Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95054
U.S.A.

Part No: 820–4803–12
December 11, 2009

091212@23031

Contents

Preface

The Sun OpenSSO Enterprise Policy Agent software consists of J2EE (Java 2 Platform Enterprise Edition) agents and web agents. This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents* provides an overview of how J2EE agents work in the Sun OpenSSO Enterprise Policy Agent 3.0 release. This guide focuses on the features and tasks that apply to all J2EE agents.

Note – This guide also provides an appendix that compares web agents and J2EE agents. See Appendix A, "Comparing Web Agents and J2EE Agents in Policy Agent 3.0."

However, for information for specific J2EE agents, such as installation information and agent-specific configuration, see the individual J2EE agent guide for that agent.

Within the Policy Agent documentation set, each agent has its own guide. Therefore, each book specific to a J2EE agent covers aspects that are unique to that particular J2EE agent.

Contents of this Chapter

Who Should Use This Book

This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents* is intended for use by IT professionals who manage access to their network. Administrators should understand the following technologies:

- JavaServer Pages™ (JSP) technology
- HyperText Transfer Protocol (HTTP)
- HyperText Markup Language (HTML)
- eXtensible Markup Language (XML)
- J2EE technologies

Before You Read This Book

You should be familiar with a variety of components and concepts related to OpenSSO Enterprise server and Policy Agent software. For example, you should be familiar with the following components and concepts:

- OpenSSO Enterprise technical concepts, as described in the *OpenSSO Enterprise 8.0 Technical Overview*
- The deployment platform, such as the Solaris™, Linux, or Windows operating system
- The web containers that will run OpenSSO Enterprise server and Policy Agent, such as Sun Java System Application Server, Sun Java System Web Server, BEA WebLogic, or IBM WebSphere Application Server

You should be familiar with the documentation related to OpenSSO Enterprise and Policy Agent 3.0. Sun Microsystems server documentation sets, some of which are mentioned in this preface, are available at `http://docs.sun.com`:

OpenSSO Enterprise Documentation Set

Policy Agent 3.0 is being introduced with OpenSSO Enterprise 8.0. The table that follows describes documents in the OpenSSO Enterprise 8.0 documentation set. Access the OpenSSO Enterprise documentation collection at the following location: `http://docs.sun.com/coll/1767.1`.

TABLE P–1 OpenSSO Enterprise Documentation Set

Title	Description
Sun OpenSSO Enterprise 8.0 Release Notes	Describes new features, installation notes, and known issues and limitations. The Release Notes are updated periodically after the initial release to describe any new features, patches, or problems.
Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide	Provides information about installing and configuring OpenSSO Enterprise, including OpenSSO Enterprise server, Administration Console only, client SDK, scripts and utilities, Distributed Authentication UI server, and session failover.
Sun OpenSSO Enterprise 8.0 Technical Overview	Provides an overview of how components work together to consolidate access control functions and to protect enterprise assets and web-based applications. It also explains basic concepts and terminology.
Sun OpenSSO Enterprise 8.0 Deployment Planning Guide	Provides planning and deployment solutions for OpenSSO Enterprise.
Sun OpenSSO Enterprise 8.0 Administration Guide	Describes how to use OpenSSO Enterprise Administration Console as well as how to manage user and service data using the command-line interface (CLI).
Sun OpenSSO Enterprise 8.0 Administration Reference	Provides reference information for the OpenSSO Enterprise command-line interface (CLI), configuration attributes, log files, and error codes.
Sun OpenSSO Enterprise 8.0 Developer's Guide	Provides information about customizing OpenSSO Enterprise and integrating its functionality into an organization's current technical infrastructure. It also provides details about the programmatic aspects of the product and its API.
Sun OpenSSO Enterprise 8.0 C API Reference for Application and Web Policy Agent Developers	Provides summaries of data types, structures, and functions that make up the public OpenSSO Enterprise C APIs.
Sun OpenSSO Enterprise 8.0 Java API Reference	Provides information about the implementation of Java packages in OpenSSO Enterprise.
Sun OpenSSO Enterprise 8.0 Performance Tuning Guide	Provides information about how to tune OpenSSO Enterprise and its related components for optimal performance.

Policy Agent 3.0 Documentation Set

Two user guides exist in the Policy Agent 3.0 documentation set:

- The guide you are reading now: *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents*
- *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web Agents*

The preceding documents are available in two documentation sets: the OpenSSO Enterprise documentation set and the Policy Agent 3.0 documentation set. The individual guides in the Policy Agent 3.0 documentation set are described in the following sections:

Individual Agent Guides

The individual agents in the Policy Agent 3.0 software set are available on a different schedule than OpenSSO Enterprise itself. Therefore, documentation for OpenSSO Enterprise and Policy Agent are available in separate sets, except for the two user's guides, which are available in both documentation sets.

The documentation for the individual agents is divided into two subsets: a web agent subset and a J2EE agent subset.

Each individual web agent guide provides agent-specific information about a particular web agent, such as installation and configuration information.

Each individual J2EE agent guide provides agent-specific information about a particular J2EE agent, such as installation and configuration information.

Related Sun Microsystems Product Documentation

The following table provides links to documentation collections for related products.

TABLE P–2 Related Product Documentation

Product	Link
Sun Java System Directory Server 6.3	`http://docs.sun.com/coll/1224.4`
Sun Java System Web Server 7.0 Update 3	`http://docs.sun.com/coll/1653.3`
Sun Java System Application Server 9.1	`http://docs.sun.com/coll/1343.4`
Sun Java System Message Queue 4.1	`http://docs.sun.com/coll/1307.3`
Sun Java System Web Proxy Server 4.0.6	`http://docs.sun.com/coll/1311.6`
Sun Java System Identity Manager 7.1	`http://docs.sun.com/coll/1514.3`

Searching Sun Product Documentation

Besides searching Sun product documentation from the docs.sun.com[SM] web site, you can use a search engine by typing the following syntax in the search field:

search-term `site:docs.sun.com`

For example, to search for "broker," type the following:

```
broker site:docs.sun.com
```

To include other Sun web sites in your search (for example, java.sun.com, www.sun.com, and developers.sun.com), use `sun.com` in place of `docs.sun.com` in the search field.

Accessing Sun Resources Online

For product downloads, professional services, patches and support, and additional developer information, go to the following:

Download Center
: `http://wwws.sun.com/software/download`

Sun Enterprise Services, Solaris Patches, and Support
: `http://sunsolve.sun.com/`

Developer Information
: `http://developers.sun.com/prodtech/index.html`

Contacting Sun Technical Support

If you have technical questions about this product that are not answered in the product documentation, go to:

`http://www.sun.com/service/contacting`

Related Third-Party Web Site References

Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused or alleged to be caused by or in connection with use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Documentation, Support, and Training

Sun Function	URL	Description
Documentation	`http://www.sun.com/documentation/`	Download PDF and HTML documents, and order printed documents
Support and Training	`http://www.sun.com/training/`	Obtain technical support, download patches, and learn about Sun courses

Typographic Conventions

The following table describes the typographic changes that are used in this book.

TABLE P–3 Typographic Conventions

Typeface or Symbol	Meaning	Example
`AaBbCc123`	The names of commands, files, and directories, and onscreen computer output	Edit your `.login` file. Use `ls -a` to list all files. `machine_name% you have mail.`
`AaBbCc123`	What you type, contrasted with onscreen computer output	`machine_name%` **`su`** `Password:`
aabbcc123	Placeholder: replace with a real name or value	The command to remove a file is `rm` *filename*.
AaBbCc123	Book titles, new terms, and terms to be emphasized	Read Chapter 6 in the *User's Guide*. Perform a *patch analysis*. Do *not* save the file. [Note that some emphasized items appear bold online.]

Shell Prompts in Command Examples

The following table shows the default system prompt and superuser prompt for the C shell, Bourne shell, and Korn shell.

TABLE P–4 Shell Prompts

Shell	Prompt
C shell prompt	`machine_name%`
C shell superuser prompt	`machine_name#`
Bourne shell and Korn shell prompt	`$`
Bourne shell and Korn shell superuser prompt	`#`

Default Path and Directory Names

Policy Agent Software: Path and Directory Names

The Policy Agent software documentation uses the terms listed in the table that follows to represent default path and directory names.

TABLE P–5 Default Paths and Directory Names for Policy Agent Software

Term	Description
Agent-HomeDirectory	This place holder represents the directory you choose in which to unpack the Policy Agent binaries.
PolicyAgent-base	This place holder represents the directory that holds all the agent-specific information. The path for this directory includes information that helps specify that particular agent. Therefore, the path information varies for each agent. While the following J2EE agent directory is agent specific, it merely serves as an example: *Agent-HomeDirectory*`/j2ee_agents/appserver_v9_agent` If you are configuring an agent other than the one shown in the preceding example, *PolicyAgent-base* will represent a different path, but the general structure of the path will be the same.

TABLE P–5 Default Paths and Directory Names for Policy Agent Software (Continued)

Term	Description
AgentInstance-Dir	This place holder represents the directory that holds all the information that is specific to an agent installation. Therefore, the *PolicyAgent-base* directory holds all the information related to a specific agent. However, if you install that same agent more than once in the same location, each installation has information specific to that installation. That specific information is held in the *AgentInstance-Dir* directory. For example: *PolicyAgent-base*/*AgentInstance-Dir* where *AgentInstance-Dir* refers to an agent instance directory, which is usually similar to the following: `Agent_001`.

OpenSSO Enterprise Server: Path and Directory Names

The OpenSSO Enterprise server documentation uses the terms listed in the table that follows to represent default path and directory names.

TABLE P–6 Default Paths and Directory Names for OpenSSO Enterprise Server

Term	Description
zip-root	Represents the directory where the `opensso.zip` file is unzipped.

OpenSSO-Deploy-base	Represents the deployment directory where the web container deploys the `opensso.war` file.
	This value varies depending on the web container. To determine the value of *OpenSSO-Deploy-base*, view the file name in the `.openssocfg` directory, which resides in the home directory of the user who deployed the `opensso.war` file. For example, consider this scenario with Application Server 9.1 as the web container:
	■ Application Server 9.1 is installed in the default directory: `/opt/SUNWappserver`.
	■ The `opensso.war` file is deployed by super user (`root`) on Application Server 9.1.
	The `.openssocfg` directory is in the `root` home directory (`/`), and the file name in `.openssocfg` is:
	`AMConfig_opt_SUNWappserver_domains_domain1_applications_j2ee-modules_opensso_`
	Then, the value for *OpenSSO-Deploy-base* is:
	`/opt/SUNWappserver/domains/domain1/applications/j2ee-modules/opensso`

TABLE P–6 Default Paths and Directory Names for OpenSSO Enterprise Server *(Continued)*

Term	Description
ConfigurationDirectory	Represents the name of the configuration directory specified during the initial configuration of OpenSSO Enterprise server instance using the Configurator. The default is `opensso` in the home directory of the user running the Configurator. Thus, if the Configurator is run by `root`, *ConfigurationDirectory* is `/opensso`.

Sun Welcomes Your Comments

Sun is interested in improving its documentation and welcomes your comments and suggestions.

To share your comments, go to `http://docs.sun.com` and click `Feedback`. In the online form, provide the document title and part number. The part number is a seven-digit or nine-digit number that can be found on the title page of the guide or at the top of the document.

For example, the title of this guide is the *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE Agents*, and the part number is 820-4803-12.

CHAPTER 1

Introduction to Policy Agent 3.0

This chapter introduces Policy Agent 3.0. The 8.0 release of OpenSSO Enterprise server and the 3.0 release of Policy Agent software were developed simultaneously and, therefore, are closely integrated. In fact, the Policy Agent 3.0 software set is more closely connected to the server (OpenSSO Enterprise) than ever before, making for a simplified administrative experience.

The sections that follow in this chapter highlight what is new in Policy Agent for the 3.0 release while also discussing the topic of compatibility as related to Policy Agent 3.0.

Overview of New Features in Policy Agent 3.0

Policy Agent 3.0 has the following new features and improvements:

- Centralized agent configuration

  The centralized agent configuration feature moves most of the agent configuration properties from a local agent properties file (formerly referred to as `AMAgent.properties` file) to the OpenSSO Enterprise central data repository. An agent administrator can then manage the multiple agent configurations from a central server location, using either the OpenSSO Enterprise Administration Console or the `ssoadm` command-line utility.

  The centralized agent configuration feature separates the Policy Agent 3.0 configuration data into two sets:

  - The properties required for the agent to start up and initialize itself are stored in the `OpenSSOAgentBootstrap.properties` file locally on the system where the agent is installed. For example, the agent profile name and password used to authenticate to the OpenSSO Enterprise server are stored in the bootstrap file.
  - The rest of the agent properties are stored either centrally in the OpenSSO Enterprise data repository (centralized configuration option) or locally in the `OpenSSOAgentConfiguration.properties` file (local configuration option).
- Agent groups

You can assign agents of the same type (`J2EEAgent` or `WebAgent`) from the Policy Agent 3.0 software set to an agent group. All agents in a group then selectively share a common set of configuration properties. Thus, the agent configuration and management are simplified because an administrator can manage all of the agents within a group as a single entity.

Although all agents in the same group can share the same properties, defining a few specific properties (for example, the notification URL or agent URI properties) for individual agents is probably necessary. For more information about agent groups, see "Creating an Agent Group and Enabling Agents to Inherit Properties From That Group" on page 48.

- More hot-swappable agent configuration properties

  Agents in the Policy Agent 3.0 software set have more hot-swappable configuration properties. An administrator can change a hot-swappable configuration property value for an agent without having to restart the agent's deployment container for the new value to take effect. Properties in the `OpenSSOAgentBootstrap.properties` file are not hot-swappable.

- One-level wildcard support for policy-related configurations (such as when creating a policy or adding entries to the not-enforced list)

  While the regular wildcard support applies to multiple levels in a resource, the one-level wildcard applies to only the level where it appears in a resource. For more information, see Appendix C, "Wildcard Matching in Policy Agent 3.0 J2EE Agents"

- Default agent installation option with minimal questions asked during the installation

  Default or custom installation:

  - **Default** (`agentadmin --install`): The `agentadmin` program displays a minimal number of prompts and uses default values for the other options. Use the default install option when the default option meets your deployment requirements. For more information on the `agentadmin --install` command, see "`agentadmin --install`" on page 36.
  - **Custom** (`agentadmin --custom-install`): The `agentadmin` program displays a full set of prompts, similar to those presented by the Policy Agent 2.2 installer. Use the custom install option when you want to specify values other than the default options. For more information on the `agentadmin --custom-install` command, see "`agentadmin --custom-install`" on page 37.

- Option to create the agent profile in the server during installation

  The Policy Agent 3.0 installer supports an option to create the agent profile in the OpenSSO Enterprise server during the agent installation so you don't have to create the profile manually using the OpenSSO Enterprise Console or the `ssoadm` utility. This option is available when you use the `agentadmin --custom-install` command.

- Option to lock the agent configuration properties

  Changing configuration properties can have unexpected results. Furthermore, hot-swappable properties take effect immediately. Therefore, configuration mistakes are instantly implemented. In the Policy Agent 3.0 release, you have a method for locking the configuration to help prevent such accidental changes. For more information about this option, see "Locking J2EE Agent Properties" on page 58.

- Automated migration support

  You can migrate Policy Agent 2.2 to the 3.0 version using the `agentadmin` program with the `--migrate` option. For more information about this option, see "`agentadmin --migrate`" on page 44.

Note: OpenSSO Enterprise does not support version 2.1 policy agents.

Compatibility and Coexistence of Policy Agent 3.0 with Previous Releases

This section consists of information about the compatibility and coexistence of the J2EE agents in the Policy Agent 3.0 software set with previous releases of both Access Manager and Policy Agent.

J2EE Agents in the Policy Agent 3.0 release are compatible with versions of Access Manager as described in this section.

Compatibility of Policy Agent 3.0 with Access Manager 7.1 and Access Manager 7 2005Q4

Access Manager 7.1 and Access Manager 7 2005Q4 are compatible with Policy Agent 3.0. However, because Access Manager does not support centralized agent configuration, an agent in the 3.0 release deployed with Access Manager must store the core of its configuration data locally in the `OpenSSOAgentConfiguration.properties` file.

- `local`: Configuration data is stored locally in the `OpenSSOAgentConfiguration.properties` file on the server where the agent is deployed.
- `centralized`: Configuration data is stored in the OpenSSO Enterprise centralized data repository.

Note – For both configurations, the `OpenSSOAgentBootstrap.properties` file on the server where the agent is deployed contains the information required for the agent to start and initialize itself.

Coexistence of Policy Agent 3.0 With Policy Agent 2.2

OpenSSO Enterprise supports both Policy Agent 3.0 and Policy Agent 2.2 in the same deployment.

Note – Be aware that while Policy Agent 3.0 and Policy Agent 2.2 can exist in the same deployment, they cannot exist on the same container.

However, agents in the 2.2 release only have the option to store their configuration data locally in the `AMAgent.properties` file. Therefore, the OpenSSO Enterprise centralized agent configuration option is not supported. To configure an agent in the Policy Agent 2.2 release, you must edit the `AMAgent.properties` file.

For more information about Policy Agent 2.2, see the documentation collection: `http://docs.sun.com/coll/1322.1`

CHAPTER 2

Role of J2EE Agents in the Policy Agent 3.0 Release

This guide focuses on J2EE agents. Therefore, this section provides more information about how J2EE agents function generally.

J2EE agents enable application containers to enforce authentication and authorization using OpenSSO Enterprise services. To ensure secure client access to hosted J2EE applications, J2EE agents enforce the following:

- J2EE Declarative and Programmatic Security (defined in the deployment descriptor of individual applications)
- URL Policies (defined in OpenSSO Enterprise)
- Single sign-on (SSO)

This chapter provides information about J2EE agents for the 3.0 release of Policy Agent as follows:

Uses of J2EE Agents

J2EE agents can protect a variety of hosted J2EE applications, which can in turn require policy implementation that varies greatly from application to application. The security infrastructure of J2EE provides declarative as well as programmatic security that is platform-independent and is supported by all the J2EE-compliant deployment containers. For details on how to use the declarative and programmatic security of the J2EE platform, refer to J2EE documentation, available at `http://java.sun.com/j2ee`.

The way J2EE agents function in the 3.0 release can be quite different when compared to previous releases because of the option of storing the configuration in the central data repository of the OpenSSO Enterprise server. However the purpose of J2EE agents has not changed significantly. The agent configurations are easier to manage, but the basic purpose is the same.

J2EE agents help enable role-to-principal mapping for protected J2EE applications with OpenSSO Enterprise principals. Thus at runtime, when a J2EE policy is evaluated, it is done against the information available in OpenSSO Enterprise. Using this functionality, administrators can configure their hosted J2EE applications to be protected by the agent, which provides real security services and also other key features such as single sign-on. Apart from enabling the J2EE security for hosted applications, the J2EE agents also provide complete support for OpenSSO Enterprise based URL policies for enforcing access control over web resources hosted in the deployment container.

The following examples demonstrate how the J2EE agents can be put to use.

J2EE Agents and an Online Auction Application

Consider a web-based application that facilitates the auction of various kinds of merchandise between interested parties. A simple implementation for such an application will require the users to be in one of three abstract roles, namely Buyer, Seller, or Administrator. Buyers in this application will have access to web pages that display the listed auction items, whereas the Sellers may have access to web pages that allow them to list their merchandise for new auctions. The Administrators may have access to yet another set of web pages that allow them to finalize or cancel existing auctions in whatever state they may be in. Using the deployment descriptors, the application developer can express this intent by protecting such components using abstract security role names.

These abstract role names in turn can be mapped to real principals in a J2EE agent. For example, the role Buyer may be mapped to an OpenSSO Enterprise role called Bidder, the role Seller to an OpenSSO Enterprise role called Vendor, and the role Administrator to an OpenSSO Enterprise role called Admin. The abstract role names used by the application developer can be used to protect the necessary web pages and any specialized Enterprise JavaBeans (EJB) components from unauthorized access by using declarative as well as programmatic security. Once this application is deployed and configured, the agent will ensure that only the authorized personnel get access to these protected resources.

