – Provides a value defined and associated with a user or group.
- Resource Attribute – Provides a value defined and associated with a resource.
- 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.

- Configuration Attribute – Used for representing configuration data associated with security providers. These attributes are not used in rules.

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;
```

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 3-2 lists some examples of user attributes.

Table 3-2 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.

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.

Resource Attributes

Like the other 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 user or group 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.

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. Most system attributes allow you to use comparison and range operators. Table 3-3 lists the built-in time and date attributes provided for you to use.

Table 3-3 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

Table 3-3 Built-In Time and Date System Attributes (Continued)

Attribute	Value	Range or Format
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 3-4 describes these attributes and provides and example of each one.

Table 3-4 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 rule 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 rule: //user/wles/system/
sys_rule_subj	string	Unqualified subject user or group name in the currently evaluated rule: 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 rule: //app/policy/foo
sys_rule_obj	string	Unqualified resource name for the currently evaluated rule: foo
sys_rule_priv_q	string	Qualified current rule privilege: //priv/write
sys_rule_priv	string	Unqualified current rule privilege: write
sys_subjectgroups_q	list of string	List of groups to which the current user belongs: ["//sgrp/wles/admin/," "//sgrp/wles/managers/"]
sys_subjectgroups	list of strings	List of unqualified group names to which user belongs: ["admin", "managers"]

Table 3-4 Built-In Request System Attributes (Continued)

Attribute	Value	Range or Format
sys_dir_q	string	Directory of the user: //dir/wles
sys_dir	string	Directory of the user, unqualified form: wles
sys_user_q	string	Current user: //user/wles/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 rule with missing data is encountered.
sys_app_q	string	Name of the binding resource for the resource on which query is performed: //app/policy/WLES/admin
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 rule: //priv/foo
sys_priv	string	Unqualified form of the effect of the current rule: foo

Table 3-4 Built-In Request System Attributes (Continued)

Attribute	Value	Range or Format
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.

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 defines the rules that enforce 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 WebLogic 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" on page 3-26
- "Action Format" on page 3-26
- "Application Context" on page 3-26
- "Using Named Keys in the Web Application Policy" on page 3-27
- "Web Application Context Handler" on page 3-28
- "Retrieval of Response Attributes" on page 3-28

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 Security Service Module to handle many virtual servers, each server with their own separate 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 Security Service Module (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 rules 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 rule:

```
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.

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

The rule:

```
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 rule 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 rule 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 is 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

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 the following constraint:

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

This rule 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 implement 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 could use. For more information on response attributes, see "Using Response Attributes" on page 3-37

Designing More Advanced Rules

There are two ways to can create and implement rules:

- Directly edit policy files that use these conventions, and then use the Policy Import tool to load the policy file as described in "Importing Policy Data" on page 5-1.
- Create and edit your policy rules through the Administration Console, and then deploy the policy. This method is far more convenient and common and the tasks involved are described in the Administration Console Help.

All rules follow a standard structure:

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

You can extend the basic rule syntax to encompass very complex situations by grouping rules and adding constraints.

You use each rule component to specify a particular aspect of the rule. The functions of the rule components are as follows:

- `GRANT` permits the specified privilege or role. `DENY` revokes it. `DELEGATE` shares the privilege or role from one person to another.
- `privilege` is the name of the operation to `GRANT`, `DENY` or `DELEGATE`.

- `role` is the name of the role to `GRANT`, `DENY` or `DELEGATE`.
- `resource` is the fully qualified name of the application or resource on which the privilege is allowed. Resources always begin with `//app/policy/`.
- `subject` is the fully qualified name of the user, group or role to receive the entitlement.
- `delegator` is only specified for `DELEGATE` rules and identifies the user delegating the role or privilege to another user or group.
- The `IF constraint` is an optional condition placed on the privilege.

Syntax requirements for basic rules include:

- Parentheses enclose the privilege or role, resource, subject, delegator, as a group.
- Commas separate the privilege or role, resource, subject, and delegator.
- Delegator is optional and only required for delegate rules.
- You may not delegate to a role, only to a user or group.
- The keyword `IF` indicates a constraint.
- All rules end with a semicolon.

Multiple Components

You are not limited to one role, privilege, resource or subject per rule. 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/]);
```

Rule 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`. The following is an example of a rule with a constraint:

```
GRANT(any, //app/MyApp, //user/ORG/USER21)
IF UserBudget < 2000;
```

Comparison Operators

Constraints support the comparison operators listed in Table 3-5.

Table 3-5 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

This section summarizes basic rules for writing regular expressions along with some simple examples for use with constraints.

There are two 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 rule provides access (`GET`) to all JPGs in the a web application (`//app/policy/MyWebApp`).

```
GRANT(//priv/GET, //app/policy/MyWebApp, //role/webusers)
IF sys_resource LIKE ".*\\.JPG";
```

Regular Expression Syntax

The regular expression syntax follows certain rules.

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 rules 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(b|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:

```
UserQualities = "59NY20BREQ";
```

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

```
(Grant Rule) 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 Rule) 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/";
```

Complex Rule Constraints

You can use additional constraints to create more complicated conditions for your rules; however, these are not always required.

Boolean Operators

You can build complex rule 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 3-6 lists the three Boolean operators allowed.

Table 3-6 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. So, if the constraint is `true`, it becomes `false`, or vice versa. For example, if you want to make sure it is not December, you can write:

```
IF NOT ThisMonth = December
```

Your 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 rules 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 rules 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 rules:

- 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 rules. 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 credentials 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 3-13). 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.

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 rule after an IF statement used in a constraint. It is best to use them in a logical *if this rule 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 rule is evaluated, they are not really constraints of the rule. Data reported by the functions are returned only if the adjudicated authorization decision agrees with the rule. This means the attributes returned from GRANT rules are not passed to the caller unless the overall access decision is PERMIT.

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 Rules 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 rule 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 rule:

```
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 rule, 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 rule 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 *BEA WebLogic Enterprise Security Administration Guide*.

Using queryResources and grantedResources

This feature allows a caller to query the authorization system to determine access on a set of resources rather then a single resource. The ASI Authorization provider determines access to all child nodes of the node specified in the access query, and returns lists indicating which nodes are granted and which nodes are denied.

The client performs an isAccessAllowed query on the parentResource. This resource must be a binding node or a resource of a binding node.

The queryResources functionality evaluation is triggered by the presence of some qrvalue value in the com.bea.security.authorization.queryResources attribute of the ContextHandler. The access decision for the parentResource is returned, as normal. One of the return attributes for this decision is a com.bea.security.Authorization.grantedResources return attribute. One of the return attributes for this decision is a com.bea.security.Authorization.deniedResources return attribute.

For grantedResources, the value of this attribute is a list of values for the qrvalue resource attribute; or, if the qrvalue is an empty string, the value is the internal 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 rules 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

Creating Policy Data Files

Before you begin, you must understand the basic concepts of the BEA WebLogic Enterprise Security as described in the *Introduction to BEA WebLogic Enterprise Security*.

There are two ways to enter policy data into the Policy Database:

- Using the Administration Console
- Using the Policy Import tool

When viewing policy data in Administration Console, you are viewing the external representation. Internally, policy data names are represented with certain prefixes or suffixes. How these names are stored in the database is transparent to console user. However, when working with policy data files, you must use the internal representation of the policy data.

This section covers the following topics:

- "Policy Data Files" on page 4-2
- "Policy Element Naming" on page 4-3
- "Sample Policy Files" on page 4-13
- "Resource Discovery" on page 4-26
- "What's Next?" on page 4-34

Policy Data Files

The Policy Import tool reads and imports policy data that are stored as text using a proprietary format. Most policy elements are stored in a separate file, referred to as a policy file. The policy files you can import, whose names are shown in square brackets, include:

- Application Bindings [binding]
- Attribute [attr]
- Declarations [decl]
- Directories [dir]
- Directory Attribute Schemas [schema]
- Mutually Exclusive Subject Groups [excl]
- Subject Group Membership [member]
- Policy Distribution [distribution]
- Policy Inquiry [piquery]
- Policy Verification [pvquery]
- Privileges [priv]
- Privilege Groups [privgrp]
- Privilege Bindings [privbinding]
- Resources [object]
- Resource Attributes [objattr]
- Roles [role]
- Rules [rule]
- Security Providers [engine]
- Subjects [subject]

Note: If you want to store users, groups, and their attributes from an external store, you can use the Metadata Directory tool, as described in the *Administration Application Installation Guide*. You cannot import these policy elements using the Policy Import tool.

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 4-4 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/wles/system/` and `//user/wles/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 4-4
- "Policy Element Qualifiers" on page 4-4
- "Size Restriction on Policy Data" on page 4-5
- "Character Restrictions in Policy Data" on page 4-7
- "Special Names and Abbreviations" on page 4-12

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 4-12.

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
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

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
A quoted literal string in a condition 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 contrarian 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 WebLogic Enterprise Security 4.2 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 4-8
- "Directory Names" on page 4-11
- "Logical Name" on page 4-11
- "Declaration Names" on page 4-12

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_
#	__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

Table 4-5 Character Substitution (Continued)

Character	Character Substitution
%	__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
-	__DASH_ Applies only to attr, priv, role, dir
:	__CLN_ Applies only to attr, priv, role, dir

Table 4-5 Character Substitution (Continued)

Character	Character Substitution
@	__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. As previously mentioned, a policy file is a text file that lists the relevant policy elements (as listed in "Policy Data Files" on page 4-2) using their fully qualified names. Sample files for each policy element are on your installation CD and are installed in the following directory when you install the product:

```
BEA_HOME\wles42-admin\examples\policy
```

For a description of each of these files, see the following topics:

- "Application Bindings" on page 4-14
- "Attribute [attr]" on page 4-15

- "Declarations" on page 4-16
- "Directories" on page 4-17
- "Directory Attribute Schemas" on page 4-17
- "Mutually Exclusive Subject Groups [excl]" on page 4-18
- "Resources" on page 4-18
- "Resource Attributes" on page 4-19
- "Policy Distribution" on page 4-20
- "Policy Inquiry" on page 4-20
- "Policy Verification" on page 4-21
- "Privileges" on page 4-22
- "Privilege Bindings" on page 4-22
- "Privilege Groups" on page 4-23
- "Roles" on page 4-23
- "Rules" on page 4-23
- "Security Providers" on page 4-24
- "Security Providers" on page 4-24

Application Bindings

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

BEA WebLogic 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

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

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

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

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

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

WebLogic 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/wles/system/ Saved Policy Inquiry #1
O //app/policy/myApplication/myBinding  grant //user/wles/system/
      Saved Policy Inquiry #1
S //wles/wles/userid/ grant //user/wles/system/ Saved Policy Inquiry #1

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

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

# Sample query 3:

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

# Sample query 4:

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

Policy Verification

WebLogic 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/wles/system/ Policy Verification #1
LO //app/policy/myApp/firstResource //user/wles/system/ Policy Verification #1
RP //priv/create //user/wles/system/ Policy Verification #1
RO //app/policy/myApp/secondResource //user/wles/system/ Policy Verification #1

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

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

# Sample query 3:

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

# Sample query 4:

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

Privileges

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

This file contains examples of how privileges are mapped 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

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

```
//grp/
```

Examples:

```
//grp/myPrivGroup
```

```
//grp/DevelopmentGroup
```

Roles

This file defines several role names. Roles are used to construct policy rules. Each line contains a role name. Each role name is prefixed with:

```
//role/
```

Examples:

```
//role/Administrators
```

```
//role/operators
```

```
//role/managers
```

```
//role/public
```

Rules

Rules are used by ASI Authorization provider and ASI Role Mapper provider 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/wles/system/);
```

```
grant(//priv/read, //app/policy/myApplication, //sgrp/wles/allusers/);
```

```
deny([//priv/read, //priv/search_text],
//app/policy/myApplication/myBinding/confidentialDocument.one,
//role/public);

delegate(//role/Administrators, //app/policy/myApplication,
//user/wles/John Doe/, //user/wles/system/) if dayofweek in weekend;
```