For example, access to the pages meant for Sellers to list their merchandise for auctions will be granted to user Deepak only if this user belongs to the OpenSSO Enterprise role called Vendor. Similarly, users Scott and Gina can place bids on this listed item only if they belong to the role called Bidder. Once the auction period expires, the auction can be finalized by user Krishnendu only if he is in the role called Admin.

J2EE Agents and a Web-Based Commerce Application

A web-based commerce application may have a variety of specialized EJB components that offer a spectrum of services to clients. For instance, there could be a specialized component that enables the creation of purchase orders. Similarly, there could be a specialized component that allows the approval of purchase orders. While such components provide the basic business services for the application to function, the very nature of tasks that they accomplish requires a security policy to enforce appropriate use of such services. Using the deployment descriptors, the application vendor or developer can express this intent by protecting such components using abstract security role names. For example, a developer can create a role called Buyer to protect the component that allows the creation of a purchase order and a role called Approver to protect the component that enables the approval of a purchase order. While these roles convey the intent of an application developer to enforce such security policies, they will not be useful unless these abstract role names are mapped to real life principals such as actual users or actual roles that reside in OpenSSO Enterprise.

The J2EE agent enables the container to enforce such a runtime linkage of abstract security roles to real life principals. Once the agent is installed and configured, the application security roles can be mapped to real principals. For example, the role Buyer is mapped to an OpenSSO Enterprise role called Staff, and the role Approver is mapped to an OpenSSO Enterprise role called Manager. Thus when user Arvind tries to access the application's protected resources to create a purchase order, the agent allows this access only if this user is a member of the mapped role Staff. Similarly, a user Jamie may wish to approve this purchase order, which will be allowed by the agent only if this user is a member of the mapped role Manager.

J2EE Agents and a Content-Based Web Application

A content-based web application can offer pay per-view services. The application may be partitioned into two domains: the public domain that is accessible to anonymous users, and the private domain that is accessible only to the subscribers of this particular service. Furthermore, the protected domain of this application can also be subject to strict conditions based on how the user has authenticated, the time of day, IP address-based conditions and so on. Using OpenSSO Enterprise based URL policies for web resources, an administrator specifies such complex policies for the application resources, which are evaluated by the agent in order to ensure that access to these resources is granted only when all conditions are satisfied. An administrator can set policies that govern access to these resources at any level of granularity, such as that for a single user or for an entire organization. For example, one such policy may govern access to certain resources in such a manner that the user must belong to a particular LDAP Group called Customer and that the time of the day be between 9:00 am and 5:00 p.m. Thus, if user Rajeev attempts to access this resource, the agent allows access only if this user is a member of the LDAP Group Customer, and if the time of day is between 9:00 am and 5:00 p.m.

How J2EE Agents Work

All J2EE agents communicate with OpenSSO Enterprise by XML over HTTP. J2EE agents contain two main components: the agent realm and the agent filter. Together, these two components affect the operation of the deployment container and the behavior of protected applications on the deployment container.

- **Agent Filter**

  The agent filter is installed within the protected application and facilitates the enforcement of the security policies, governing the access to all resources within the protected application. Every application protected by the agent must have its deployment descriptors changed to reflect that it is configured to use the agent filter. Applications that do not have this setting are not protected by the agent and might malfunction or become unusable if deployed on a deployment container where the agent realm is installed.

  The agent realm and agent filter work in tandem with OpenSSO Enterprise to enforce J2EE security policies as well as OpenSSO Enterprise based URL policies for authentication and authorization of clients attempting to access protected J2EE applications.

  The agent provides a fully configured and ready-to-use client installation of OpenSSO Enterprise SDK for the deployment container. This SDK offers a rich set of APIs supported by OpenSSO Enterprise that can be used to create security-aware applications that are tailored to work in the security framework offered by OpenSSO Enterprise.

- **Agent Realm**

  The agent realm, which is installed as a deployment container-specific platform component, enables the deployment container to interact with principals stored in OpenSSO Enterprise. The deployment container then communicates with OpenSSO Enterprise about user profile information. The agent realm needs to be configured correctly for the agent to enforce J2EE security policies for protected applications.

Policy Decision Process for J2EE Agents

The figure that follows is a flow chart of the policy decision process for J2EE agents. This figure illustrates how a single request is processed. The chart is useful in that it demonstrates to some degree how J2EE agents function.

The chart illustrates possible scenarios that can take place when an end user makes a request for a resource. Therefore, the end user points a browser to a URL. That URL is a resource, such as a JPEG image, HTML page, JSP page, etc. When a resource is under the sphere of influence of the J2EE agent, the agent intervenes to varying degrees, depending on the specifics of the situation, checks the request, and takes the appropriate action, which culminates with the user either being allowed or denied access to the resource. The chart reflects the potential paths a request makes before finally being allowed or denied. Moreover the chart illustrates how the filter mode is involved in the resource-request process.

You can see how this J2EE agent-specific flow chart compares to the web agent flow chart as illustrated in "Examples of the Policy Decision Process by Agent Type" on page 103. The comparison gives a sense of how the two agent types differ in how they handle requests for resources.

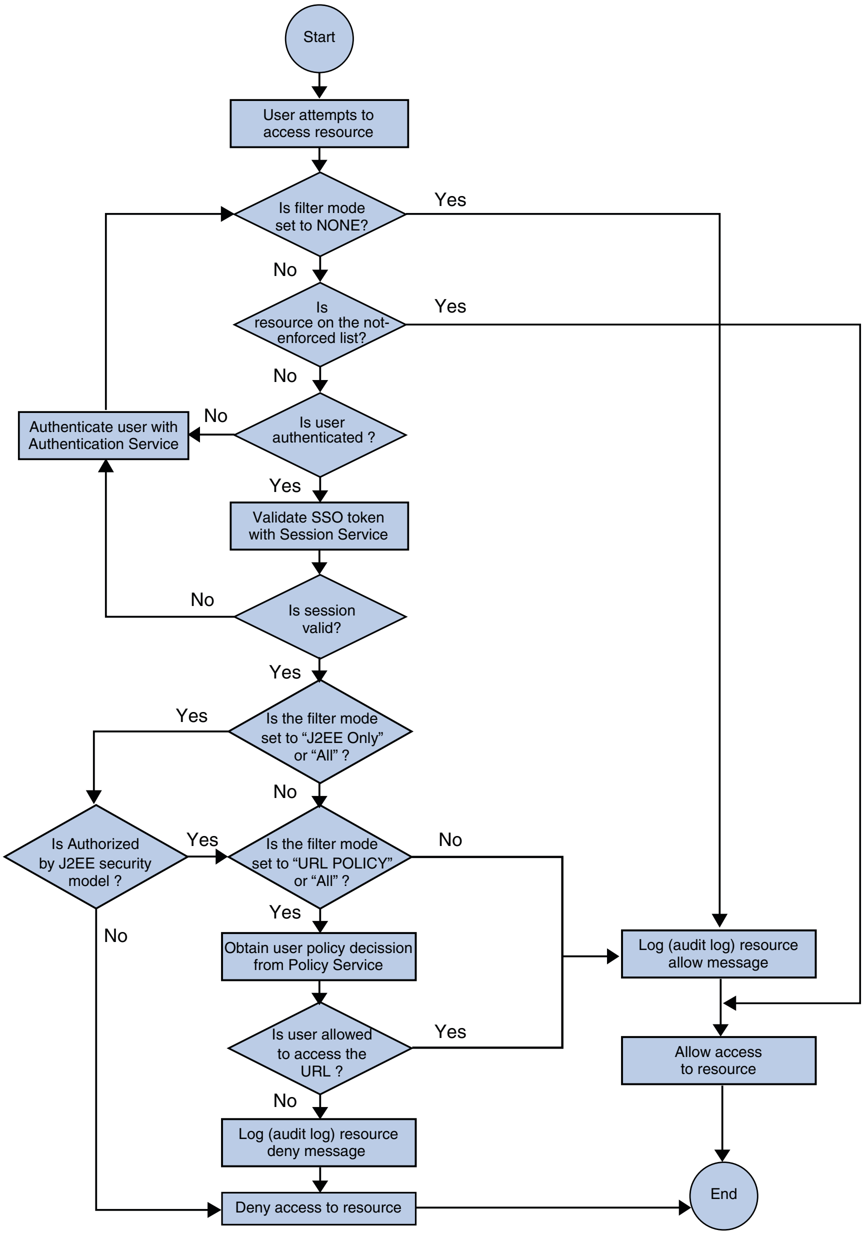

FIGURE 2–1 J2EE Agents and the Policy Decision Process

Using the J2EE Agent Sample Application in Policy Agent 3.0

Deploy, and interact with the sample application included with Policy Agent 3.0. Interacting with the sample application is perhaps the best way available to you to learn how J2EE agents work.

The sample application is deployed at *URI*`/agentsample`. The sample application demonstrates agent configuration options and features. With this application, you can view agent configuration examples and you can test if an agent was deployed successfully. To learn more about this application, read about the `sampleapp` directory explained in the list following Table 3–1. Moreover, the `sampleapp` directory includes a `readme.txt` explaining how to build and deploy the sample application. When you unpack the J2EE agent distribution, a sample application named `agentsample.ear` is created for you. The full path to this application is as follows:

PolicyAgent-base`/sampleapp/dist/agentsample.ear`

CHAPTER 3

Vital Information for J2EE Agents in the Policy Agent 3.0 Release

To facilitate the installation and configuration of a J2EE agent in Policy Agent 3.0, essential information is provided in this chapter.

This chapter applies to all the J2EE agents in the Policy Agent 3.0 release. However, throughout this chapter, when a specific J2EE agent is used for example purposes, such as in a command, only one J2EE agent is shown, Policy Agent 3.0 for Sun Java System Application Server 9.1. These examples are provided to illustrate general format. Replace J2EE agent specific information where necessary.

In simple terms, this chapter provides information to help you with the following:

- ❒ Getting the J2EE agent distribution files on the machine that hosts the deployment container. The J2EE agent is going to protect the content on that deployment container.
- ❒ Issuing install-related commands using the `agentadmin` program. The `agentadmin` program is a command-line utility that you will use to install and configure the agent. You should know the supported command options besides the more common `--install` option.
- ❒ Locating the various J2EE agent files after you get them onto the deployment container.
- ❒ Creating a J2EE agent profile.
- ❒ Updating a J2EE agent profile.
- ❒ Creating a J2EE agent group.
- ❒ Creating an agent authenticator.

The information referred to in the preceding list is described in the following sections of this chapter:

- "Unpacking the Distribution Files of an Agent in Policy Agent 3.0" on page 30
- "J2EE Agent Directory Structure in Policy Agent 3.0" on page 30
- "Role of the `agentadmin` Program in Policy Agent 3.0" on page 35
- "Creating a J2EE Agent Profile in Policy Agent 3.0" on page 47
- "J2EE Agent Task Reference for Policy Agent 3.0" on page 53

Unpacking the Distribution Files of an Agent in Policy Agent 3.0

The distribution files for an agent in Policy Agent 3.0 are provided to you in a `.zip` file, regardless of the platform on which it is to be deployed. For more information, see the individual agent guide for the specific agent you are installing.

▼ To Unpack the `.zip` File of an Agent in Policy Agent 3.0

You must download the respective agent binaries from the appropriate location. See the following download site `http://wwws.sun.com/software/download`.

Follow the steps described in this task to unpack an agent `.zip` file.

1. **Log in to the server where you want to install the agent.**

2. **Create a directory in which to unzip the agent distribution file.**

3. **Download and unzip the agent distribution file.**

   In terms of unzipping the file, the following command is one possible method:

   `unzip` *agent_download*`.zip`

   where *agent_download* represents specific information about that specific agent.

More Information Purpose of This Task

The preceding steps unpack the archive and provide you with the agent deliverables for Policy Agent 3.0.

J2EE Agent Directory Structure in Policy Agent 3.0

The Policy Agent installation directory is referred to as the Policy Agent base directory (or *PolicyAgent-base* in code examples). The location of this directory and its internal structure are important facts that are described in this section.

Location of the J2EE Agent Base Directory in Policy Agent 3.0

Unpacking the J2EE agent binaries creates a directory named `j2ee_agents`, within which an agent-specific directory is created. For example, if the J2EE agent being installed is Policy Agent

3.0 for Sun Java System Application Server 9.1, the directory created is named `appserver_v9_agent`. For other J2EE agents, the directory name is slightly different, but the naming format is the same.

This agent-specific directory is the Policy Agent base directory, referred to throughout this guide as the *PolicyAgent-base* directory. For the full path to the *PolicyAgent-base* directory, see Example 3–1.

EXAMPLE 3–1 Policy Agent Base Directory

The directory you choose in which to unpack the J2EE agent binaries is referred to here as *Agent-HomeDirectory*. The following path is an example of the location for the *PolicyAgent-base* directory of Policy Agent 3.0 for Sun Java System Application Server 9.1:

Agent-HomeDirectory`/j2ee_agents/appserver_v9_agent`

For other J2EE agents, the directory names are different, but the naming format is the same. References in this book to the *PolicyAgent-base* directory are references to the preceding path.

Inside the J2EE Agent Base Directory in Policy Agent 3.0

After you finish installing an agent by issuing the `agentadmin --install` command or the `agentadmin --custom-install` command and interacting with the installer, you will need to access J2EE agent files in order to configure and otherwise work with the product. Within the Policy Agent base directory are various subdirectories that contain all agent configuration and log files. The structure of the Policy Agent base directory for a J2EE agent is illustrated in Table 3–1.

The list that follows the table provides information about many of the items in the example Policy Agent base directory. The Policy Agent base directory is represented in code examples as *PolicyAgent-base*. The full path to any item in this directory is as follows:

PolicyAgent-base/*item-name*

where *item-name* represents the name of a file or subdirectory. For example, the full path to the `bin` directory is as follows:

PolicyAgent-base`/bin`

TABLE 3–1 Example of Policy Agent Base Directory for a J2EE Agent

Directory Contents: Files and Subdirectories	
Agent_001	installer-logs
README.TXT	lib
bin	license.txt
config	locale
data	sampleapp
etc	

The preceding example of *PolicyAgent-base* lists files and directories you are likely to find in this directory. The notable items in this directory are summarized in the list that follows:

`sampleapp`
: This directory contains the sample application included with Policy Agent 3.0. This application is extremely useful. Not only does it demonstrate configuration options and features, but the application can be used to test if an agent is running.

  Use the sample application that comes with the agent or build the application from scratch. Find instructions for building, deploying, and running this application at the following location:

  PolicyAgent-base`/sampleapp/readme.txt`

  The full path to the sample application is as follows:

  PolicyAgent-base`/sampleapp/dist/agentsample.ear`

  For more information about the sample application, see "Using the J2EE Agent Sample Application in Policy Agent 3.0" on page 27.

`bin`
: This directory contains the `agentadmin` script for the agent bits. You will use this script a great deal. For details about the tasks performed with this script, see "Role of the `agentadmin` Program in Policy Agent 3.0" on page 35.

`etc`
: This directory contains the `agentapp` file (specifically, the `agentapp.war` or `agentapp.ear` file), which has to be deployed after installation is complete. This application helps the agent perform certain housekeeping tasks.

`installer-logs`
: This directory contains various log files, including log files created when you issue the `agentadmin` command.

This directory also contains the installation log file. For the 3.0 release of Policy Agent, log information is stored in the installation log file after you install a J2EE agent instance. The following is the location of this log file:

PolicyAgent-base`/installer-logs/audit/install.log`

`lib` — The `lib` directory has a list of all the agent libraries that are used by the installer as well as the agent run time.

`locale` — This directory has all the agent installer information as well as agent run time specific locale information pertaining to the specific agent.

`data` — This directory has all the installer specific data.

Caution – Do not edit any of the files in the `data` directory under any circumstance. If this directory or any of its content loses data integrity, the `agentadmin` program cannot function normally.

`Agent_001` — The full path for this directory is as follows:

PolicyAgent-base/*AgentInstance-Dir*

where *AgentInstance-Dir* refers to an agent instance directory, which in this case is `Agent_001`.

Note – This directory does not exist until you successfully install the first instance of a J2EE agent. Once you have successfully executed one run of the installer (`agentadmin --install` command or the `agentadmin --custom-install` command), an agent specific directory, `Agent_00x` is created in the Policy Agent base directory. This directory is uniquely tied to an instance of the deployment container, such as an application server instance. Depending on the number of times the installer is run, the number that replaces the `x` in the `Agent_00x` directory name will vary.

After you successfully install the first instance of a J2EE agent, an agent instance directory named `Agent_001` appears in the Policy Agent base directory. The path to this directory is as follows:

PolicyAgent-base`/Agent_001`

The next installation of the agent creates an agent instance directory named `Agent_002`. The directories for uninstalled agents are not automatically removed. Therefore, if `Agent_001` and `Agent_002` are uninstalled, the next agent instance directory is `Agent_003`.

Agent instance directories contain directories named `config` and `logs`.

Note – When a J2EE agent is uninstalled, the `config` directory is removed from the agent instance directory but the `logs` directory still exists.

The following table is an example of the contents of an agent instance, such as `Agent_001`, directory.

Example of an Agent Instance (`Agent_001`) Directory
`logs`
`config`

`logs` Two subdirectories exist within this directory as follows:

- `audit` This directory contains the local audit trail for the agent instance.
- `debug` This directory has all the agent-specific debug information. When the agent runs in full debug mode, this directory stores all the debug files that are generated by the agent code.

  By default, the agent writes all the debug information into one file, `debug.out`. If you change the property `com.sun.services.debug.mergeall` in the `OpenSSOAgentBootstrap.properties` from "off" to "on", then each agent component writes into its own debug file.

`config` This directory contains the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file, which are specific to the agent instance. The `OpenSSOAgentBootstrap.properties` file applies regardless of the agent configuration: centralized in the OpenSSO Enterprise server or local to the agent. However, the `OpenSSOAgentConfiguration.properties` file is only meaningful when an agent instance is configured locally. In that scenario, the `OpenSSOAgentConfiguration.properties` file holds the key to the agent behavior at runtime.

Role of the `agentadmin` Program in Policy Agent 3.0

The `agentadmin` utility is the predominant install and configuration tool forPolicy Agent 3.0. The most basic of tasks, such as installation and uninstallation can be performed with this tool.

Note – Installation and configuration tasks that can be performed with the `agentadmin` utility can also be performed with the OpenSSO Enterprise `ssoadm` utility. For more information, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."

The location of the `agentadmin` program is as follows:

PolicyAgent-base/bin

For information about the *PolicyAgent-base* directory, see "Default Path and Directory Names" on page 13.

The following information about `agentadmin` program demonstrates the scope of this utility:

- All agent installation and uninstallation can be achieved with the `agentadmin` command.
- All tasks performed by the `agentadmin` program, except those involving uninstallation, require the acceptance of a license agreement. This agreement is only presented the first time you use the program.
- The following table lists options that can be used with the `agentadmin` command and gives a brief description of the specific task performed with each option.

  A detailed explanation of each option follows the table.

TABLE 3–2 The `agentadmin` Program: Supported Options

Option	Task Performed
`--install`	Installs a new agent instance
`--custom-install`	Installs a new agent instance
`--uninstall`	Uninstalls an existing Agent instance
`--listAgents`	Displays details of all the configured agents
`--agentInfo`	Displays details of the agent corresponding to the specified agent IDs
`--version`	Displays the version information
`--encrypt`	Encrypts a given string
`--getEncryptKey`	Generates an Agent Encryption key

TABLE 3–2 The agentadmin Program: Supported Options (Continued)

Option	Task Performed
--uninstallAll	Uninstalls all agent instances
--migrate	Migrates agent to a newer version
--usage	Displays the usage message
--help	Displays a brief help message

agentadmin --install

This section demonstrates the format and use of the agentadmin command with the --install option. The --install option is very similar to the --custom-install option. However, when you install an agent using the agentadmin --install command, the installer provides you with a minimal number of prompts and uses default values for the other options. For example, with the --custom-install option, you can create the agent profile during agent installation. You do not have this option with the --install option.

EXAMPLE 3–2 Command Format: agentadmin --install

The following example illustrates the format of the agentadmin command with the --install option:

./agentadmin --install [--useResponse] [--saveResponse] *filename*

The following arguments are supported with the agentadmin command when using the --install option:

--saveResponse
Use this argument to save all supplied responses to a state file, or response file, represented as *filename* in command examples. The response file, or state file, can then be used for silent installations.

--useResponse
Use this argument to install an agent in silent mode as all installer prompts are answered with responses previously saved to a response file, represented as *filename* in command examples. When this argument is used, the installer runs in non-interactive mode. At which time, user interaction is not required.

filename
Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the --saveResponse argument and provides your responses when this argument is used in conjunction with the --useResponse argument.

EXAMPLE 3–3 Command Usage: `agentadmin --install`

When you issue the `agentadmin` command, you can choose the `--install` option. With the `--install` option, you can choose the `--saveResponse` argument, which requires a file name be provided. The following example illustrates this command when the file name is `myfile`:

```
./agentadmin --install --saveResponse myfile
```

Once the installer has executed the preceding command successfully, the responses are stored in a state file that can be used for later runs of the installer.

If desired, you can modify the state file and configure the second installation with a different set of configuration parameters.

Then you can issue another command that uses the `./agentadmin --install` command and the name of the file that you just created with the `--saveResponse` argument. The difference between the previous command and this command is that this command uses the `--useResponse` argument instead of the `--saveResponse` argument. The following example illustrates this command:

```
./agentadmin --install --useResponse myfile
```

With this command, the installation prompts run the installer in silent mode, registering all debug messages in the install logs directory.

agentadmin --custom-install

This section demonstrates the format and use of the `agentadmin` command with the `--custom-install` option. The `--custom-install` option is very similar to the `--install` option. However, when you install an agent using the `agentadmin --custom-install` command, the installer provides you with a greater number of prompts, and therefore allows you to select a greater number of settings. The `--install` option, on the other hand, selects default values for many options. For example, with the `--custom-install` option, you can create the agent profile during agent installation. You do not have this option with the `--install` option.

Note – The arguments available with the `--custom-install` option, as listed in the next section, are the same as for the `--install` option.

EXAMPLE 3–4 Command Format: `agentadmin --custom-install`

The following example illustrates the format of the `agentadmin` command with the `--custom-install` option:

EXAMPLE 3–4 Command Format: `agentadmin --custom-install` *(Continued)*

```
./agentadmin --custom-install [--useResponse] [--saveResponse] filename
```

The following arguments are supported with the `agentadmin` command when using the `--custom-install` option:

`--saveResponse` Use this argument to save all supplied responses to a state file, or response file, represented as *filename* in command examples. The response file, or state file, can then be used for silent installations.

`--useResponse` Use this argument to install an agent in silent mode as all installer prompts are answered with responses previously saved to a response file, represented as *filename* in command examples. When this argument is used, the installer runs in non-interactive mode. At which time, user interaction is not required.

filename Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the `--saveResponse` argument and provides your responses when this argument is used in conjunction with the `--useResponse` argument.

EXAMPLE 3–5 Command Usage: `agentadmin --custom-install`

When you issue the `agentadmin` command, you can choose the `--custom-install` option. With the `--custom-install` option, you can choose the `--saveResponse` argument, which requires a file name be provided. The following example illustrates this command when the file name is `myfile`:

```
./agentadmin --custom-install --saveResponse myfile
```

Once the installer has executed the preceding command successfully, the responses are stored in a state file that can be used for later runs of the installer.

If desired, you can modify the state file and configure the second installation with a different set of configuration parameters.

Then you can issue another command that uses the `./agentadmin --custom-install` command and the name of the file that you just created with the `--saveResponse` argument. The difference between the previous command and this command is that this command uses the `--useResponse` argument instead of the `--saveResponse` argument. The following example illustrates this command:

```
./agentadmin --custom-install --useResponse myfile
```

EXAMPLE 3–5 Command Usage: `agentadmin --custom-install` *(Continued)*

With this command, the installation prompts run the installer in silent mode, registering all debug messages in the install logs directory.

`agentadmin --uninstall`

This section demonstrates the format and use of the `agentadmin` command with the `--uninstall` option.

EXAMPLE 3–6 Command Format: `agentadmin --uninstall`

The following example illustrates the format of the `agentadmin` command with the `--uninstall` option:

`./agentadmin --uninstall [--useResponse] [--saveResponse]` *filename*

The following arguments are supported with the `agentadmin` command when using the `--uninstall` option:

`--saveResponse`	Use this argument to save all supplied responses to a state file, or response file, represented as *filename* in command examples. The response file, or state file, can then be used for silent uninstallations.
`--useResponse`	Use this argument to uninstall an agent in silent mode as all uninstaller prompts are answered with responses previously saved to a response file, represented as *filename* in command examples. When this argument is used, the uninstaller runs in non-interactive mode. At which time, user interaction is not required.
filename	Use this argument to specify the name of a file that will be created as part of the processing of this command. This file stores your responses when this argument is used in conjunction with the `--saveResponse` argument and provides your responses when this argument is used in conjunction with the `--useResponse` argument.

EXAMPLE 3–7 Command Usage: `agentadmin --uninstall`

When you issue the `agentadmin` command, you can choose the `--uninstall` option. With the `--uninstall` option, you can choose the `--saveResponse` argument, which requires a file name be provided. The following example illustrates this command where the file name is `myfile`:

```
./agentadmin --uninstall --saveResponse myfile
```

EXAMPLE 3–7 Command Usage: `agentadmin --uninstall` *(Continued)*

Once the uninstaller has executed the preceding command successfully, the responses are stored in a state file that can be used for later runs of the uninstaller.

If desired, you can modify the state file and configure the second uninstallation with a different set of configuration parameters.

Then you can issue another command that uses the `./agentadmin --uninstall` command and the name of the file that you just created with the `--saveResponse` argument. The difference between the previous command and this command is that this command uses the `--useResponse` argument instead of the `--saveResponse` argument. The following example illustrates this command:

```
./agentadmin --uninstall --useResponse myfile
```

With this command, the uninstallation prompts run the uninstaller in silent mode, registering all debug messages in the install logs directory.

agentadmin --listAgents

This section demonstrates the format and use of the `agentadmin` command with the `--listAgents` option.

EXAMPLE 3–8 Command Format: `agentadmin --listAgents`

The following example illustrates the format of the `agentadmin` command with the `--listAgents` option:

```
./agentadmin --listAgents
```

No arguments are currently supported with the `agentadmin` command when using the `--listAgents` option.

EXAMPLE 3–9 Command Usage: `agentadmin --listAgents`

Issuing the `agentadmin` command with the `--listAgents` option provides you with information about all the configured agents on that deployment container.

agentadmin --agentInfo

This section demonstrates the format and use of the `agentadmin` command with the `--agentInfo` option.

EXAMPLE 3–10 Command Format: `agentadmin --agentInfo`

The following example illustrates the format of the `agentadmin` command with the `--agentInfo` option:

```
./agentadmin --agentInfo AgentInstance-Dir
```

The following argument is supported with the `agentadmin` command when using the `--agentInfo` option:

AgentInstance-Dir Use this option to specify which agent instance directory, therefore which agent instance, such as `Agent_002`, you are requesting information about.

EXAMPLE 3–11 Command Usage: `agentadmin --agentInfo`

Issuing the `agentadmin` command with the `--agentInfo` option provides you with information on the specific agent instance that you name in the command. For example, if you want information about an agent instance named `Agent_002`, you can issue the command illustrated in the following example:

```
./agentadmin --agentInfo Agent_002
```

agentadmin --version

This section demonstrates the format and use of the `agentadmin` command with the `--version` option.

EXAMPLE 3–12 Command Format: `agentadmin --version`

The following example illustrates the format of the `agentadmin` command with the `--version` option:

```
./agentadmin --version
```

No arguments are currently supported with the `agentadmin` command when using the `--version` option.

EXAMPLE 3–13 Command Usage: `agentadmin --version`

Issuing the `agentadmin` command with the `--version` option provides you with version information for the configured agents on that deployment container. For example, the `agentadmin --version` command provides the version of the agent, such as 3.0 and the build date of the agent.

agentadmin --encrypt

This section demonstrates the format and use of the `agentadmin` command with the `--encrypt` option.

EXAMPLE 3–14 Command Format: `agentadmin --encrypt`

The following example illustrates the format of the `agentadmin` command with the `--encrypt` option.

```
./agentadmin --encrypt AgentInstance-Dir agentpasswordfile
```

The following arguments are supported with the `agentadmin` command when using the `--encrypt` option:

AgentInstance-Dir Use this option to specify which agent instance directory, therefore which agent instance such as `Agent_002`, for which the given password file will be encrypted. Encryption functionality requires that an encryption key be available for an agent instance. Therefore, a default encryption key can be assigned by the agent during agent installation. You can also assign an encryption key yourself. The encryption key is stored in the `OpenSSOAgentBootstrap.properties` file. For example:

```
am.encryption.pwd = EphgFHmF6X3XmMjYGCUtYHSYA9C7qQlk
```

agentpasswordfile Use this option to specify the full path to the password file that contains a clear text agent password to be encrypted.

The password file should be created as an agent pre-installation task.

EXAMPLE 3–15 Command Usage: `agentadmin --encrypt`

Issuing the `agentadmin` command with the `--encrypt` option enables you to change the password for an existing agent profile in OpenSSO Enterprise after the agent is installed.

For example, issuing the following command encrypts the password file, `pwfile1` for the agent instance directory `Agent_001`:

```
./agentadmin --encrypt Agent_001 pwfile1
```

The following is an example of an encrypted value:

```
ASEWEJIowNBJHTv1UGD324kmT==
```

Each agent uses a unique agent ID and password to communicate with OpenSSO Enterprise. Once the agent profile for a specific agent has been created in OpenSSO Enterprise, the installer assigns the Policy Agent profile name and encrypted password in the respective agent instance.

EXAMPLE 3–15 Command Usage: agentadmin --encrypt *(Continued)*

If you choose a new password for the Policy Agent profile, encrypt it and enter that encrypted password in the `OpenSSOAgentBootstrap.properties` as the value for the following property:

```
com.iplanet.am.service.secret
```

agentadmin --getEncryptKey

This section demonstrates the format and use of the `agentadmin` command with the `--getEncryptKey` option.

EXAMPLE 3–16 Command Format: agentadmin --getEncryptKey

The following example illustrates the format of the `agentadmin` command with the `--getEncryptKey` option:

```
./agentadmin --getEncryptKey
```

No arguments are currently supported with the `agentadmin` command when using the `--getEncryptKey` option.

EXAMPLE 3–17 Command Usage: agentadmin --getEncryptKey

This option may be used in conjunction with the `--encrypt` option to encrypt and decrypt sensitive information in the `OpenSSOAgentBootstrap.properties` file. Issuing the `agentadmin` command with the `--getEncryptKey` option generates a new encryption key for the agent.

For example, the following text demonstrates the type of output that would result from issuing this command:

```
./agentadmin -getEncryptKey


Agent Encryption Key : k1441g4EejuOgsPlFOSg+m6P5x7/G9rb
```

The encryption key is stored in the `OpenSSOAgentBootstrap.properties` file. Therefore, once you generate a new encryption key, use it to replace the value of the property that is currently used to store the encryption key. The following property in the `OpenSSOAgentBootstrap.properties` file stores the encryption key:

```
am.encryption.pwd
```

EXAMPLE 3–17 Command Usage: `agentadmin --getEncryptKey` *(Continued)*

For example, using the encryption key example provided previously, updating the encryption key value for the applicable agent property could appear as follows:

```
am.encryption.pwd = k1441g4EejuOgsPlFOSg+m6P5x7/G9rb
```

Once you have updated the `OpenSSOAgentBootstrap.properties` file with the new encryption key, issue the `agentadmin --encrypt` command to actually encrypt a password. The `--encrypt` option uses the encryption key in its processing.

agentadmin --uninstallAll

This section demonstrates the format and use of the `agentadmin` command with the `--uninstallAll` option.

EXAMPLE 3–18 Command Format: `agentadmin --uninstallAll`

The following example illustrates the format of the `agentadmin` command with the `--uninstallAll` option:

```
./agentadmin --uninstallAll
```

No arguments are currently supported with the `agentadmin` command when using the `--uninstallAll` option.

EXAMPLE 3–19 Command Usage: `agentadmin --uninstallAll`

Issuing the `agentadmin` command with the `--uninstallAll` option runs the agent uninstaller in an iterative mode, enabling you to remove select agent instances or all agent instances on that deployment container. You can exit the recursive uninstallation process at any time.

The advantage of this option is that you do not have to remember the details of each installation-related configuration. The `agentadmin` program provides you with an easy method for displaying every instance of an agent on a deployment container. You can then decide, case by case, to remove an agent instance or not.

agentadmin --migrate

This section demonstrates the format and use of the `agentadmin` command with the `--migrate` option.

EXAMPLE 3–20 Command Format: `agentadmin --migrate`

The following example illustrates the format of the `agentadmin` command with the `--migrate` option:

```
./agentadmin --migrate
```

No arguments are currently supported with the `agentadmin` command when using the `--migrate` option.

EXAMPLE 3–21 Command Usage: `agentadmin --migrate`

Issuing the `agentadmin` command with the `--migrate` option allows you to update an agent in the Policy Agent software set to a newer version of that same agent.

For example, you can migrate Policy Agent 2.2 to Policy Agent 3.0. The `agentadmin --migrate` command allows you to migrate the agent's binary files, update the agent's deployment container configuration, and convert the agent's `AMAgent.properties` file to the property files used for the 3.0 version: the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file.

agentadmin --usage

This section demonstrates the format and use of the `agentadmin` command with the `--usage` option.

EXAMPLE 3–22 Command Format: `agentadmin --usage`

The following example illustrates the format of the `agentadmin` command with the `--usage` option:

```
./agentadmin --usage
```

No arguments are currently supported with the `agentadmin` command when using the `--usage` option.

EXAMPLE 3–23 Command Usage: `agentadmin --usage`

Issuing the `agentadmin` command with the `--usage` option provides you with a list of the options available with the `agentadmin` program and a short explanation of each option. The following text is the output you receive after issuing this command:

```
./agentadmin --usage
```

EXAMPLE 3–23 Command Usage: `agentadmin --usage` *(Continued)*

```
Usage: agentadmin <option> [<arguments>]

The available options are:
--install: Installs a new Agent instance.
--custom-install: Installs a new Agent instance
--uninstall: Uninstalls an existing Agent instance.
--version: Displays the version information.
--uninstallAll: Uninstalls all the agent instances.
--migrate: migrate agent to newer one
--listAgents: Displays details of all the configured agents.
--agentInfo: Displays details of the agent corresponding to the specified
agent ID.
--encrypt: Encrypts a given string.
--getEncryptKey: Generates an Agent Encryption key.
--usage: Display the usage message.
--help: Displays a brief help message.
```

agentadmin --help

This section demonstrates the format and use of the `agentadmin` command with the `--help` option.

EXAMPLE 3–24 Command Format: `agentadmin --help`

The following example illustrates the format of the `agentadmin` command with the `--help` option:

```
./agentadmin --help
```

No arguments are currently supported with the `agentadmin` command when using the `--help` option.

EXAMPLE 3–25 Command Usage: `agentadmin --help`

Issuing the `agentadmin` command with the `--help` option provides similar results to issuing the `agentadmin` command with the `--usage` option. Both commands provide the same explanations for the options they list. With the `--usage` option, all `agentadmin` command options are explained. With the `--help` option, explanations are not provided for the `--usage` option or for the `--help` option itself.

A another difference is that the `--help` option also provides information about the format of each option while the `--usage` option does not.

Creating a J2EE Agent Profile in Policy Agent 3.0

Caution – Creating a J2EE agent profile in OpenSSO Enterprise Console is a required task that you can perform prior to installing the J2EE agent or during installation. Though the installation of the J2EE agent actually succeeds without performing this task, the lack of a valid agent profile in OpenSSO Enterprise prevents the J2EE agent from authenticating or having any further communication with OpenSSO Enterprise.

J2EE agents work with OpenSSO Enterprise to protect resources. However, for security purposes these two software components can only interact with each other to maintain a session after the J2EE agent authenticates with OpenSSO Enterprise by supplying an agent profile name and password. During the installation of the J2EE agent, you must provide a valid agent profile name and the respective password to enable authentication attempts to succeed.

You can create agent profiles using any of the following methods:

- Use OpenSSO Enterprise Console as described in the task that follows, "To Create a J2EE Agent Profile in Policy Agent 3.0 Using OpenSSO Enterprise Console" on page 47.
- Use the `ssoadm` command-line utility with the `create-agent` subcommand. For more information on the `ssoadm` command-line utility, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."
- Choose "Option to create the agent profile in the server during installation" when you run the `agentadmin --custom-install` command. For more information on the `agentadmin --custom-install` command, see "`agentadmin --custom-install`" on page 37.

Creating a J2EE Agent Profile in Policy Agent 3.0 Using OpenSSO Enterprise Console

This section provides instructions for creating a J2EE agent profile using OpenSSO Enterprise Console.

▼ To Create a J2EE Agent Profile in Policy Agent 3.0 Using OpenSSO Enterprise Console

Perform the following tasks in OpenSSO Enterprise Console. The key steps of this task involve creating an agent name (ID) and an agent password.

1. **Log in to OpenSSO Enterprise Console as a user with AgentAdmin privileges, such as `amadmin`.**

2. **Click the Access Control tab.**

3 Click the name of the realm to which the agent will belong, such as the following: /(Top Level Realm).

4 Click the Agents tab.

5 Click the J2EE tab.

6 Click New in the agent section.

7 Enter values for the following fields:

Name: Enter the name or identity of the agent. This is the agent profile name, which is the name the agent uses to log into OpenSSO Enterprise. Multi-byte names are not accepted.

Password: Enter the agent password. However, it must be the same password entered in the agent profile password file that is used by the `agentadmin` utility to install the agent.

Re-Enter Password: Confirm the password.

Configuration: For configuration, check the location of the agent configuration properties.

- **Local:** Properties stored in the `OpenSSOAgentConfiguration.properties` file on the server where the agent is deployed.
- **Centralized:** Properties stored in the OpenSSO Enterprise centralized data repository.

8 In the Server URL field, enter the OpenSSO Enterprise server URL.

For example: `http://OpenssoHost.example.com:58080/opensso`

9 In the Agent URL field, enter the URL for the agent application.

For example: `http://agentHost.example.com:8090/agentapp`

10 Click Create.

The Console creates the agent profile and displays the J2EE Agent page again with a link to the new agent profile.

To perform additional configuration of the agent, click this link to display the Edit agent page.

Creating an Agent Group and Enabling Agents to Inherit Properties From That Group

The Concept of agent groups is new in Policy Agent 3.0. You can create an agent group and then allow an agent to inherit specified properties from the group. The following tasks describe how to enable this type of property inheritance.

▼ To Create a New Group

If desired, perform this task in the OpenSSO Enterprise Administration Console. This task applies to J2EE agent groups.

Create a group if you want agents to inherit specific properties from the group. J2EE agents can inherit properties from a J2EE agent group.

1. **Click the Access Control tab.**
2. **Click the name of the realm to which the group will belong.**
3. **Click the Agents tab.**
4. **If necessary, select the J2EE Agents tab.**
5. **In the Group section, click New.**
6. **In the Name field, enter the name for the new group name.**
7. **In the Server URL field, enter the OpenSSO Enterprise server URL.**

   For example, `http://OpenssoHost.example.com:58080/opensso`.
8. **Click Create.**

   The Console creates the agent group and displays the J2EE agent page again, with a link to the group.

   To do additional configuration of the group, click this link to display the Edit agent group page.

   The properties you can set to configure a group are the same as they are for an individual agent except that the Group, Password, and Password Confirm properties are not available at the group level.

Note – Also, be aware that some group properties already have variable values assigned that in most cases should not be changed. The following is one example of such a value:

```
@AGENT_PROTO@://@AGENT_HOST@:@AGENT_PORT@/amagent
```

▼ To Enable a J2EE Agent to Inherit Properties From a Group

Before You Begin The group from which you want an agent to inherit properties must be created first.

1. **Using a browser, navigate through OpenSSO Enterprise Console to the agent properties of the J2EE agent that you want to configure.**

   For the steps to navigate to the J2EE agent properties, see "To Navigate in OpenSSO Enterprise 8.0 Console to the J2EE Agent Properties" on page 53.

2. **With the Global tab selected, for the attribute labeled Group, select the name of the group from which you want the agent to inherit properties.**

3. **Click Save.**

   At the top of the page, the Inheritance Settings button becomes active.