Security Providers

There are two types of distribution targets in BEA WebLogic 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*`\wles42-ssm\apache-ssm\instance\<`*instancename*`>\bin`
IIS Web Server	`set-env.bat` (Windows only)	*BEA_HOME*`\wles42-ssm\iis-ssm\instance\<`*instancename*`>\bin`
Java	`set-env.bat` (`.sh` for Unix)	*BEA_HOME*`\wles42-ssm\java-ssm\instance\<`*instancename*`>\bin`
Web Services	`wlesws.wrapper.conf`	*BEA_HOME*`\wles42-ssm\webservice-ssm\instance\<`*instancename*`>\config`
WebLogic Server 8.1	`set-wls-env.bat` (`.sh` for Unix)	*BEA_HOME*`\wles42-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 5-1.

Among other things, the ASI Authorization and ASI Role Mapping (ARME) 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 4-7 for further details.

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

- "Subject Transformation" on page 4-28
- "Resource Transformation" on page 4-29
- "WebLogic Resource Transformation" on page 4-30
- "Java API Resource Transformation" on page 4-31
- "Action Transformation" on page 4-32
- "Attribute Transformations" on page 4-32

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
```

WebLogic 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 theWebLogic 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*	"srvr"		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 WebLogic Enterprise Security for Java 4.2 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

Table 4-10 Attributes and Resources Class (Continued)

Resource Class	Attributes
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 policy, the policy data files can be exported from database using the Policy Export tool (as described in "Exporting Policy Data" on page 6-1, and imported back to another database using the Policy Import tool (as described in "Importing Policy Data" on page 5-1). If you have a lot of policy data and want to use a more expedient method of importing it, the Policy Import tool is recommended.

CHAPTER 5

Importing Policy Data

This section provides instructions and information on how to load your policy data. Before you begin, you should understand the basic concepts of the BEA WebLogic Enterprise Security policy model as described in Security Administration in the *Introduction to BEA WebLogic Enterprise Security*. You should also know how to edit text files containing program parameters and environment variables for the operating system you are using.

This section covers the following topics:

- "Policy Import Tool" on page 5-1
- "Configuring the Policy Import Tool" on page 5-2
- "Running the Policy Import Tool" on page 5-10
- "Errors that Halt the Policy Importing Process" on page 5-11

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 a proprietary 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" on page 4-1.

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 "Policy Data Files" on page 4-2.

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 the *Administration Application Installation Guide*. You cannot import these policy elements using the Policy Import tool.

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" on page 5-3
- "Sample Configuration File" on page 5-8

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 5-1 lists the parameters you need to configure for the Policy Import tool.

To create the configuration file (see Listing 5-1 for a complete sample), you need a text editor such as Notepad. Create the file by entering the necessary parameters and parameter values. The following sections describe the contents of a sample configuration file, with a detailed explanation of each parameter and its default value.

Enter the following parts of the configuration file in the format described. These are only sample entries. Your entries depend on the names you create and where your files are stored. An *italics* font is used here to represent variables that you replace with your own parameter names. You do not need to list the parameters in the configuration file in this order.

There is a sample of a Policy Import configuration file named `policy_loader_sample.conf` located in the .../`examples/policy` directory. You can modify this file for your own use. BEA recommends that you use this file as a template and customize it for your particular needs.

Note: The configuration parameters are listed in alphabetical order in Table 5-1. This is not the order in which they are listed in the `policy_loader_sample.conf` file.

Table 5-1 Configuration Parameters

Parameter	Description
`Action`	Indicates the Action that the Policy Import tool will perform. Supported values are `LOAD` and `REMOVE (case insensitive).` `REMOVE` = Unloads the specific policy from the database. `ADD` = Loads the specific policy data into the database.
ApplicationNode	Specifies the application node that holds the administration policy. If this parameter is commented out, the default value of `admin` is used.

Table 5-1 Configuration Parameters (Continued)

Parameter	Description
`BLMContext Retries`	Specifies the number of times retries should take place. If the WLES Administration Console server is still starting up, then you need to retry the BLM API Authentication. In most cases the WLES Administration Console server is always running. Default: 100.
`BLMContext Interval_ms`	Specifies the amount of time (in milliseconds) to wait between context retries. DEFAULT: 100ms.