4. **Click Inheritance Settings.**

   A list of inheritance settings for the Global tab appear in alphabetical order.

5. **Select the properties that you want the agent to inherit from the group.**

6. **Click Save.**

Next Steps This task just describes how to change the inheritance settings for properties listed in the Global tab. For the inheritance settings of properties listed in other tabs, such as Application, click the desired tab and edit the inheritance settings in the same manner described in the preceding steps.

About the Agent Authenticator in Policy Agent 3.0

An agent authenticator is a special type of agent that, once authenticated, can have access to agent profiles that have been selected for the agent authenticator to read. Therefore, the agent authenticator has read-only access to these other agents profiles. In this case, agent profiles refer to a broad range of "agent" types (Web, J2EE, WSP, Discovery, and so forth). These agent profiles must exist in the same realm as the agent authenticator.

Users who have the agent authenticator's credentials (user name and password) can read the agent profile data, but do not have the create, update, or delete permissions of the agent administrator.

An advantage of creating an agent authenticator is that you can configure the agent authenticator to have access (read-only access) to a variety of other agents. Therefore, using a single user name and password to access the agent authenticator, you then have access to all the agents to which the agent authenticator has access.

The agent authenticator is also used when configuring the agent to support web services security. For more information, see "Web Services Security Support for J2EE Agents in Policy Agent 3.0" on page 82

For more information about the agent authenticator see the following guides:

- *Sun OpenSSO Enterprise 8.0 Administration Guide*
- *Sun OpenSSO Enterprise 8.0 Administration Reference*

The two tasks that follow describe how to create an agent authenticator, assign one or more agent profiles to the agent authenticator, and then edit the respective bootstrap files to configure the agent instances that correspond to those agent profiles.

▼ To Create an Agent Authenticator To Access Other Agent Profiles

This task details how to use OpenSSO Enterprise Console to create an agent authenticator.

Before You Begin The instructions that follow start with the assumption that OpenSSO Enterprise server and at least one agent instance have been properly installed and configured.

1. **Log in to OpenSSO Enterprise Console as a user with AgentAdmin privileges, such as `amadmin`.**

2. **Click the Access Control tab.**

3. **Click the name of the appropriate realm, such as the following: /(Top Level Realm).**

4. **Click the Agents tab.**

5. **Click on Agent Authenticator tab.**

6. **Click the New button.**

7. **Enter an agent authenticator name and password.**

8. **Click the Create button.**

9. **On the Agent Authenticator page, click the link for the newly created agent authenticator.**

   The agent authenticator page is displayed. In the section labeled "Agent Profiles allowed to Read," two lists exist: Available and Selected. The Available list has all the available agents in the system, and the Selected list has all the agents whose configurations can be read by this agent authenticator.

10 **From the available list, select one or more agent profile names.**

The agent authenticator can access any of the various agent types. Select all the agent profiles to which you would like the agent authenticator to have access.

11 **Click Add to move the selected item from the Available list to the Selected list.**

12 **Click Save.**

▼ To Enable the Agent Authenticator to Access Other Agent Instances

This task describes how to edit the bootstrap file of each agent instance that corresponds to an agent profile you added to the Selected list of the agent authenticator. Therefore, if you added four agents profiles (for example, a combination of J2EE agent and Web agent instances) to the agent authenticator, you must perform this task four times if you want each of those agent instances to be readable by the agent authenticator. In such a scenario, all four agent instances would then use the same user name and password to authenticate to OpenSSO Enterprise server.

Agents in Policy Agent 3.0 have the two following properties in the `OpenSSOAgentBootstrap.properties` file that enable the agent to communicate with OpenSSO Enterprise server:

- `com.sun.identity.agents.app.username`
- `com.sun.identity.agents.config.profilename`

The first property, the user name property, enables the agent to authenticate with the OpenSSO Enterprise server. The second property, the profile name property, enables the agent to retrieve its configuration data from the OpenSSO Enterprise server. By default, the value assigned to these two properties is the same. However, for the agent authenticator, these properties should have different values. Therefore, the user name property must be changed as indicated in this task.

1 **Stop the agent container.**

2 **Edit the `OpenSSOAgentBootstrap.properties` file as described in the substeps that follow.**

The bootstrap file is located at the following location:

PolicyAgent-base/*AgentInstance-Dir*/`config`

For information about this location, see Table P–6

a. **Using your text editor of choice, open the `OpenSSOAgentBootstrap.properties` file.**

b. **Change the value for the property named** `com.sun.identity.agents.app.username` **to the agent authenticator name.**

   Therefore, the setting would be as such:

   `com.sun.identity.agents.app.username =` *AgentAuthenticatorName*

   where *AgentAuthenticatorName* represents the name provided for the agent authenticator.

c. **Change the value for the property named** `com.iplanet.am.service.secret` **to the agent authenticator password.**

   Therefore, the setting would be as such:

   `com.iplanet.am.service.secret =` *EncryptedAgentAuthenticatorPassword*

   where *EncryptedAgentAuthenticatorPassword* represents the encrypted version of the password provided for the agent authenticator as demonstrated previously in this task.

   Note – To encrypt the password, use the `agentadmin --encrypt` command as described in "`agentadmin --encrypt`" on page 42.

   ---

d. **Save and close the bootstrap file.**

3 **Restart the agent container.**

J2EE Agent Task Reference for Policy Agent 3.0

This section lists tasks that are often repeated when performing various J2EE agent configurations. A task listed here might be referenced from various other sections of this guide.

▼ To Navigate in OpenSSO Enterprise 8.0 Console to the J2EE Agent Properties

1 **Log in to OpenSSO Enterprise Console as a user with AgentAdmin privileges, such as** `amadmin`**.**

The OpenSSO Enterprise login page is available at a URL similar in format to the following:

`http://OpenssoHost.example.com:58080/opensso`

2 **Click the Access Control tab.**

3 **Click the name of the realm to which the agent will belong, such as the following: /(Top Level Realm).**

4 Click the Agents tab.

5 Click the J2EE tab.

6 Click the name of the agent you are attempting to access.

After performing this step, by default, the Global tab is selected. You can start editing J2EE agent properties, moving from tab to tab as necessary.

CHAPTER 4

Common J2EE Agent Tasks and Features in Policy Agent 3.0

After installing the J2EE agent and performing the required post-installation steps, you must adjust the agent configuration to your site's specific deployment. This chapter describes how to modify J2EE agents generally. Therefore, the information in this chapter tends to apply to all J2EE agents in the Policy Agent 3.0 software set.

This chapter focuses on methods available for managing this J2EE agent, specifying the features you can configure and the tasks you can perform using each method as follows:

- "Common J2EE Agent Tasks and Features" on page 55
- "Key Features and Tasks Performed With the J2EE `agentadmin` Program" on page 94
- "Key Features and Tasks Performed With the J2EE Agent API" on page 95

Common J2EE Agent Tasks and Features

This section focuses on features that involve setting J2EE agent property values. Assigning values to properties is the key method available for configuring agent features. The topics described in this section are typically those of greatest interest in real-world deployment scenarios. This section does not cover every property. For a description of all the agent properties, see the following link: `http://wikis.sun.com/display/OpenSSO/agent3properties`

The manner in which these properties are set varies depending on if the agent configuration is centralized on the OpenSSO Enterprise server or contained locally with the agent. However, regardless of if the agent configuration is local or centralized, a small subset of properties is always stored locally with the agent in the `OpenSSOAgentBootstrap.properties` file. The properties in the bootstrap file are required for the agent to start up and initialize itself. For example, the agent profile name and password used to access the OpenSSO Enterprise server are stored in this bootstrap file. To change the values in the bootstrap file, you must edit the file directly. For information about the `ssoadm` utility, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."

In terms of the properties *not* stored in the `OpenSSOAgentBootstrap.properties` file and when the J2EE agent configuration is centralized on the OpenSSO Enterprise server, you can use the OpenSSO Enterprise Console or the `ssoadm` command-line utility to set the J2EE agent properties.

When the J2EE agent configuration is local, OpenSSO Enterprise Console and the `ssoadm` utility tool are not available for agent configuration. Instead, you must configure the majority of J2EE agent properties using the `OpenSSOAgentConfiguration.properties` file. However, the properties in the bootstrap file still apply in a local configuration. In all situations, you must edit the properties in the bootstrap file directly.

Caution – The content of the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file are very sensitive. Changes made can result in changes in how the agent works. Errors made can cause the agent to malfunction.

The following is the location of the `OpenSSOAgentBootstrap.properties` file and the `OpenSSOAgentConfiguration.properties` file:

PolicyAgent-base/*AgentInstance-Dir*/`config`

For more information about the Policy Agent 3.0 directory structure, see "J2EE Agent Directory Structure in Policy Agent 3.0" on page 30.

The following topics are discussed in this section:

How J2EE Agent Properties Are Discussed in this Guide

When this guide discusses editing a property, the emphasis is on the property as set in OpenSSO Enterprise Console. For a description of how to access the J2EE agent properties using OpenSSO Enterprise Console, see "To Navigate in OpenSSO Enterprise 8.0 Console to the J2EE Agent Properties" on page 53.

To set a property in OpenSSO Enterprise Console, the name of the Console tab in which the property is located is useful as well as the property label. The property name is not usually as important when using OpenSSO Enterprise Console, but can still be of some use and is therefore included in this guide whenever a property is mentioned.

EXAMPLE 4–1 The Method Used for Discussing J2EE Agent Properties in This Guide

Most agent properties mentioned in this guide are presented using a format that is most useful for those configuring properties using OpenSSO Enterprise Console. However, that format is not used when the property is in the `OpenSSOAgentBootstrap.properties` file since the bootstrap file is not accessible using OpenSSO Enterprise Console. The following example illustrates the format usually used in this guide to mention an agent property:

Example: The property labeled Login Form URI (Tab: Application, Name: `com.sun.identity.agents.config.login.form`).

Caution – Every time you change a property using OpenSSO Enterprise Console, you must click Save on that page for the change to take place. Furthermore, if the property is not hot-swappable, you must restart the Policy Agent container for the new value to take effect.

The above example provides the following details about the property:

Property label used in OpenSSO Enterprise Console:
Login Form URI

The OpenSSO Enterprise Console tab within which the property is located:
Application

The property name:
`com.sun.identity.agents.config.login.form`

Since the emphasis of this guide is on the use of OpenSSO Enterprise Console, other methods of editing properties are not as consistently mentioned. If you are configuring agent properties using the `ssoadm` utility, the explanations provided in this guide about the purpose of the agent properties are still applicable, however for details on configuring the agent properties using the `ssoadm` utility, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."

If the agent is configured locally, therefore using the `OpenSSOAgentConfiguration.properties` file, you cannot use OpenSSO Enterprise Console. For such a scenario, descriptions of the properties provided in this guide are still applicable and some limited information about setting the property in the `OpenSSOAgentConfiguration.properties` file is provided. However. for the most part, you should consult the `OpenSSOAgentConfiguration.properties` file itself for information about setting the properties.

Hot-Swap Mechanism in J2EE Agents

Most J2EE agent properties are hot-swap enabled. Properties are identified as hot-swappable or not in OpenSSO Enterprise Console (a centralized agent configuration option) and in the `OpenSSOAgentConfiguration.properties` file (the local agent configuration option). When changes are made to the agent configuration, the changes must be communicated to the property configuration cache for it to be updated. Hot-swap enabled (or Hot-swappable) properties allow such changes to be made without having to restart the agent server. For properties that are not hot-swappable, the new values are only picked up by the agent once the agent container is restarted.

Therefore, when hot-swappable properties are changed on the OpenSSO Enterprise server (using OpenSSO Enterprise Console or the `ssoadm` utility), the changes can be communicated using notifications or polling. However, when the agent configuration is stored with the agent locally (in the `OpenSSOAgentConfiguration.properties`), changes to the property values can only be communicated through polling. The communication of property value changes results in an update to the property configuration cache.

For notifications, the property that controls the hot-swap mechanism is labeled Agent Configuration Change Notification (Tab: Global, Name: `com.sun.identity.agents.config.change.notification.enable`) and for polling, the property that controls the hot-swap mechanism is labeled Configuration Reload Interval (Tab: Global, Name: `com.sun.identity.agents.config.load.interval`). These two properties, which both control the hot-swap mechanism, are themselves hot-swap enabled. This means that if the hot-swap mechanism is enabled by one of these properties and you change the value of the respective property, the new value will take effect.

Locking J2EE Agent Properties

Caution – Changing configuration properties can have unexpected results. Furthermore, hot-swappable properties take effect immediately. Therefore, configuration mistakes are instantly implemented.

You can lock the configuration to help prevent such accidental changes. Then you can unlock and lock the configuration as necessary.

▼ To Lock the Agent Configuration Properties

1. **Using your text editor of choice, open the `OpenSSOAgentBootstrap.properties` file.**

   The bootstrap file is available at the following location:

   PolicyAgent-base/*AgentInstance-Dir*/`config/OpenSSOAgentBootstrap.properties`

   For information about this location, see Table P–6

2. **In the bootstrap file, locate the following setting:**

   ```
   com.sun.identity.agents.config.lock.enable = false
   ```

   This property is set to false, which is unlocked, by default.

3. **Change the false setting to true to lock the properties.**

4. **Save and close the bootstrap file.**

5. **Restart the agent instance container for the change to take effect.**

J2EE Agent Properties That Are List Constructs

Certain J2EE properties are specified as lists. Knowledge of the format of these list constructs is often *not* required in order to set them. For example when you configure the properties using OpenSSO Enterprise Console, you do not interact with the "`<key>[<index>] = <value>`" formatting involved with list constructs. However, if you use OpenSSO Enterprise Console to set a list, though the formatting information provided in this section is not applicable, the general information about lists is useful.

See the following table to determine when the list construct format is required to set this property.

TABLE 4–1 Use of the List Construct Format: Required or Not

Method for Setting Properties	Location of Agent Configuration	Knowledge of List Construct Format Required
Using the OpenSSO Enterprise Console	Centralized agent configuration	NO
Using the `ssoadm` command-line utility	Centralized agent configuration	YES
Using the `OpenSSOAgentConfiguration.properties` file	Local agent configuration	YES

A list construct has the following format (Does not apply when the OpenSSO Enterprise Console is used):

```
<key>[<index>] = <value>
```

`key` The configuration key (name of the configuration property)

`index` A positive number starting from 0 that increments by 1 for every value specified in this list.

`value` One of the values specified in this list

Note – Properties that are specified in this manner must follow the preceding format, otherwise they will be treated as invalid or missing properties.

More than one property can be specified for this key by changing the value of `<index>`. This value must start from the number 0 and increment by 1 for each entry added to this list.

If certain indices are missing, those indices are ignored and the rest of the specified values are loaded at adjusted list positions.

Duplicate index values result in only one value being loaded in the indexed or adjusted indexed position.

EXAMPLE 4–2 Example of J2EE Agent Properties That Are List Constructs

```
com.sun.identity.agents.config.notenforced.uri[0]=/agentsample/public/*
com.sun.identity.agents.config.notenforced.uri[1]=/agentsample/images/*
com.sun.identity.agents.config.notenforced.uri[2]=/agentsample/index.html
```

J2EE Agent Properties That Are Map Constructs

Knowledge of the format of these map constructs is often *not* required in order to set them. For example when you configure the properties using OpenSSO Enterprise Console, you do not interact with the "`<key>[<name>]=<value>`" formatting involved with map constructs. However, if you use OpenSSO Enterprise Console to set a map, though the formatting information provided in this section is not applicable, the general information about maps is useful.

See the following table to determine when the map construct format is required to set this property.

TABLE 4–2 Use of the Map Construct Format: Required or Not

Method for Setting Properties	Location of Agent Configuration	Use of Map Construct Format Required
Using the OpenSSO Enterprise Console	Centralized agent configuration	NO
Using the `ssoadm` command-line utility	Centralized agent configuration	YES
Using the `OpenSSOAgentConfiguration.properties` file	Local agent configuration	YES

Certain J2EE agent properties are specified as maps. A map construct has the following format (Does not apply when OpenSSO Enterprise Console is used):

```
<key>[<name>]=<value>
```

`key` The configuration key (name of the configuration property)

`name` A string that forms the lookup key as available in the map

`value` The value associated with the name in the map

Note – Properties that are specified in this manner must follow the preceding format, otherwise they will be treated as invalid or missing properties.

For a given `<name>`, there may only be one entry in the configuration for a given configuration key (`<key>`). If multiple entries with the same <name> for a given configuration key are present, only one of the values will be loaded in the system and the other values will be discarded.

EXAMPLE 4–3 Example of J2EE Agent Properties That Are Map Constructs

```
com.sun.identity.agents.config.filter.mode[app1]=ALL
com.sun.identity.agents.config.filter.mode[app2]=SSO_ONLY
```

J2EE Property Configuration: Application Specific or Global

Certain J2EE agent properties can be configured for specific applications. Therefore, the agent can use different values of the same property for different applications as defined in the configuration file. Properties that are not configured for specific applications apply to all the applications on that deployment container. Such properties are called global properties.

Knowledge of the format of these application-specific constructs is often *not* required in order to set them. For example when you configure the properties using OpenSSO Enterprise Console, you do not interact with the "`<key>[<appname>]=<value>`" formatting involved with application-specific constructs. However, if you use OpenSSO Enterprise Console to set an application-specific property, though the formatting information provided in this section is not applicable, the general information about properties that can be both application-specific and global is useful.

See the following table to determine when the application-specific construct format is required to set these types of properties.

TABLE 4–3 Use of the Map Construct Format: Required or Not

Method for Setting Properties	Location of Agent Configuration	Use of Application-Specific Construct Format Required
Using the OpenSSO Enterprise Console	Centralized agent configuration	NO
Using the `ssoadm` command-line utility	Centralized agent configuration	YES
Using the `OpenSSOAgentConfiguration.properties` file	Local agent configuration	YES

An application-specific property has the following format (Does not apply when the OpenSSO Enterprise Console is used):

```
<key>[<appname>]=<value>
```

`key` The configuration key (name of the configuration property)

`appname` The application name to which this configuration belongs. The application name is the context path of the application without the leading forward slash character. In a scenario where the application has been deployed at the root-context of the server, the application name should be specified as `DefaultWebApp`.

`value` The value used by the agent to protect the application identified by the given application name

---

Note – When an application specific configuration is not present, the agent uses different mechanisms to identify a default value. Configurations are possible where the default value is used as the value specified for the same key without any application specific suffix `[<appname>].` The following settings for a single property serve as an example:

```
com.sun.identity.agents.config.example[Portal] = value1
com.sun.identity.agents.config.example[DefaultWebApp] = value2
com.sun.identity.agents.config.example = value3
```

The preceding example illustrates that for applications other than the ones deployed on the root context and the context `/Portal`, the value of the property defaults to `value3`.

Application Specific configuration properties must follow the rules and syntax of the map construct of configuration entries.

EXAMPLE 4–4 Example of Application Specific and Global Configuration

```
com.sun.identity.agents.config.example[Portal] = value1
com.sun.identity.agents.config.example[BankApp] = value2
com.sun.identity.agents.config.example[DefaultWebApp] = value3
```

J2EE Agent Filter Modes

The agent installation program and the agent property labeled Agent Filter Mode (`com.sun.identity.agents.config.filter.mode`) allow you to set the agent filter in one of the five available modes of operation. Depending upon your security requirements, choose the mode that best suits your site's deployment.

The value for the Agent Filter Mode property can be one of the following:

- `NONE`
- `SSO_ONLY`
- `J2EE_POLICY`
- `URL_POLICY`
- `ALL`

The sections that follow describe the different agent filter modes.

J2EE Agent Filter Mode-NONE

This mode of operation effectively disables the agent filter. When operating in this mode, the agent filter allows all requests to pass through. However, if the logging is enabled, the agent filter will still log all the requests that it intercepts.

Caution – This mode is provided to facilitate development and testing efforts in a controlled development or test environment. Do not use this mode of operation in a production environment at any time.

When the agent filter is operating in this mode, any declarative J2EE security policy or programmatic J2EE security API calls will return a negative result regardless of the user.

J2EE Agent Filter Mode - SSO_ONLY

This is the least restrictive mode of operation for the agent filter. In this mode, the agent simply ensures that all users who try to access protected web resources are authenticated using OpenSSO Enterprise Authentication Service.

When operating in this mode, any declarative J2EE security policy or programmatic J2EE security API calls evaluated for the application will result in negative evaluation.

J2EE Agent Filter Mode - J2EE_POLICY

In this mode, the agent filter and agent realm work together with variousOpenSSO Enterprise services to ensure the correct evaluation of J2EE policies.

You can set these policies either in the application's deployment descriptors or, in cases where the application uses the J2EE programmatic security APIs, in the application code. URL policies that are defined in OpenSSO Enterprise do not take effect in this mode. If the application uses declarative security in the Web tier, you must configure the agent to enable that feature. See "Enabling Web-Tier Declarative Security in J2EE Agents" on page 65 for more information on how to enable this feature. While running in the J2EE_POLICY mode, the Policy Agent ensures that the security principal is set in the system for all authorized accesses.

J2EE Agent Filter Mode - URL_POLICY

In the URL_POLICY mode, the agent filter enforces the URL policies that are defined in OpenSSO Enterprise.

When the agent filter is in the URL_POLICY mode, the agent does not enforce any applicable J2EE declarative security policies. Such policies along with any calls to J2EE programmatic security API return negative results.

J2EE Agent Filter Mode - ALL

This is the most restrictive mode of the agent filter. In this mode, the filter enforces both J2EE policies and URL policies as defined in OpenSSO Enterprise. This mode of operation requires that the agent realm be configured in the deployment container. When running in the ALL mode, the agent ensures that the security principal is set in the system for every authorized access.

The ALL mode is highly recommended for deployed production systems.

Enabling Web-Tier Declarative Security in J2EE Agents

Certain applications might require the use of web-tier declarative security that enforces role-based access control over web resources such as Servlets, JSPs, HTML files and any other resource that can be represented as a URI. This type of security is enforced by adding `security-constraint` elements to the deployed application's `web.xml` deployment descriptor.

Typically `security-constraint` elements are tied with `auth-constraint` elements that identify the role membership that will be enforced when a request for a protected resource is made by the client browser. The following example illustrates this idea:

```
<security-constraint>
   <web-resource-collection>
     <web-resource-name>Report Servlet</web-resource-name>
     <url-pattern>/ReportGenServlet</url-pattern>
   </web-resource-collection>
   <auth-constraint>
     <role-name>MANAGER</role-name>
   </auth-constraint>
</security-constraint>
```

This fragment of deployment descriptor can be used to ensure that access to the report generation servlet is allowed only to those users who are members of the role called Manager.

In order for such a construct to work, you must make the necessary modifications to the applicable J2EE agent properties to ensure it can identify and handle such requests.

▼ To Enable J2EE Agents to Handle Security Constraint Settings

1 **Ensure that a `login-config` element is specified for the web application that is being protected and that the `login-config` element has the `auth-method` set to `FORM`.**

The supporting `form-login-config` element is also required.

2 **Add the `form-login-page` element of `form-login-config` as one of the values for the following agent property:**

`com.sun.identity.agents.config.login.form`

As an example, consider the following `login-config` element of a protected application:

```
<login-config>
   <auth-method>FORM</auth-method>
   <form-login-config>
      <form-login-page>/jsp/login.jsp</form-login-page>
      <form-error-page>/block.html</form-error-page>
   </form-login-config>
</login-config>
```

Notice how the `form-login-page` is specified for the supporting `form-login-config` element. This value must be set for the property labeled Login Form URI (Tab: Application, Name: `com.sun.identity.agents.config.login.form`). The following is a feasible value for this property:

`/Portal/jsp/login.jsp`

Notice that the value of the `form-login-page` as specified in the deployment descriptor is not the same as what is specified for the Login Form URI property. The difference being that when you enter this value in the configuration file, you must prefix it with the context path for the application on which this `form-login-page` is going to be used. In this particular example, the context path of the application is "`/Portal`."

Similarly, if you have more than one application deployed that require web-tier declarative security, you must add their respective form-login-pages to the property labeled Login Error URI (`com.sun.identity.agents.config.login.error.uri`). For example, The following are feasible values for the Login Error URI property:

`/BankApp/SignOn`

`/ERP/LoginServlet`

Ensure that each such element added to this list has a unique index entry. Having duplicate index entries can result in the loss of data and consequently result in the malfunction of the application.

Once you have configured the web application's deployment descriptor to use the form-login mechanism for web-tier declarative security and have added the full URI of the form-login-page for each such application to the applicable agent property, the web-tier declarative security is enabled for these applications.

Note –

- When a protected application is configured for web-tier declarative security handling by the agent, it must be redeployed with a form-login configuration as described in this section. This configuration requires that two application resources be specified in the application's `web.xml` deployment descriptor: one for the `form-login-page` and the other for the `form-error-page`. Regardless of whether the resource corresponding to the `form-login-page` exists in the application or not (this depends on how the agent is configured to handle the form-login requests), the resource corresponding to the `form-error-page` must be present in the application. This resource is directly invoked by the deployment container to indicate authentication failures and, optionally, authorization failures. If the application does not contain a valid `form-error-page` matching the URI specified in this deployment descriptor, it could result in HTTP 404 errors when the container chooses to display this error page.
- For applications that do not contain a `form-login-page`, you can specify any URI as long as that URI does not conflict with any application resource and the matching value has been added to the configuration property `com.sun.identity.agents.config.login.form`.
- By default, the agent is configured to intercept all form-login requests and handle them without invoking the actual `form-login-page` resource as specified in the `web.xml` of the protected application. Thus, when using a default installation of the agent, the application is not required to have a resource corresponding to the `form-login-page` element specified in `web.xml`. This allows for the configuration of web-tier declarative security for applications that were not designed to use the form-login mechanism and instead relied on other login schemes available in J2EE specification. This behavior of the agent can be changed so that it allows the form-login requests to be handled by actual resources that exist within the application by changing the agent configuration properties as applicable. For details on how this can be done, please refer to the section "Customizing Agent Response for Form Login" on page 68.
- If the agent filter is operating in the URL_POLICY mode, any necessary URL policies to allow access to the `form-error-page` resource must be created for all users.

To further customize the behavior of the application when using web-tier declarative security, see "Web-Tier Security Details" on page 67.

Web-Tier Security Details

When the deployment container gets a request for a resource that is protected by the web-tier declarative `security-constraint`, it must evaluate the credentials of the user against the agent realm to ensure that only authorized requests go through. In order to process such a request, the deployment container requires the user to sign on using the specified form login page as mentioned in the `form-login-config` element of the `web.xml` descriptor. Based on the specification of the `FORM` authentication mechanism, it is required that the user submits a valid user name as `j_username` and a valid password as `j_password` to the special URI `j_security_check` using the `HTTP POST` method of form submission.

The agent, once configured to support web-tier declarative security for the given application can isolate the request for accessing form-login-page and instead can stream out some data to the client browser. This data contains the user's login name and temporary encrypted password, which in turn uses Javascript to do automatic form submission as required. This gives the user a seamless single sign-on experience since the user does not have to re-login in order to access the protected resources for a deployed application that uses web-tier declarative security.

By default, the content that the agent sends to the client browser on intercepting a request for the form login page is read from the file called `FormLoginContent.txt` located in the `locale` directory of the agent installation. This file contains the following HTML code:

```
<html>
   <head>
      <title>Security Check</title>
   </head>
   <body onLoad="document.security_check_form.submit()">
      <form name="security_check_form" action="j_security_check" method="POST">
         <input type="hidden" value="am.filter.j_username" name="j_username">
         <input type="hidden" value="am.filter.j_password" name="j_password">
      </form>
   </body>
</html>
```

Before the agent streams out the contents of this file, it replaces all occurrences of the string `am.filter.j_username` by the appropriate user name. Similarly, all occurrences of the string `am.filter.j_password` are replaced by a temporary encrypted string that acts as a one-time password for the user.

Customizing Agent Response for Form Login

J2EE agent properties allow you to completely control the content that is sent out to the user when the deployment container requires a form login from the user.

Note – The ability to customize the agent response form login is not a feature whose purpose is to change the form login page nor is the purpose of this feature to bypass the default OpenSSO Enterprise login page.

Using the J2EE agent properties, you can customize the agent response in the following ways:

▼ To Customize the Agent Response to Form Login

1. **Modify the content of the `FormLoginContent.txt` file to suit your UI requirements as necessary.**

   Ensure that regardless of the modifications you make, the final file submits the `j_username` and `j_password` to the action `j_security_check` via `HTTP POST` method.

2. **(Conditional) You can specify the name of a different file by using the property labeled Login Content File Name (Tab: Application, Name: `com.sun.identity.agents.config.login.content.file`).**

   If you specify the file name, you must ensure that it exists within the `locale` directory of the agent installation.

   If you wish that this file be used from another directory, you can simply specify the full path to this new file.

   Ensure that regardless of the modifications you make, the final file submits the `j_username` and `j_password` to the action `j_security_check` via `HTTP POST` method.

3. **(Conditional) If you have more than one application and would like to have an application-specific response to the form login requests, instruct the agent to allow the form login request to proceed to the actual form login page.**

   This can be done by enabling the property labeled Use Internal Login (Tab: Application, Name: `com.sun.identity.agents.config.login.use.internal`).

   In this situation, you must ensure that the resource that receives this request extracts the `am.filter.j_username` and `am.filter.j_password` from the `HttpServletRequest` as attributes and uses that to ensure that eventually a submit of these values as `j_username` and `j_password` is done to the action `j_security_check` via `HTTP POST` method.

   The following JSP fragment demonstrates how this can be done:

```
<form action="j_security_check" method="POST">
<%
   String user = (String) request.getAttribute("am.filter.j_username");
   String password = (String) request.getAttribute("am.filter.j_password");
%>
   <ul>
      <li>Your username for login is: <b><%=user%></b></li>
      <li>Your password for login is: <b><%=password%></b></li>
   </ul>
   <input type=hidden name="j_username" value="<%=user%>">
   <input type=hidden name="j_password" value="<%=password%>">
   <input type="submit" name="submit" value="CONTINUE">
</form>
```

This mechanism would therefore allow you to have an application-specific form-login handling mechanism.

Enabling Failover in J2EE Agents

The agent allows basic failover capabilities. This helps you ensure that if the primary OpenSSO Enterprise instance for which the agent has been configured becomes unavailable, the agent will switch to the next OpenSSO Enterprise instance as specified by setting the appropriate agent property. This configuration can be achieved by implementing the following steps.

▼ To Enable Failover in J2EE Agents

1 **Provide a list of OpenSSO Enterprise authentication services URLs that may be used by the agent to authenticate users who do not have sufficient credentials to access the protected resources.**

To create the list configure the property labeled OpenSSO Login URL (Tab: OpenSSO Services, Name: `com.sun.identity.agents.config.login.url`).

You may specify more than one login URL to this property as follows:

primary-OSSO-server — Represents the URL of the primary OpenSSO Enterprise instance to which users are redirected for authentication.

failover-OSSO-server1 — Represents the URL of the OpenSSO Enterprise instance to which users are redirected for authentication if the primary OpenSSO Enterprise instance fails.

failover-OSSO-server2 — Represents the URL of the OpenSSO Enterprise instance to which users are redirected for authentication if the primary OpenSSO Enterprise instance fails and the first failover OpenSSO Enterprise instance fails.

If a URL list is provided to OpenSSO Login URL property, the agent first tries to establish a connection to the first server (*primary-OSSO-server*) specified in the URL list. If the agent is successful in establishing this connection, it redirects the user to the OpenSSO Enterprise instance for authentication.

2 **(Optional) Turn prioritization on for the failover lists by enabling the property labeled Login URL Prioritized (Tab: OpenSSO Services, Name:** `com.sun.identity.agents.config.login.url.prioritized`**).**

Note – Enabling this property turns prioritization on for the login URL list and the CDSSO URL list. The two cases shown in this step specifically mention the login URL list. However, this explanation of prioritization is exactly the same for the CDSSO URL list. The final step in this procedure describes how to create the CDSSO URL list in case such a scenario applies to your site's deployment.

The following cases describe the behavior of the agent in different situations: when you enable prioritization and when you do not enable prioritization for the login URL list.

Case 1: The Login URL Prioritized property is enabled.

Enabling this property means that priority is established for the login URL list described in Step 1. The list was created by configuring the property labeled Login URL.

Therefore, the first URL on the list has a higher priority than the second URL on the list, which has a higher priority than the third URL on the list, and so on. If the server (*primary-OSSO-server*) specified in this example as the first URL on the list is running, the agent sends all requests to this server only. However, if *primary-OSSO-server* fails, from that point on, subsequent requests are sent to the server (*failover-OSSO-server1*), which is second on the list. Furthermore, if at some point *primary-OSSO-server* comes back, then the subsequent requests from that point on are sent to *primary-OSSO-server*, since it takes priority over *failover-OSSO-server1*. This mechanism always fails back to the highest priority OpenSSO Enterprise instance among the OpenSSO Enterprise instances that are running at the point in time the agent must redirect requests to an OpenSSO Enterprise instance.

Case 2: The Login URL Prioritized property is not enabled.

In this case, no server takes priority over another. Failover occurs in a round-robin fashion. If all the servers are running, the agent sends requests to the server (*primary-OSSO-server*), which is first on the list. If *primary-OSSO-server* goes down then all subsequent requests are sent to the server (*failover-OSSO-server1*), which is second on the list. The agent keeps sending the requests to *failover-OSSO-server1* unless that server goes down. If *failover-OSSO-server1* does go down then the agent routes all the subsequent requests to the server (*failover-OSSO-server2*), which is third on the list, until it goes down. If it goes down, the agent tries to connect to *primary-OSSO-server* once again. Assuming that by then the *primary-OSSO-server* is running, all the subsequent requests from then on are sent to *primary-OSSO-server*. This is a simple round-robin mechanism without any priority involved.

3. **Provide a list of OpenSSO Enterprise Naming Service URLs that may be used by the agent to get access to the various other service URLs that may be needed to serve the logged on user.**

   This can be done by using the following property:

   `com.iplanet.am.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file)

   More than one naming service URL may be specified as a space delimited list of URLs. The following example illustrates this idea:

`com.iplanet.am.naming.url` = *primary-OSSO-server failover-OSSO-server1*

4 **(Conditional) If the deployment consists of an agent instance that is on a different domain than multiple OpenSSO Enterprise instances for which you want to enable failover, provide a URL list of the remote OpenSSO Enterprise instances.**

Configure the property labeled CDSSO Servlet URL (Tab: SSO, Name: `com.sun.identity.agents.config.cdsso.cdcservlet.url`) to create the list, specifying multiple CDSSO URLs as such:

primary-remoteOSSO-server

failover-remoteOSSO-server1

failover-remoteOSSO-server2

Login Attempt Limit in J2EE Agents

When a user tries to access a protected resource without having authenticated with OpenSSO Enterprise Authentication Services, the request is treated as a request with insufficient credentials. The default action taken by the agent when it encounters such a request is to redirect the user to the next available login URL as configured with the property labeled Login Attempts Limit (Tab: Global, Name: `com.sun.identity.agents.config.login.attempt.limit`).

Despite the repeated redirects performed by the agent, the user could still be unable to furnish the necessary credentials. In such a case, the agent can be directed to block such a request.

If a non-zero positive value is specified for this property, the agent will only allow that many attempts before it blocks the access request without the necessary credentials. When set to a value of zero, this feature is disabled.

To guard against potential denial-of-service attacks on your system, enable this feature.

Redirect Attempt Limit in J2EE Agents

The processing of requests by the agent can result in redirects for the client browser. Such redirects can happen when the user has not authenticated with OpenSSO Enterprise Authentication Service, lacks the sufficient credentials necessary to access a protected resource, and a variety of other reasons.

While the agent ensures that only the authenticated and authorized users get access to the protected resources, there is a remote possibility that due to misconfiguration of the system, the client browser may be put into an infinite redirection loop.

The property labeled Redirect Attempt Limit (Tab: Global, Name: `com.sun.identity.agents.config.redirect.attempt.limit`) allows you to guard against such potential situations by ensuring that after a given number of consecutive requests from a particular user that result in the same exact redirect, the agent blocks the user request. This blocking of the request is only temporary and is removed the moment the user makes a request that does not result in the same redirect or results in access being granted to the protected resource.

If a non-zero positive integer is specified as the value of this property, the agent will break the redirection loop after the specified number of requests result in the same redirects. When its value is set to zero, this feature is disabled.

To protect the system from such situations, enable this feature. Furthermore, enabling this feature can help in breaking potential denial of service attacks.

Not-Enforced URI List in J2EE Agents

Agents in the Policy Agent 3.0 software set allow you to specify a list of URIs that are treated as not-enforced. Access to these resources is always granted by the agent. The property labeled Not Enforced URIs (Tab: Application, Name: `com.sun.identity.agents.config.notenforced.uri`) controls the list.

If your deployed application has pages that use a bulk of graphics that do not need the agent protection, such content should probably be added to the agent's not-enforced list to ensure the optimal utilization of the system resources. Following is an OpenSSO Enterprise Console example of the entries that you can specify in the not-enforced list:

```
/images/*

/public/*.html

/registration/*
```

This enables the agent to focus on enforcing access control only over requests that do not match these given URI patterns. The use of a wildcard (*) is allowed to indicate the presence of one or more characters in the URI pattern being specified. For more information about the use of wildcards with OpenSSO Enterprise and Policy Agent, see Appendix C, "Wildcard Matching in Policy Agent 3.0 J2EE Agents."

Inverting the Not-Enforced URI List

In situations where only a small portion of the deployed application needs protection, you can configure the agent to do just that by inverting the not-enforced list. This results in the agent enforcing access control over the entries that are specified in the not-enforced list and allowing access to all other resources on the system. This feature is controlled by the property labeled Invert Not Enforced URIs (Tab: Application, Name: `com.sun.identity.agents.config.notenforced.uri.invert`).

When you enable this property, it changes the entries specified in the not-enforced list to enforced and the rest of the application resources are treated as not-enforced.

Caution – When the not-enforced list is inverted, the number of resources for which the agent will not enforce access control is potentially very large. The use of this feature should therefore be used with extreme caution and only after extensive evaluation of the security requirements of the deployed applications.

Note –

- When an access denied URI is specified, it is never enforced by the agent regardless of the configuration of the not-enforced list. Therefore, when a URI is listed as a value for the property labeled Resource Access Denied URI (Tab: Application, Name: `com.sun.identity.agents.config.access.denied.uri`), that resource is displayed when applicable by the agent whether or not that URI is listed as a value for the properties labeled Not Enforced URIs or Invert Not Enforced URIs. This behavior is necessary to ensure that the agent can use the access denied URI to block any unauthorized access for protected system resources.
- When configuring access denied URIs within the deployment descriptor of the web application, you must ensure that these values are added to the not-enforced list of the agent. Failing to do so can result in application resources becoming inaccessible by the user.
- Any resource that has been added to the not-enforced list must not access any protected resource. If it does so, it can result in unauthorized access to protected system resources. For example, if a servlet that has been added to the not-enforced list, in turn sends the request to another servlet, which is protected, it can potentially lead to unauthorized access to the protected servlet.

Fetching Attributes in J2EE Agents

Certain applications rely on the presence of user-specific profile information in some form in order to process the user requests appropriately. J2EE agents provide the functionality that can help such applications by making these attributes from the user's profile available in various forms. Policy Agent 3.0 allows the following attribute types to be fetched using the corresponding properties:

Profile Attributes
  Profile Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.profile.attribute.fetch.mode`)

Session Attributes
  Session Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.session.attribute.fetch.mode`)

Policy Response Attributes
  Response Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.response.attribute.fetch.mode`)

The following values are possible for these three properties:

- `NONE`
- `HTTP_HEADER`
- `REQUEST_ATTRIBUTE`
- `HTTP_COOKIE`

The default value for these properties is `NONE`, which specifies that that particular attribute type (profile attribute, session attribute, or policy response attribute) is not fetched. The other possible values (`HTTP_HEADER`, `REQUEST_ATTRIBUTE`, or `HTTP_COOKIE`) that can be used with these properties specify which method will be used to fetch a given attribute type. For more information, see "Methods for Fetching Attributes in J2EE Agents" on page 77.

Depending upon how these values are set, the agent retrieves the necessary attributes available for the logged on user and makes them available to the application.