`BulkSize`	Specifies the number of records to send at one time in a thread. Default: 200. **Note:** When there are multiple threads importing policy data, each processing a number of records, the number of records processed may result in an "out-of-sync" message. However, it does not harm the data when importing the policy. The policy import tool switches to single thread when importing some policy elements, such as resources and declarations, as the later records have dependency on earlier records.
`ConsoleDisplay`	Specifies whether to hide console interaction or not (yes/no). If you want to run the policy loader in the background as a batch process, set to `no`. Default: `yes` `no =` Error messages are not displayed on the console and the user is requested to enter their `Username` and `Password` if they are missing in the configuration file. `yes` = Error messages are displayed on the console. This parameter must be enabled if you want to type in your password on the command prompt, rather then use the one specified in the `password.xml` or in the configuration file.
`Debug`	Specifies whether you want to log debug information. Default: 0 `0` = Does not log debug information. `1` = Sends debug information to the file defined by: `ErrorLogFile.`
`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 5-1 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.
`Password`	Specifies the `password` for the administrator (optional). The `password` is case sensitive. If the `password` is not specified in the configuration file and the `ConsoleDisplay` parameter is enabled, you are prompted to enter one. Default: `weblogic`. For automation or starting the policy loader without user intervention, uncomment this parameter.
`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`. For information on how to use the `asipasswd` utility, "Using the asipasswd Utility to Configure the Metadirectory Password."
`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`. For more information about the `asipasswd` utility, see "Using the asipasswd Utility to Configure the Metadirectory Password."
`Policy DirectoryPath`	Specifies the directory path from which to import policy files. For example: `../examples/policy`. The path may be relative. Default: "`.`" (for relative)

Table 5-1 Configuration Parameters (Continued)

Parameter	Description
`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
`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.
`StopOnError`	Specifies the action to take after soft exceptions. Takes a value of 0 (Continue) or 1 (Terminate). Default: 0. **Note:** The Import Policy tool terminates on hard exceptions regardless of the `StopOnError` setting.
`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 5-7
- "Policy Import Parameters" on page 5-7
- "StopOnError Parameter" on page 5-8

Username and Password

Including the username and 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 username and 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.

StopOnError Parameter

In the event of an error, the `StopOnError` parameter determines whether the Policy Import tool continues importing other policy files or terminates. This parameter takes two values: 0, which means continue and only stop on hard exceptions; or 1, which means stop on any error. The Policy Import tool reports an error if a policy file does not exist or if it cannot be opened for reading. At times, the Policy Import tool also logs an `ObjectExistsException` when you are trying to import a policy element that already exists. The exceptions are just messages to inform the user that the Policy Import tool tried to load a policy element, but it was already there. These importing exceptions do not cause the Policy Import tool to terminate.

Sample Configuration File

Use the sample file shown in Listing 5-1 to guide you through the process of creating your configuration file. Each parameter description includes comments, indicated by the # symbol. The sample configuration file assumes that all of your policy files are located in the directory specified by *`BEA_HOME`*`/wles42-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 5-1 Sample Configuration File

```
# Required

## In addition to this file, asi.properties is read in from the WLES_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 WLES 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
```

```
#### By default password need to be entered at prompt.
#### Since it will stored in clear text.
#### For automation or starting policy loader without user intervention
#### uncomment.
#PassWord weblogic

#### 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

#### Time in milliseconds to wait for server to respond (10 mins here)
#### Should be longer for loading large files,
#### possibily infinite (ASI.INFINITE) if very large files
RequestTimeout 600000

#### 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 WLES Admin console server is still coming up then you need to retry
#### the BLM API Authentication. In most cases the WLES 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

#### To specify the the course of action after soft exceptions.
#### It takes in value of 0 (Continue) and 1 (Terminate).
#### Loader terminates on hard exceptions irrespective of StopOnError value.
#### (Terminate here)
StopOnError 0

#### 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. Construct your policy files.

   You can create your own policy files as described in "Sample Policy Files" on page 4-13 or you can use files that you have exported from your policy database as described in "Exporting Policy Data" on page 6-1.