The final subsection in this section describes other J2EE agent properties that can influence the attribute fetching process, see "Common Attribute Fetch Processing Related Properties" on page 79.

The following subsections provide information about how to set the type of attribute that is fetched.

Fetching Profile Attributes in J2EE Agents

To obtain user-specific information by fetching profile attributes, assign a mode to the profile attribute property and map the profile attributes to be populated under specific names for the currently authenticated user. The following example first demonstrates how to assign the `REQUEST_ATTRIBUTE` mode for fetching profile attributes and then demonstrates a way to map those attributes:

Example:

In OpenSSO Enterprise Console, Select the `REQUEST_ATTRIBUTE` mode option of the Profile Attribute Fetch Mode property (Tab: Application, Name: `com.sun.identity.agents.config.profile.attribute.fetch.mode`).

Then, map profile attributes using the property labeled Profile Attribute Mapping (Tab: Application, Name: `com.sun.identity.agents.config.profile.attribute.mapping`), such as illustrated in the following example:

Map Key	`cn`
Corresponding Map Value	`CUSTOM-Common-Name`
Map Key	`mail`
Corresponding Map Value	`CUSTOM-Email`

When you are done setting the Profile Attribute Mapping property as described in this example, it appears in OpenSSO Enterprise Console with the following format:

```
[cn]=CUSTOM-Common-Name
[mail]=CUSTOM-Email
```

Fetching Session Attributes in J2EE Agents

To obtain user-specific information by fetching profile attributes, assign a mode to the session attribute property and map the session attributes to be populated under specific names for the currently authenticated user. The following example first demonstrates how to assign the `REQUEST_ATTRIBUTE` mode for fetching session attributes and then demonstrates a way to map those attributes:

Example:

In OpenSSO Enterprise Console, Select the `REQUEST_ATTRIBUTE` mode option of the Session Attribute Fetch Mode property (Tab: Application, Name: `com.sun.identity.agents.config.session.attribute.fetch.mode`).

Then, map session attributes using the property labeled Session Attribute Mapping (Tab: Application, Name: `com.sun.identity.agents.config.session.attribute.mapping`), such as illustrated in the following example:

Map Key	`UserToken`
Corresponding Map Value	`CUSTOM-userid`

When you are done setting the Session Attribute Mapping property as described in this example, it appears in OpenSSO Enterprise Console with the following format:

```
[UserToken]=CUSTOM-userid
```

Fetching Policy Response Attributes in J2EE Agents

To obtain user-specific information by fetching policy response attributes, assign a mode to the policy response attribute property and map the policy response attributes to be populated under specific names for the currently authenticated user. The following example first demonstrates how to assign the `REQUEST_ATTRIBUTE` mode for fetching policy response attributes and then demonstrates a way to map those attributes:

Example:

In OpenSSO Enterprise Console, Select the `REQUEST_ATTRIBUTE` mode option of the Response Attribute Fetch Mode property (Tab: Application, Name: `com.sun.identity.agents.config.response.attribute.fetch.mode`).

Then, map response attributes using the property labeled Response Attribute Mapping (Tab: Application, Name: `com.sun.identity.agents.config.response.attribute.mapping`), such as illustrated in the following example:

Map Key	`cn`
Corresponding Map Value	`COMMON_NAME`
Map Key	`mail`
Corresponding Map Value	`CUSTOM-EMAIL_ADDR`

When you are done setting the Profile Attribute Mapping property as described in this example, it appears in OpenSSO Enterprise Console with the following format:

```
[cn]=COMMON_NAME
[mail]=CUSTOM-EMAIL_ADDR
```

With this property, you can specify any number of attributes that are required by the protected application. For the preceding example, the application requires the attributes `cn` and `mail` and searches for these attributes under the names `COMMON_NAME` and `EMAIL_ADDR`.

Methods for Fetching Attributes in J2EE Agents

The attribute types can be fetched by different methods as follows:

- HTTP Headers
- Request Attributes
- Cookies

Fetching Attributes as HTTP Headers

When the agent is configured to provide the LDAP attributes as HTTP headers, these attributes can be retrieved using the following methods on the `javax.servlet.http.HttpServletRequest` interface:

```
long getDateHeader(java.lang.String name)

java.lang.String getHeader(java.lang.String name)

java.util.Enumeration getHeaderNames()

java.util.Enumeration getHeaders(java.lang.String name)

int getIntHeader(java.lang.String name)
```

The property labeled Fetch Attribute Date Format (Tab: Application, Name: `com.sun.identity.agents.config.attribute.date.format`) controls the parsing of a date value from an appropriate string as set in the LDAP attribute.

This property defaults to the value `EEE, d MMM yyyy hh:mm:ss z` and should be changed as necessary.

Multi-valued attributes can be retrieved as an instance of `java.util.Enumeration` from the following method:

```
java.util.Enumeration getHeaders(java.lang.String name)
```

Fetching Attributes as Request Attributes

When the agent is configured to provide the LDAP attributes as request attributes, the agent populates these attribute values into `HttpServletRequest` as attributes that can later be used by the application as necessary. These attributes are populated as `java.util.Set` objects, which must be cast to this type before they can be successfully used.

Fetching Attributes as Cookies

When the agent is configured to provide the LDAP attributes as cookies, the necessary values are set as server specific cookies by the agent with the path specified as "/."

Multi-valued attributes are set as a single cookie value in a manner that all values of the attribute are concatenated into a single string using a separator character that can be specified by the property labeled Cookie Separator Character property (Tab: Application, Name: `com.sun.identity.agents.config.attribute.cookie.separator`).

One of the tasks of the application is to parse this value back into the individual values to ensure the correct interpretation of the multi-valued LDAP attributes for the logged on user.

When you are fetching attributes as cookies, also use the cookie reset functionality to ensure that these cookies get cleaned up from the client browser when the client browser's session expires. For more information, see "Using Cookie Reset Functionality in J2EE Agents" on page 81.

Common Attribute Fetch Processing Related Properties

This section lists the most common configuration properties that are used to influence attribute fetching.

Cookie Separator Character property (Tab: Application, Name: `com.sun.identity.agents.config.attribute.cookie.separator`)
: This property allows you to assign a character to be used to separate multiple values of the same attribute when it is being set as a cookie. The value that you assign to this property is the character, for example the pipe symbol "|", that will separate multiple values of the same attribute when it is being set as a cookie.

Attribute Cookie Encode property (Tab: Application, Name: `com.sun.identity.agents.config.attribute.cookie.encode`)
: This property is a flag (enabled or not enabled) that indicates if the value of the attribute should be URL encoded before being set as a cookie.

Fetch Attribute Date Format (Tab: Application, Name: `com.sun.identity.agents.config.attribute.date.format`)
: This property allows you to set the format of date attribute values to be used when the attribute is set to HTTP header. This format is based on the definition as provided in `java.text.SimpleDateFormat`. The format for the value of this property is as follows:

```
EEE, d MMM yyyy hh:mm:ss z
```

Configuring FQDN Handling in J2EE Agents

To ensure appropriate user experience, the use of valid URLs by users to access resources protected by the agent must be enforced. This functionality is controlled by three separate properties:

FQDN Check (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.check.enable`)
: Enables FQDN Check

FQDN Default (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.default`)
: Stores the default FQDN value

FQDN Virtual Host Map (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.mapping`)
: Sets FQDN mapping

The property labeled FQDN Default provides the necessary information needed by the agent to identify if the user is using a valid URL to access the protected resource. If the agent determines that the incoming request does not have a valid hostname in the URL, it redirects the user to the corresponding URL with a valid hostname. The difference between the redirect URL and the URL originally used by the user is only the hostname, which is now changed by the agent to a fully qualified domain name (FQDN) as per the value specified in this property.

The property labeled FQDN Virtual Host Map provides another way by which the agent can resolve malformed access URLs used by users and take corrective action. The agent gives precedence to entries defined in this property over the value defined in the FQDN Default property. If none of the entries for this property matches the hostname specified in the user request, the agent uses the value specified for FQDN Default property to take the necessary corrective action.

The FQDN Virtual Host Map property can be used for creating a mapping for more than one hostname. This can be done when the deployment container protected by this agent can be accessed using more than one hostname. As an example, consider a protected deployment container that can be accessed using the following host names:

- `www.externalhostname.com`
- `internalhostname.interndomain.com`
- *IP address*

In this case, assuming that `www.externalhostname.com` is the value assigned to the FQDN Default property, then the FQDN Virtual Host Map property can be configured using OpenSSO Enterprise Console as follows to allow access to the application for users who will use the hostname `internalhostname.interndomain.com` or the raw IP address, 192.101.98.45:

Map Key	`internalhostname.interndomain.com`
Corresponding Map Value	`internalhostname.interndomain.com`
Map Key	`192.101.98.45`
Corresponding Map Value	`192.101.98.45`

When you are done setting the FQDN Virtual Host Map property as described in this example, it appears in OpenSSO Enterprise Console with the following format:

```
[internalhostname.interndomain.com] = internalhostname.interndomain.com

[192.101.98.45] = 192.101.98.45
```

Using Cookie Reset Functionality in J2EE Agents

The agent allows you to reset certain cookies that might be present in the user's browser session if the user's OpenSSO Enterprise session has expired. This feature is controlled by the following configuration properties:

- Cookie Reset (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset.enable`)
- Cookies Reset Name List (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset.name`
- Cookies Reset Domain Map (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset.domain`)
- Cookies Reset Path Map (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset.path`)

The preceding four properties can be used to specify the exact details of the cookie that should be reset by the agent when a protected resource is accessed without a valid session.

The property labeled Cookies Reset Name List specifies a list of cookie names that will be reset by the agent when necessary. Each entry in this list can correspond to a maximum of one entry in the properties labeled Cookies Reset Domain Map and Cookies Reset Path Map, both of which are used to define the cookie attributes - the domain on which a particular cookie should be set and the path on which it will be set.

When using this feature, ensure that the correct values of the domain and path are specified for every cookie entry in the cookie list. If these values are inappropriate, the result might be that the cookie is not reset in the client browser.

When a cookie entry does not have an associated domain specified in the domain map, it is handled as a server cookie. Similarly, when a cookie entry does not have a corresponding path entry specified, the anticipated cookie path is "`/`."

Enabling Port Check Functionality in J2EE Agents

In situations when OpenSSO Enterprise and the deployment container are installed on the same system but on different ports, certain browsers may not send the HOST header correctly to the agent in situations where there are redirects involved between OpenSSO Enterprise Authentication Service and the agent. In such situations, the agent, relying on the availability of the port number from the deployment container, might misread the port number that the user is trying to access.

When such a situation occurs, it can have a severe impact on the system since the agent now senses a resource access that in reality did not occur and consequently the subsequent redirects as well as any policy evaluations may fail thereby making the protected application inaccessible to the end user.

This situation can be controlled by enabling port check functionality on the agent. This is controlled by the property labeled Port Check Enable (Tab: Miscellaneous, Name: `com.sun.identity.agents.config.port.check.enable`).

When this property is enabled, the agent verifies the correctness of the port number read from the request against its configuration. The configuration that provides the reference for this checking is set by the property labeled Port Check Setting (Tab: Miscellaneous, Name: `com.sun.identity.agents.config.port.check.setting`).

This property allows the agent to store a map of various ports and their corresponding protocols. When the agent is installed, this map is populated by the preferred port and protocol of the agent server as specified during the installation. However, if the same agent is protecting more than one HTTP listeners, you must add that information to the map accordingly.

When the agent discovers an invalid port in the request, it takes corrective action by sending some HTML data to break the redirection chain so that the browser can reset its HOST header on the subsequent request. This content is read from the file that resides in the `locale` directory of agent installation. The name of the file is controlled by the property labeled Port Check File (Tab: Miscellaneous, Name: `com.sun.identity.agents.config.port.check.file`).

This property can also be used to specify the complete path to the file that may be used to achieve this functionality. This file contains special HTML that uses a `META-EQUIV REFRESH` tag in order to allow the browser to continue automatically when the redirect chain is broken. Along with this HTML, this file must contain the string `am.filter.request.url`, which is dynamically replaced by the actual request URL by the agent.

You can modify the contents of this file or specify a different file to be used, if necessary, so long as it contains the `am.filter.request.url` string that the agent can substitute in order to construct the true request URL with the correct port. The contents of this file should be such that it should either allow the user to automatically be sent to this corrected location or let the user click a link or a button to achieve the same result.

Web Services Security Support for J2EE Agents in Policy Agent 3.0

A web service is a software system identified by a URI, whose public interfaces and bindings are defined and described using XML. The definition for a given web service can be discovered by other software systems. These systems can then interact with the web service in a manner prescribed by its definition, using XML-based messages conveyed by Internet protocols.

A web service consists of two main components, a web service client (WSC) and a web service provider (WSP). A web service client is a software component that acts as a requester for a certain service using web service interface. A web service provider is a software component that provides the service.

J2EE agents in the Policy Agent 3.0 support Web Services Security for web service providers. A web service provider deployed on an application server protected by a J2EE agent instance can have additional security provided by the agent. Together, J2EE agent software and OpenSSO Enterprise server support various Web Services Security profiles including UsernameToken, X509Token, and SAML2 Token.

The task that follows describes how to configure Web Services Security support in J2EE Agents for Policy Agent 3.0.

▼ To Configure Web Services Security Support in J2EE Agents for Policy Agent 3.0

Before You Begin The instructions that follow start with the assumption that OpenSSO Enterprise server and a J2EE agent instance have been properly installed and configured.

1. **Create a web service provider configuration for the web service provider endpoint protected by the J2EE Agent instance as follows:**

   a. **Log in to OpenSSO Enterprise Console as a user with AgentAdmin privileges, such as `amadmin`.**

      The OpenSSO Enterprise login page is available at a URL similar in format to the following:

      `http://OpenssoHost.example.com:58080/opensso`

   b. **Click the Access Control tab.**

   c. **Click the name of the appropriate realm, such as the following: /(Top Level Realm).**

   d. **Click the Agents tab.**

   e. **Click the Web Service Provider tab.**

   f. **Click the New button in the Agent section.**

g. Enter the full URL of the web service provider endpoint as the name of the Web Service Provider agent.

For example, `http://myhost.mydomain.com:8080/StockService/StockService`.

h. Enter a password.

i. Click the Create button.

2 Create an agent authenticator type agent profile as follows.

This agent and its password will be used for the agent to authenticate to the OpenSSO Enterprise server.

Fore more information about the agent authenticator, see "About the Agent Authenticator in Policy Agent 3.0" on page 50.

a. Log in to OpenSSO Enterprise Console as a user with AgentAdmin privileges, such as `amadmin`.

b. Click the Access Control tab.

c. Click the name of the appropriate realm, such as the following: /(Top Level Realm).

d. Click the Agents tab.

e. Click on Agent Authenticator tab.

f. Click the New button.

g. Enter an agent authenticator name and password.

h. Click the Create button.

i. On the Agent Authenticator page, click the link for the newly created agent authenticator.

The agent authenticator page is displayed. In the section labeled "Agent Profiles allowed to Read," two lists exist: Available and Selected. The Available list has all the available agents in the system, and the Selected list has all the agents whose configurations can be read by this agent authenticator.

j. From the available list, select the following items:

- The J2EE agent profile name (this is the profile name of a previously installed J2EE agent instance)
- The web service provider agents protected by the J2EE agent

- The item labeled SecurityTokenService

k. Click Add to move the selected items from the Available list to the Selected list

l. Click Save

3 Stop the agent container.

4 Edit the `OpenSSOAgentBootstrap.properties` file of the previously installed J2EE agent, as follow:

The bootstrap file is located at the following location:

PolicyAgent-base/*AgentInstance-Dir*/`config`

For information about this location, see Table P–6

a. Using your text editor of choice, open the `OpenSSOAgentBootstrap.properties` file.

b. At the end of the bootstrap file, add five lines as follows:

i. Add the following line:

```
com.sun.identity.wss.trustclient.enablemetro=false
```

ii. Add the four following lines related to keystore information.

The various place holders for these four lines must be replaced with the actual keystore information.

`com.sun.identity.saml.xmlsig.keystore=`*KeystorePathName*
`com.sun.identity.saml.xmlsig.storepass=`*FileContainingEncryptedKeystorePassword*
`com.sun.identity.saml.xmlsig.keypass=`*FileContainingEncryptedKeyPassword*
`com.sun.identity.saml.xmlsig.certalias=`*KeyAliasName*

c. Change the value for the property named `com.sun.identity.agents.app.username` to the agent authenticator name.

Therefore, the setting would be as such:

`com.sun.identity.agents.app.username =` *AgentAuthenticatorName*

where *AgentAuthenticatorName* represents the name provided for the agent authenticator as demonstrated previously in this task.

d. Change the value for the property named `com.iplanet.am.service.secret` to the agent authenticator password.

Therefore, the setting would be as such:

`com.iplanet.am.service.secret =` *EncryptedAgentAuthenticatorPassword* where *EncryptedAgentAuthenticatorPassword* represents the encrypted version of the password provided for the agent authenticator as demonstrated previously in this task.

Note – To encrypt the password, use the `agentadmin --encrypt` command as described in "`agentadmin --encrypt`" on page 42.

---

e. **Save and close the bootstrap file.**

5 Configure the J2EE agent to enable Web Services Security as follows:

a. **Using a browser, navigate through OpenSSO Enterprise Console to the agent properties of the J2EE agent that you want to configure.**

For the steps to navigate to the J2EE agent properties, see "To Navigate in OpenSSO Enterprise 8.0 Console to the J2EE Agent Properties" on page 53.

b. **Enable the property labeled Web Service Enable (Tab: Advanced, Name:** `com.sun.identity.agents.config.webservice.enable`**)**

c. **Add the web service endpoint URIs as values for the property labeled Web Service End Points (Tab: Advanced, Name:** `com.sun.identity.agents.config.webservice.endpoint`**) as follows:**

i. **Add a value, such as** `/StockService/StockService`**, in the New Value field and.**

ii. **Click Add.**

d. **Add the web service endpoint related WSDL to the not enforced URI list (Tab: Advanced, Name:** `com.sun.identity.agents.config.notenforced.uri`**) as follows:**

`com.sun.identity.agents.config.notenforced.uri[`*n*`]=`*your-wsp-endpointuri*`?wsdl*`

For example:

```
com.sun.identity.agents.config.notenforced.uri[0]=/StockService/StockService?wsdl*
```

e. **Click Save on the Advanced page.**

6 Change the Security Token Service as follows:

a. **As a user with AgentAdmin privileges, such as** `amadmin`**, use a browser to log in to the OpenSSO Enterprise Console.**

b. **Click the Configuration tab.**

c. **Click the Global tab.**

d. **Click Security Token Service in the Global Properties list.**

e. **Add a new value to the property labeled For the Trusted Issuers, in the Token Validation Attributes section as follows:**

   i. **For the Trusted Issuers property, enter the fully qualified domain name of the OpenSSO Enterprise server in the New Value field.**

   ii. **Click Add.**

   iii. **Click Save on the Security Token Service page.**

7 **Set up a Web Services Security profile by following the substep alternative that fits your site's requirements.**

The J2EE agents in the Policy Agent 3.0 software set together with OpenSSO Enterprise server support various Web Services Security profiles including UsernameToken, X509Token, and SAML2 Token. The following alternative steps describe how to set up the four different profiles. Implement the substeps for the appropriate profile for your site's deployment.

- **Set up the UsernameToken-Plain profile as follow:**

  a. **As a user with AgentAdmin privileges, such as `amadmin`, use a browser to log in to the OpenSSO Enterprise Console.**

  b. **Click the Access Control tab.**

  c. **Click the name of the appropriate realm, such as the following: /(Top Level Realm).**

  d. **Click the Agents tab.**

  e. **Click the Web Service Provider sub tab.**

  f. **Click the web service provider.**

  For example, `http://myhost.mydomain.com:8080/StockService/StockService`.

  g. **Select the checkbox next to UserNameToken-Plain to choose it as a Security Mechanism.**

  h. **In the Authentication Chain drop-down list, select ldapService.**

  i. **In the "Web Service End Point" field, enter the name of the web service provider.**

  For example, `http://myhost:mydomain.com:8080/StockService/StockService`.

j. **(Optional) If you would like, you can choose to enable request signature verification, request decryption, response signing, and response encryption.**

k. **Click Save on that page.**

l. **Click "Back to Main Page."**

m. **Click the Web Service Client subtab.**

n. **Click the corresponding web service client profile.**

   The phrase "corresponding web service client profile" in this case refers to a client profile that you create. You can edit the default profile "wsc" or create a new web service client.

o. **Select UserNameToken-Plain as the Security Mechanism.**

p. **(Optional) If applicable, you can choose to enable request signing, request encryption, response signature verification, and response decryption.**

Note – The web service client and web service provider signing and encryption should be in sync. For example, if in the web service client, the request signing is enabled, then in the web service provider the request signature verification should be enabled as well.