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 5-8.

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 distribute 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.

Note: Because the policy object and configuration object are exported into separate files (`object` vs. `object_config`, and `objattr` vs. `objattr_config`), an application binding is still in one binding file. When loading the exported files, the exported configuration is not loaded by default; therefore, when loading the binding for a configuration, the loader may fail, because is unable to locate the configuration objects. According to the separation above, file "binding" must have the same separation also, `binding` and `binding_config`.

Errors that Halt the Policy Importing Process

When an `Object Exists Error` occurs, meaning 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 stops the import process upon reaching the end of the files associated with the current parameter.

Similarly, when you use the `Action REMOVE` command to unload a policy, the removal process does not stop if it cannot locate the target policy entry. The Policy Import tool only stops when it encounters other kinds of errors, unless you set the configuration parameter `StopOnError` to zero. Therefore, if you want to continue the import process after encountering a minor error, set the `StopOnError` parameter to 0. When the import stops, you may correct the errors and run the Policy Import tool again.

CHAPTER 6

Exporting Policy Data

This section provides instructions and information on how to export policy data from the database. Before you begin, you should understand the basic concepts of the BEA WebLogic Enterprise Security policy model as described in the *Introduction to BEA WebLogic Enterprise Security*. You should also know how to edit text files containing program parameters and environment variables for the operating system you are using.

This section covers the following topics:

- "Policy Exporter Tool" on page 6-1
- "Before You Begin" on page 6-2
- "Exporting Policy Data on Unix Platforms" on page 6-3
- "Exporting Policy Data on Windows Platforms" on page 6-4
- "What's Next" on page 6-5

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 5-1. This tool allows you to transfer your policy data easily to a production environment.

To perform policy exporting, you need access to the policy database. In general, you can access the policy database when you are the policy owner or the database administrator.

The policy exporter does not export user-related and group-related data, such as user lists, group lists, user to group membership, and user and group attribute values (for example, data stored in an external repository). However, the policy exporter can output any rules written on users and groups. Therefore, when you are importing the exported policy into another instance of policy database using the Policy Import tool, be sure that the users and groups directly referenced in that policy exist in that policy database before the import takes place. An import rule checks for the existence of users and groups.

The policy files exported that you can import by using the Policy Import tool include: `dir`, `decl`, `schema`, `object`, `engine`, `binding`, `role`, `priv`, `privgrp`, `privbinding`, `rule`, `objattr`, `piquery`, `pvquery`. 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 in general are not imported using Policy Import tool. They contain the data for resource configuration.

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 run the tools from bin subdirectory for the product installation.

   You need to run the exporting scripts in this directory because the scripts need to locate the 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 Unix 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.

Running the 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 6-1.

Table 6-1 Information Require to Export Policy Data

Information	Description
server	Name of your database server (Sybase) or the service name of the database server instance (Oracle).
database	Name of your database (Sybase only).
policyowner	Name of the owner of the policy database. Do not confused this with the database owner for Sybase.
login	The Sybase or Oracle ID.
password	Password used to access your Sybase or Oracle database.
directory	The target directory for the exported policy files, including the full path. This directory cannot contain white space.

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

1. Open a command window and change to `BEA_HOME/wles42-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.

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 6-1:

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

1. Open a command window and change to `WLES_HOME/bin` directory.

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 *Administration Application Installation Guide*.

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 database policyowner login password
   directory
   ```

   where *server*, *database*, *policyowner*, *login*, and *directory* are as defined in Table 6-1.

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

Now, 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 5-1.

Index

A

Action
    configuration parameter 5-3
application binding 4-24
    sample file 4-2
attribute 4-16
    configuration 3-19
    directory 4-2, 4-17
    dynamic 3-18, 3-21
    identity 3-18
    resource 2-10, 3-18
        sample file 4-2
    system 3-18
attributes
    time and date 3-18
authentication 2-4
    methods 2-4
    policy 2-4

B

binding node 4-18
BLMContextInterval_ms
    configuration parameter 5-4
BLMContextRetries
    configuration parameter 5-3
BulkSize
    configuration parameter 5-7

C

caching
    authorization 3-16
certificate authentication 2-4
character set 4-7
character substitution 4-9
condition 2-11
condtion 2-11
configuration
    file, sample 5-8
    parameters 5-3
    policy 2-2, 2-3
    policy import 5-3
configuration file 5-8
ConsoleDisplay
    configuration parameter 5-4
constant 4-16
ContextHandler
    request 3-42
customer support
    contact information 1-4

D

data
    dynamic 3-16
    exporting policy 6-3, 6-4
    importing
        policy 5-1
data transformation 4-8
Debug
    configuration parameter 5-4
declaration
    sample file 4-2, 4-16
declaration name 4-11
deniedResources 3-42
directory
    name 4-11
    sample file 4-2, 4-17
directory attribute
    sample file 4-2
distribution target 4-24
Domain
    configuration parameter 5-4
dynamic
    data 3-16

E

engine 4-24
enumerated type 4-16
ErrorLogFile
    configuration parameter 5-4
errors, checking for 5-11
evaluation function 4-16
events
    recorded 2-12
expiration functions
    for caching 3-16
export
    on Unix 6-3
exporting
    policy data 6-1
exporting policy 6-1, 6-3, 6-4

G

grantedResources 3-41, 3-42

I

identity
    user or group 2-5
importing
    policy data 5-1

K

keyword 4-12

L

Log4j Audit Channel Provider 2-12
Log4j auditing
    features 2-12
logical name 4-11, 4-18

M

metadata directory
    tool 4-2

N

name
    declaration 4-11
    logical 4-11
    qualified 4-4
    special 4-12
node
    binding 4-18
normalization 4-8

O

objattr_config 6-4, 6-5
Object Exists Error 5-11
object_config 6-4, 6-5
ObjectExistsException 5-8

P

Password
    configuration parameter 5-5
PasswordFile 5-7
    configuration parameter 5-5
PasswordKeyFile
    configuration parameter 5-5
policy
    auditing 2-12
    authorization 2-9
    role mapping 2-6
policy data
    exporting 6-1
    importing 5-1
    removing 5-2
    sample files 4-2
policy database
    limits 4-8
policy distribution
    sample file 4-2
policy distributor 4-24

policy file
import 6-1
samples 4-13
policy files 4-2
Policy Import tool 4-2, 5-1
Policy Import tool, ordered policy files 5-11
Policy Importer, running 5-10
policy inquiry
sample file 4-2
policy verification
sample file 4-2
PolicyDirectoryPath
configuration parameter 5-5
PolicyDistribution
configuration parameter 5-6
privilege
defininng 2-10
sample file 4-2
privilege bindings
sample file 4-2
privilege group
sample file 4-2

Q

qualifier 4-7
queryResources 3-41, 3-42

R

recorded events 2-12
requestTimeout
configuration parameter 5-6
resource
attribute 2-10
sample file 4-2
types 2-9
resource attribute 4-19
sample file 4-2
resource name 4-18
resources
retriving a list of 3-42
ResponseContextCollector 3-42
restrictions
on input data 4-8
roles
sample file 4-2
rule
sample file 4-2
RunningThread
configuration parameter 5-6, 5-7

S

sample 5-8
sample configuration file 5-8
security providers 4-24
sample file 4-2
Service Control Manager 4-24
single sign-on 2-4
StopOnError
configuration parameter 5-6, 5-8
support
technical 1-4

U

user
privilege 2-6
Username
configuration parameter 5-6
username 5-7