---

q. **Click Save on that page.**

- **Set up the X509Token profile as described in the substeps that follow.**

a. **As a user with AgentAdmin privileges, such as `amadmin`, use a browser to log in to the OpenSSO Enterprise Console.**

b. **Create an authentication module, such as `certmod`, assigning the following type: Certificate.**

   See the *Sun OpenSSO Enterprise 8.0 Administration Guide* for more information about creating authentication modules and authentication chains.

c. **Create an authentication chain, such as `certauth`, that includes the module, such as `certmod`, that you just created in the previous substep.**

d. **Navigate to the web service provider.**

   For example, `http://myhost.mydomain.com:8080/StockService/StockService`.

e. **Select the checkbox next to X509Token to choose it as a Security Mechanism.**

f. **In the Authentication Chain drop-down list, select the newly created authentication chain, such as** `certauth`**.**

g. **In the "Web Service End Point" field, enter the name of web service provider.**

   For example, `http://myhost:mydomain.com:8080/StockService/StockService`.

h. **(Optional) If you would like, you can choose to enable request signature verification, request decryption, response signing, and response encryption.**

i. **Click Save on that page.**

j. **Click "Back to Main Page."**

k. **Click the Web Service Client subtab.**

l. **Select the corresponding web service client profile.**

   The phrase "corresponding web service client profile" in this case refers to a client profile that you create. You can edit the default profile "wsc" or create a new web service client.

m. **Select X509Token as the Security Mechanism.**

n. **(Optional) If applicable, you can choose to enable request signing, request encryption, response signature verification, and response decryption.**

Note – The web service client and web service provider signing and encryption should be in sync. For example, if in the web service client, the request signing is enabled, then in the web service provider the request signature verification should be enabled as well.

---

o. **Click Save on that page.**

- **Set up the SAML2-HolderOfKey profile as described in the substeps that follow.**

a. **As a user with AgentAdmin privileges, such as** `amadmin`**, use a browser to log in to the OpenSSO Enterprise Console.**

b. **Navigate to the web service provider.**

   For example, `http://myhost.mydomain.com:8080/StockService/StockService`.

c. **Select the checkbox next to SAML2-HolderOfKey to choose it as a Security Mechanism.**

d. **In the Authentication Chain drop-down list, select ldapService.**

e. **If not already entered, enter `urn:sun.wss.ssotoken` in the field for Token Conversion Type.**

f. **In the "Web Service End Point" field, enter the name of the web service provider.**

For example, `http://myhost:mydomain.com:8080/StockService/StockService`.

g. **(Optional) If you would like, you can choose to enable request signature verification, request decryption, response signing, and response encryption.**

h. **Click Save on that page.**

i. **Click "Back to Main Page."**

j. **Click the Web Service Client subtab.**

k. **Select the corresponding web service client profile.**

The phrase "corresponding web service client profile" in this case refers to a client profile that you create. You can edit the default profile "wsc" or create a new web service client.

l. **Select SAML2-HolderOfKey as the Security Mechanism.**

m. **(Optional) If applicable, you can choose to enable request signing, request encryption, response signature verification, and response decryption.**

Note – The web service client and web service provider signing and encryption should be in sync. For example, if in the web service client, the request signing is enabled, then in the web service provider the request signature verification should be enabled as well.

---

n. **Click Save on that page.**

- **Set up the SAML2-SenderVouches profile as described in the substeps that follow.**

a. **As a user with AgentAdmin privileges, such as `amadmin`, use a browser to log in to the OpenSSO Enterprise Console.**

b. **Navigate to the web service provider.**

For example, `http://myhost.mydomain.com:8080/StockService/StockService`.

c. **Select the checkbox next to SAML2-SenderVouches to choose it as a Security Mechanism.**

d. **In the Authentication Chain drop-down list, select ldapService.**

e. **If not already entered, enter** `urn:sun.wss.ssotoken` **in the field for Token Conversion Type.**

f. **In the "Web Service End Point" field, enter the name of the web service provider.**

For example, `http://myhost:mydomain.com:8080/StockService/StockService`.

g. **(Optional) If you would like, you can choose to enable request signature verification, request decryption, response signing, and response encryption.**

h. **Click Save on that page.**

i. **Click "Back to Main Page."**

j. **Click the Web Service Client subtab.**

k. **Select the corresponding web service client profile.**

The phrase "corresponding web service client profile" in this case refers to a client profile that you create. You can edit the default profile "wsc" or create a new web service client.

l. **Select SAML2-SenderVouches as the Security Mechanism.**

m. **(Optional) If applicable, you can choose to enable request signing, request encryption, response signature verification, and response decryption.**

Note – The web service client and web service provider signing and encryption should be in sync. For example, if in the web service client, the request signing is enabled, then in the web service provider the request signature verification should be enabled as well.

---

n. **Click Save on that page.**

8 **Enable Web Services Security in the web service provider as described in the substeps that follow.**

a. **Using your text editor of choice, open the** `web.xml` **file of the web service provider.**

b. **Add the agent filter element as the first filter in the** `web.xml` **file.**

For example, add the following:

```
<filter>
    <filter-name>Agent</filter-name>
    <filter-class> com.sun.identity.agents.filter.AmAgentFilter </filter-class>
</filter>
```

c. **Add the agent filter mapping element as the first filter mapping in the `web.xml` file.**

For example, add the following:

```
<filter-mapping>
    <filter-name>Agent</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

d. **Save and close the `web.xml` file.**

9 **Recreate the web service provider `.war` file.**

10 **Redeploy the web service provider on the application server.**

Resetting the J2EE Agent Profile Password in Policy Agent 3.0

Agents in the Policy Agent software set must authenticate with the OpenSSO Enterprise server in order for the two components to interact. To authenticate, the agent must provide its name (the agent profile name) and agent profile password. This password was established and encrypted as part of the J2EE agent installation process. For more information, see "Creating a J2EE Agent Profile in Policy Agent 3.0 Using OpenSSO Enterprise Console" on page 47. However, you can change this password if you choose.

The agent profile password can be reset with a combination of configuration steps involving both the OpenSSO Enterprise Console and the `OpenSSOAgentBootstrap.properties` file. The agent profile password should originally be created prior to agent installation. However, after you install a J2EE agent, you can update the agent profile password at anytime.

▼ To Update a J2EE Agent Profile Password in Policy Agent 3.0

The instructions that follow describe how to change agent profile password.

1 **Using a browser, navigate through OpenSSO Enterprise Console to the J2EE agent properties of the agent that you want to configure.**

For the steps to navigate to the J2EE agent properties, see "To Navigate in OpenSSO Enterprise 8.0 Console to the J2EE Agent Properties" on page 53.

2 Update the agent profile password in the J2EE agent properties section as described in the substeps that follow:

a. In the Global tab (which is the default tab), locate the property labeled Password.

b. Update the Password property to a password of your choice.

c. Update the property labeled "Password (confirm)" to the same value you chose for the Password property.

d. Click Save at the top of that Global page.

3 Update or create an agent profile password in a password file as described in the substeps that follow.

The password file should originally have been created as a J2EE agent pre-installation task.

a. (Conditional) If an ASCII text agent password file does not already exist, create one .

For example, create a file such as the following: `/tmp/pwf1`

b. Using your text editor of choice, enter in clear text on the first line, (or replace the original password, if one already exists with) the password you just updated in OpenSSO Enterprise Console.

c. Secure the password file appropriately, depending on the requirements for your deployment.

4 In the command line, issue the `agentadmin --encrypt` command to encrypt the new password.

For example:

PolicyAgent-base`/bin/agentadmin --encrypt Agent_001 /tmp/pwf1`

The `agentadmin` program returns the new encrypted password with a message such as the following:

```
The encrypted value is: AQICFtkDruE1iBJrZvPW2Yfpgitm/3NjmpIQ
```

For more information on this command, see "`agentadmin --encrypt`" on page 42.

5 Copy the encrypted value that is returned.

6 Using your text editor of choice, access the J2EE agent `OpenSSOAgentBootstrap.properties` configuration file at the following location:

PolicyAgent-base/*AgentInstance-Dir*/`config`

7 In the bootstrap configuration file, edit the property for the agent password as follows:

`com.iplanet.am.service.secret =` *encryptedPassword*

where *encryptedPassword* represents the new encrypted password you created when you issued the `agentadmin --encrypt` command.

This property is set in a manner similar to the following:

```
com.iplanet.am.service.secret=AQICFtkDruE1iBJrZvPW2Yfpgitm/3NjmpIQ
```

8 Restart the J2EE agent container.

The container must be restarted for the changes to the bootstrap file to take effect.

Key Features and Tasks Performed With the J2EE `agentadmin` Program

The `agentadmin` program is a utility used to perform a variety of tasks from required tasks, such as installation to optional tasks, such as displaying version information. This section summarizes the tasks that can be performed with the `agentadmin` program. Many of the tasks performed with this program are related to installation or uninstallation. For detailed information about the options available with this program, see "Role of the `agentadmin` Program in Policy Agent 3.0" on page 35.

In this section, the options are listed for your quick review to help you get a sense of how the `agentadmin` program fits in with the other methods of managing J2EE agents, which are all discussed in this chapter.

The location of the `agentadmin` program is as follows:

PolicyAgent-base`/bin`

For information about the location of *PolicyAgent-base*, see Table P–5.

The following table lists options that can be used with the `agentadmin` command and gives a brief description of the specific task performed with each option.

TABLE 4–4 The `agentadmin` Program: Supported Options

Option	Task Performed
`--install`	Installs a new agent instance
`--custom-install`	Installs a new agent instance
`--uninstall`	Uninstalls an existing Agent instance

TABLE 4–4 The `agentadmin` Program: Supported Options *(Continued)*

Option	Task Performed
`--listAgents`	Displays details of all the configured agents
`--agentInfo`	Displays details of the agent corresponding to the specified agent IDs
`--version`	Displays the version information
`--encrypt`	Encrypts a given string
`--getEncryptKey`	Generates an Agent Encryption key
`--uninstallAll`	Uninstalls all agent instances
`--migrate`	Migrates agent to a newer version
`--usage`	Displays the usage message
`--help`	Displays a brief help message

Key Features and Tasks Performed With the J2EE Agent API

The agent runtime provides access to all the OpenSSO Enterprise application program interfaces (API) that can be used to further enhance the security of your application. Besides the OpenSSO Enterprise API, the agent also provides a set of API that allow the application to find the SSO token string associated with the logged-in user. These API can be used from within the web container or the EJB container of the deployment container. These are agent utility API. However, an equally viable option is to use client SDK public API directly to fetch the SSO token.

Note – Certain containers, such as Apache Tomcat Servlet/JSP Container do not have an EJB container. Hence, the EJB related agent API would not be applicable for such containers.

The subsections that follow illustrate the available agent API that can be used from within an application. The J2EE agent API have changed in Policy Agent 3.0 as explained in this section. This section includes an example of the new API in use, see "Usage of New J2EE Agent API in Policy Agent 3.0" on page 97.

Class `AmFilterManager`

`com.sun.identity.agents.filter.AmFilterManager`

Available API for Class `AmFilterManager`

- `public static com.sun.identity.agents.filter.AmSSOCache getAmSSOCacheInstance() throws com.sun.identity.agents.arch.AgentException`

  Note – Deprecated: This method has been deprecated. The best practice is not to use this method, but to use the new public API for this `AmFilterManager` class as follows:

  ```
  public static com.sun.identity.agents.filter.IAmSSOCache getAmSSOCache()
  ```

  This method returns an instance of `Class AmSSOCache`, which can be used to retrieve the SSO token for the logged-in user. This method can throw `AgentException` if an error occurs while processing this request.

- `public static com.sun.identity.agents.filter.IAmSSOCache getAmSSOCache()`

  This method returns an instance of `IAmSSOCache` interface, which can be used to retrieve the SSO token for the logged-in user.

Interface `IAmSSOCache`

`com.sun.identity.agents.filter.IAmSSOCache`

Available API for Interface `IAmSSOCache`

`public String getSSOTokenForUser(Object ejbContextOrServletRequest)`

This method can be used to retrieve the SSO token for the logged-in user. If called from the web tier, this method passes an instance of `javax.servlet.http.HttpServletRequest` as an argument. If called from the EJB tier, this method passes an instance of `javax.ejb.EJBContext` as an argument. This method eradicates the need to use two separate methods in `AmSSOCache` to retrieve the SSO token.

Class `AmSSOCache`

`com.sun.identity.agents.filter.AmSSOCache`

Note – Deprecated: This class and its methods have been deprecated. The best practice is not to use the methods in this class, but to use the unified API in `com.sun.identity.agents.filter.IAmSSOCache`.

Available API for Class `AmSSOCache`

- `public java.lang.String getSSOTokenForUser(javax.servlet.http.HttpServletRequest request)`

  Note – Deprecated: This method has been deprecated as explained in the Note in "Class AmSSOCache" on page 96.

  This method returns the SSO token for the logged-in user whose request is currently being processed in the web container within the deployment container. This method can return null if the requested token is not available at the time of this call.

- `public java.lang.String getSSOTokenForUser(javax.ejb.EJBContext context)`

  Note – Deprecated: This method has been deprecated as explained in the Note in "Class AmSSOCache" on page 96.

  This method returns the SSO token for the logged on user whose request is currently being processed in the deployment container's EJB tier. This method can return null if the requested token is not available at the time of this call.

Note – The API `getSSOTokenForUser(javax.ejb.EJBContext)` can be used only when the agent operation mode is either J2EE_POLICY or ALL.

Usage of New J2EE Agent API in Policy Agent 3.0

The following example demonstrates the new J2EE agent API in use.

EXAMPLE 4–5 Usage of New J2EE Agent API

- Web Tier Use Case:

  ```
  String ssotoken =
  AmFilterManager.getAmSSOCache().getSSOTokenForUser(HTTPRequest);
  ```

- EJB Tier Use Case:

```
String ssotoken =
AmFilterManager.getAmSSOCache().getSSOTokenForUser(EJBContext);
```

Caution – This public API can only retrieve the `SSOToken` object in EJB context if the value of the following property labeled User Principal Flag (`com.sun.identity.agents.config.user.principal`) is enabled

APPENDIX A

Comparing Web Agents and J2EE Agents in Policy Agent 3.0

OpenSSO Enterprise Policy Agent 3.0 software consists of web agents and J2EE agents. This chapter explains the similarities and differences of these two types of agents.

An Overview of Policy Agent 3.0

Access control in Sun OpenSSO Enterprise is enforced using agents. Agents protect content on designated deployment containers, such as web servers and application servers, from unauthorized intrusions. Agents are separate from OpenSSO Enterprise.

Note – The most current agents in the Policy Agent software set can be downloaded from the Identity Management page of the Sun Microsystems Download Center: `http://www.sun.com/software/download`

Web agents and J2EE agents differ in a few ways. One significant way the two agent types differ is in the resources that the two agent types protect. Web agents protect resources on web and proxy servers while J2EE agents protect resources on application and portal servers. However, the most basic tasks that the two agent types perform in order to protect resources are similar.

This chapter does the following:

- Explains what agents do.
- Describes briefly what a web agent is.
- Describes briefly what a J2EE agent is.
- Explains how these two types of agents are similar to each other and yet different.

Agents do the following:

- Determine whether a user is authenticated.
- Determine whether a resource is protected.

- For an authenticated user attempting to access a protected resource, determine whether the user is authorized to access that resource.
- Allow or deny a user access to a protected resource according to the results of the authentication and authorization processes.
- Log access information and diagnostic information.
- Support cross-domain single sign-on (CDSSO). CDSSO applies to most agents.

The preceding task descriptions provide a simplified explanation of what agents do. Agents perform these tasks in conjunction with OpenSSO Enterprise. More specifically, agents work with various OpenSSO Enterprise services, such as Authentication Service, Session Service, Logging Service, and Policy Service to perform these tasks. Both agent types, J2EE agents and web agents interact with these OpenSSO Enterprise services in a similar manner.

Interaction Between Policy Agent 3.0 and OpenSSO Enterprise Services

This section shows how J2EE agents and web agents interact with OpenSSO Enterprise Services.

The figure that follows applies to web agents and J2EE agents. At this level, the agent types are the same in what they accomplish, even though at a deeper level the methods used vary to some degree.

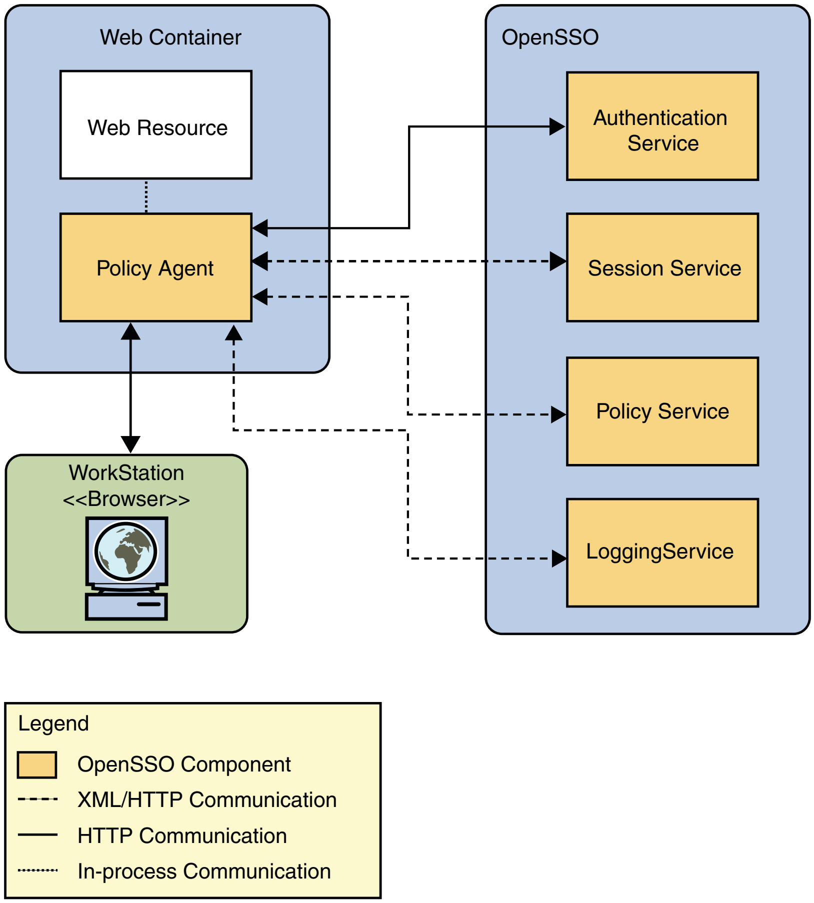

FIGURE A–1 Policy Agent Interaction with OpenSSO Enterprise Services

A key point is that the agent must interact continuously with various OpenSSO Enterprise services. The interactions that take place between Policy Agent and OpenSSO Enterprise are not covered in detail in Policy Agent documentation. For a more information on such interactions, see *Sun OpenSSO Enterprise 8.0 Administration Guide*.

A Generalized Example of the Policy Decision Process

When a user attempts to access content on a protected resource, many deployment variables are involved. For example, firewalls might be present or load balancers might be involved. From a high level, the two agent types (J2EE agents and web agents) handle these deployments in the same manner. The following reference presents figures that demonstrate how web agents and

J2EE agents have the same role in a complex OpenSSO Enterprise deployments: Part I, "About This Deployment," in *Deployment Example: Single Sign-On, Load Balancing and Failover Using Sun OpenSSO Enterprise 8.0*.

Therefore, J2EE agents and web agents do not differ in the general role they play in an OpenSSO Enterprise deployment. However, at the policy decision level, the differences between the two agent types are more easily observed, and components such as firewalls and load balancers can add complexity to the policy decision process. Therefore, for the basic example provided in this section about the policy decision process, such complex details are not included.

Another example of a deployment variable concerns authentication levels. In a real-world deployment, different resources on a deployment container (such as an application server or web server) might require different levels of authentication. Suffice to say a great deal of complexity is involved in providing a generic example of a policy decision process, especially one that applies equally to J2EE agents and web agents. For one, the process varies greatly depending on the specifics of the deployment. Many other factors can affect the policy decision process, such as the IP address, time zone, and policy expiration time.

Each deployment variable can add a layer of complexity, which might affect how an agent reacts and how OpenSSO Enterprise reacts. This section provides a simple example of a policy decision process that highlights the role of an agent. Therefore, many of the detailed tasks and interactions, especially those processes that occur in OpenSSO Enterprise are left out. Do not expect the deployment represented in this example to match the deployment at your site. This is a generalized example that is applicable to both web agents and J2EE agents. Some of the basic steps in the policy decision process are depicted in Figure A–2. The figure is followed by a written description of the process. Notice that the figure illustrates the policy decision process in terms of the components the decision passes through. To see the policy decision process in a flow chart view, see Figure A–3 for the J2EE agent view and Figure A–4 for the web agent view.

For this example, in order to focus on stages of the process most relevant to Policy Agent, certain conditions are assumed as follows:

The user is attempting to access a protected resource after having already accessed a protected resource on the same Domain Name Server (DNS) domain. When the user accessed the first protected resource, OpenSSO Enterprise started a session. The user's attempt to access a second resource, makes this user's session a single sign-on (SSO) session. Therefore, at this point, the following already occurred:

- The user attempted to access a protected resource through a browser (the first resource that the user attempts to access during this session).
- The browser request was intercepted by the agent.
- The browser was redirected to a login uniform resource locator (URL), which is the interface to OpenSSO Enterprise Authentication Service.
- After the user entered valid credentials, the service authenticated the credentials.

The following figure and the corresponding step descriptions demonstrate what occurs after a previously authenticated user attempts to access a second protected resource through a browser. This figure depicts user profiles and policy stored together. Note that these data types are often stored separately.

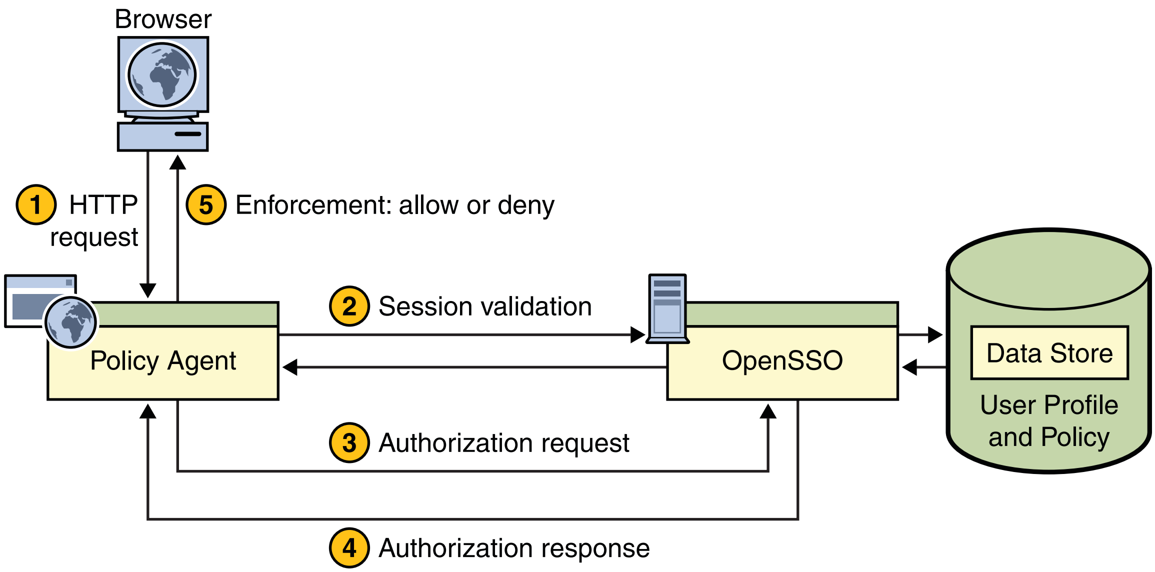

FIGURE A–2 Policy Agent and the Policy Decision Process

1. The browser sends a request for the protected resource to the deployment container (such as a web or application server) protected by the agent.
2. The agent intercepts the request, checks for a session token embedded in a cookie, and validates the session via OpenSSO Enterprise session service.
3. The agent sends a request to OpenSSO Enterprise Policy Service for user access to the protected resource.
4. OpenSSO Enterprise replies with the policy decision.
5. The agent interprets the policy decision and allows or denies access.

Examples of the Policy Decision Process by Agent Type

This section illustrates the policy decision process through the use of flow charts: one for J2EE agents and one for web agents. These two flow charts show how J2EE agents and web agents can differ in that J2EE security can be enabled for J2EE agents.

These charts illustrate possible scenarios that can take place when an end user makes a request for a resource. Therefore, the end user points a browser to a URL. That URL is a resource, such as a JPEG image, HTML page, JSP page, etc. When a resource is under the sphere of influence of the agent, the agent intervenes to varying degrees, depending on the specifics of the situation, checks the request, and takes the appropriate action, which culminates with the user either being allowed or denied access to the resource. The charts that follow reflect the potential paths a request makes before finally being allowed or denied.

Policy Decision Process for J2EE Agents

The figure that follows is a flow chart of the policy decision process for J2EE agents. This figure illustrates how a single request is processed and how the filter mode is involved in the process.

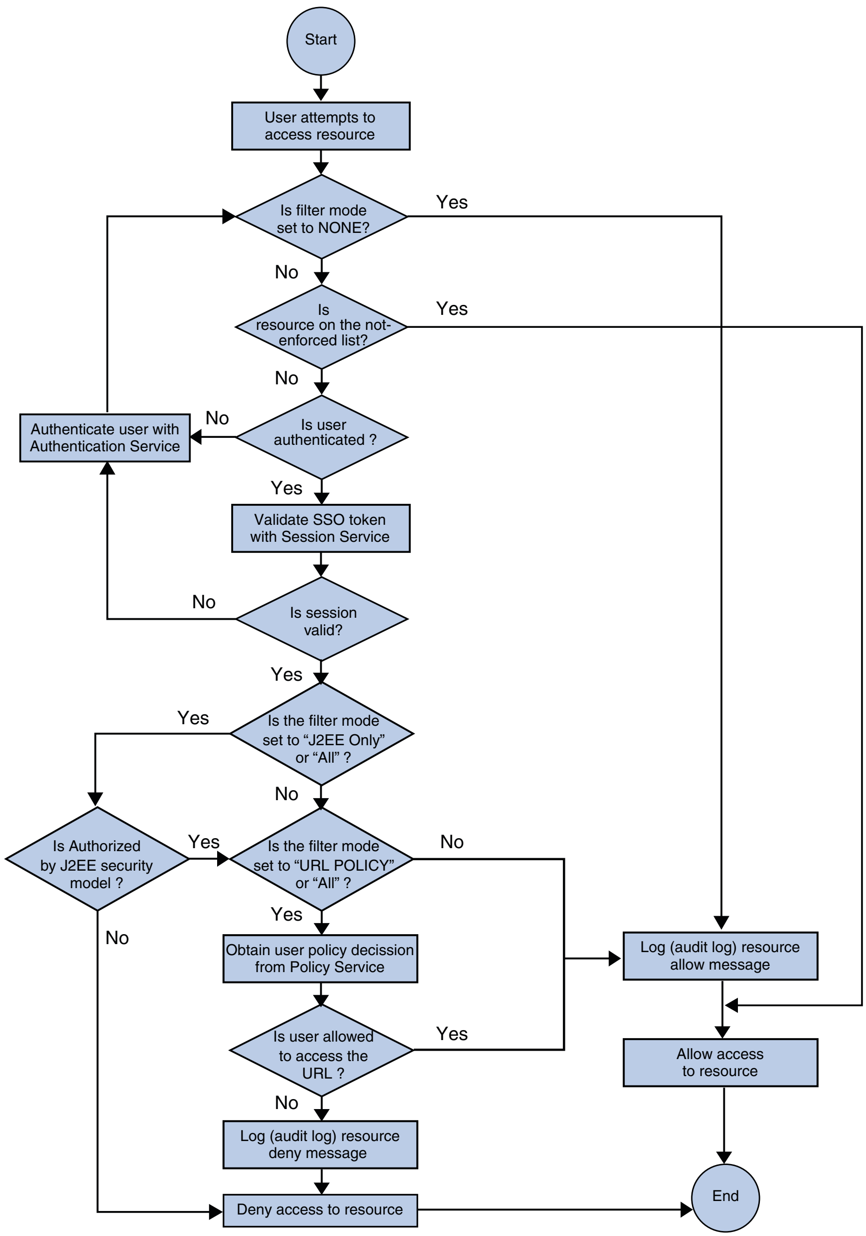

FIGURE A–3 J2EE Agents and the Policy Decision Process

Policy Decision Process for Web Agents

The figure that follows is a flow chart of the policy decision process for web agents. This figure illustrates how a single request is processed.

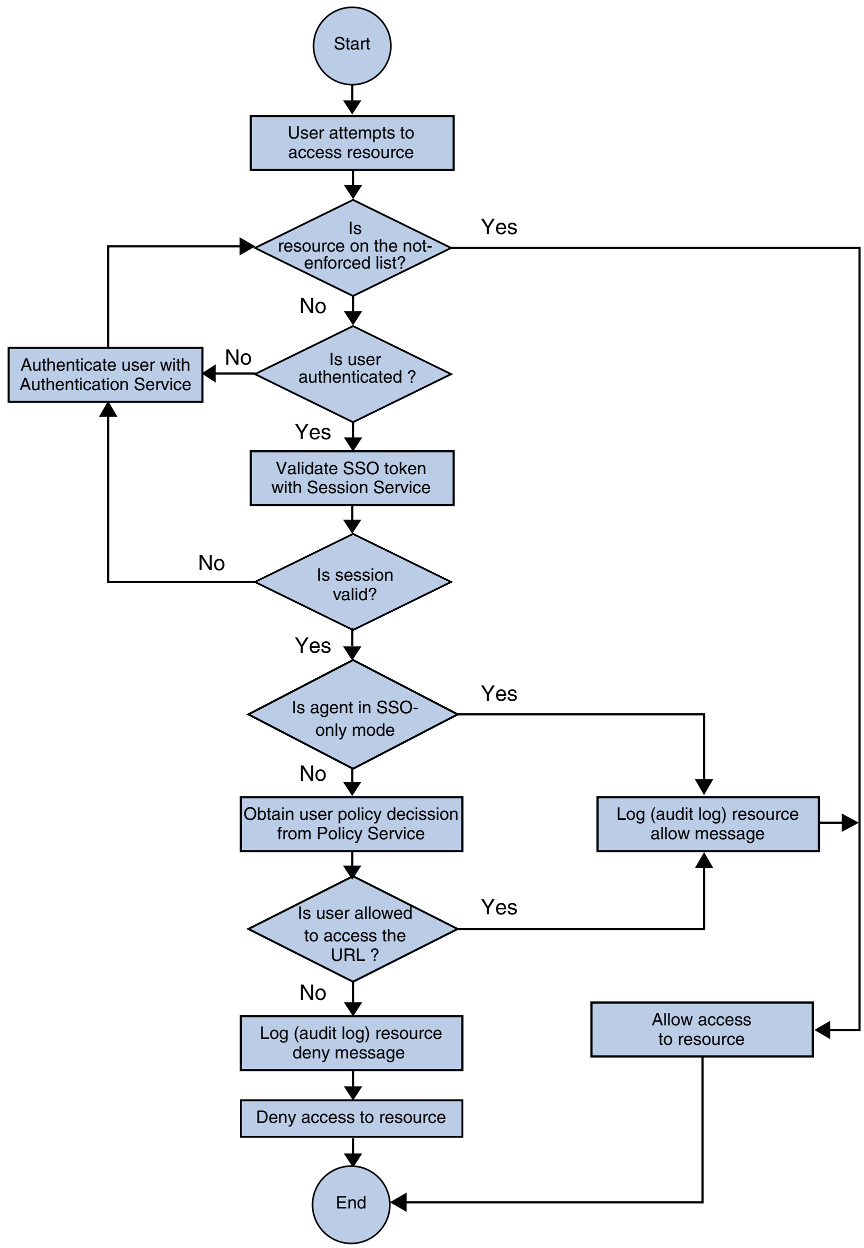

FIGURE A–4 Web Agents and the Policy Decision Process

Web Agents and J2EE Agents: Similarities and Differences

Both web agents and J2EE agents protect resources hosted on deployment containers (such as web and application servers) or enforce single sign-on with systems that use deployment containers as the front-end in an environment secured by OpenSSO Enterprise. The two types of agents are similar in some ways and yet different in others as outlined in this section.

Web Agents

Web agents control access to content on web servers and proxy servers. The content that web agents can protect include a multitude of services and web resources based on policies configured by an administrator. When a user points a browser to a URL deployed on a protected web or proxy server, the agent intercepts the request and validates the user's session token, if any exists. If the token's authentication level is insufficient (or none exists), the appropriate Authentication Service is called for a login page, prompting the user for (further) authentication. The Authentication Service verifies that the user credentials are valid. After the user's credentials are properly authenticated, the agent examines all the groups (which contain the policies) assigned to the user. Based on the aggregate of all policies assigned to the user, the individual is either allowed or denied access to the URL.

J2EE Agents

OpenSSO Enterprise provides agents for protecting J2EE applications in a variety of deployment containers, such as application and portal servers.

A J2EE agent can be installed for protecting a variety of hosted J2EE applications, which might require a varying set of security policy implementation. The security infrastructure of J2EE provides declarative as well as programmatic security that are platform-independent and are supported by all the J2EE-compliant servers. For details on how to use J2EE platform declarative as well as programmatic security, refer to J2EE documentation at `http://java.sun.com/j2ee`.

The agent helps enable role-to-principal mapping for protected J2EE applications with OpenSSO Enterprise principals. Therefore, at runtime, when a J2EE policy is evaluated, the evaluation is against the information available in OpenSSO Enterprise. Using this functionality, you can configure hosted J2EE applications so that they are protected by the J2EE agent, which provides real security services and other key features such as single sign-on. Apart from enabling J2EE security for hosted applications, J2EE agents also provide complete support for OpenSSO Enterprise based URL policies for enforcing access control over web resources hosted in deployment containers, such as an application servers.

While web agents and J2EE agents both work with OpenSSO Enterprise to implement authentication and authorization processes, the design of the J2EE agents allows them to also enforce J2EE security. You can see this difference in terms of enforcing J2EE security by comparing the flow charts of the two agents types: Figure A–3 and Figure A–4. The J2EE agents are generally comprised of two components (although this is partially subject to the interfaces exposed and supported by the deployment container): an agent filter for authentication and an agent realm for authorization.

Agent Filter and Authentication

In J2EE agents, the agent filter component manages authentication and URL policy related authorization. The agent filter is a servlet filter, which is supported starting with J2EE 1.3. The agent filter intercepts an inbound request to the server. It checks the request to see if it contains a session token. If one is available, the agent filter validates the token using OpenSSO Enterprise Session Service. If no token is available, the browser is redirected to the Authentication Service as in a typical SSO exchange. Once the user credentials are authenticated, the request is directed back to the server where the agent filter once again intercepts it, and then validates the newly acquired token. After the user's credentials are validated, the filter enforces J2EE policies or fine-grained URL policies on the resource the user is trying to access. Through this mechanism, the agent filter ensures that only requests with a valid OpenSSO Enterprise token are allowed to access a protected application.

Agent Realm and Authorization

In J2EE agents, the agent realm component facilitates the authorization related to J2EE security policies defined in the applications. A *realm* is a means for a J2EE-compliant application server to provide information about users, groups, and access control to applications deployed on it. It is a scope over which security policy is defined and enforced.

The server is configured to use a specific realm for validation of users and their roles, when attempts are made to access resources. By default, many application servers ship with a number of realm implementations, including the default File Based as well as LDAP, NT, UNIX, and Relational Database Management System (RDBMS). The agent realm component implements the server's realm interface, and allows user and role information to be managed by the OpenSSO Enterprise deployment. The agent realm component makes it possible to provide granular role-based authorization of J2EE resources to users who have been authenticated by the agent filter component.

Key Similarities of the Two Agent Types

The section "A Generalized Example of the Policy Decision Process" on page 101 describes a deployment that emphasizes the similar tasks performed by web agents and J2EE agents. The two agent types share various other features and tasks that are *not* described in that section. Though this section describes similarities of the two agent types, the features and tasks that they have in common tend to have some differences. However, those differences are often subtle. A list of key features and tasks that web agents and J2EE agents have in common follows along with an explanation of each item:

- "Configuration Properties" on page 110

- "Policy Agent Log Files" on page 110
- "Not-Enforced Lists" on page 110
- "Personal Profile Attributes, Session Attributes, and Policy Response Attributes." on page 111

Configuration Properties

Configuration properties for both agent types tend to be very similar in terms of functionality.

All agent configurations have an `OpenSSOAgentBootstrap.properties` file. Regardless of the configuration type, local or centralized, a small subset of properties that are required for the agent to start up and initialize itself are stored in the bootstrap file locally on the server where the agent is installed. This bootstrap file can only be configured by editing it directly. You cannot use either OpenSSO Enterprise Console or the `ssoadm` command-line utility to edit this file.

When the agents are installed using a centralized configuration (an option available starting with Policy Agent 3.0), both agent types allow the configuration properties, except those located in the `OpenSSOAgentBootstrap.properties` file, to be configured using either OpenSSO Enterprise Console or the `ssoadm` command-line utility.

If an agent is configured locally on the agent host, then for both agent types, the properties that are not stored in the bootstrap file must be configured by directly editing the `OpenSSOAgentConfiguration.properties` file.

Policy Agent Log Files

Web agents and J2EE agents can log access information and diagnostic information to an agent log file. Each agent has its own log file, a flat file located on the same host system as the agent. The log file size is configurable. When the active log file reaches the size limit, the log is rotated, which means that the older log information is moved and stored in another log file.

Furthermore, both agent types are capable of logging access information to an OpenSSO Enterprise log file or database table.

Not-Enforced Lists

Both agent types support not-enforced lists. These lists allow for the regular authentication and authorization processes to be bypassed. Two types of not-enforced lists exist: a not-enforced URI/URL list and a not-enforced IP Address list.

A not-enforced URI/URL list is a list of URIs or URLs that are not protected by an agent.

- Web agents support a not-enforced URL list. For example, `http://agentHost:port/myapp`
- J2EE agents support a not enforced URI list. For example, `/myapp/index.html`.

A resource represented by a URI/URL on a not-enforced URI/URL list is widely available, without restrictions. This list can be set to have a reverse meaning. With a reverse meaning, only URIs/URLs on the list are protected. All other URIs/URLs are not protected.

A not-enforced IP Address list is a list of IP addresses that are automatically allowed access to resources. When a user is using a computer that has an IP address on the not-enforced IP address list, that user is allowed access to all the resources protected by the respective agent.

Personal Profile Attributes, Session Attributes, and Policy Response Attributes.

Both agent types can fetch and pass along personal profile attributes, session attributes, and policy response attributes. Client applications protected by an agent can then use information from these attributes to personalize content for the user.

Key Differences Between the Two Agent Types

Many differences exist between J2EE agents and web agents in the way they perform tasks. However, the basic tasks they perform are similar. While the primary purpose of both types of agents is to enforce authentication and authorization before a user can access a protected resource, the two agent types differ in the kind of resources that they can protect and in the way they enforce such policy decisions.

Default Scope of Protection

When installed, a web agent automatically protects the entire set of resources available on the web server. On the other hand, J2EE agents only protect resources within a web application hosted on an application server after the web application has been configured to use the J2EE agent. Thus, if multiple web applications are hosted on an application server on which a J2EE agent has been installed, only the web applications that have been specifically configured to use the J2EE agent will be protected by the agent. Other applications will remain unprotected and can potentially malfunction if they depend upon any J2EE security mechanism.

Further, the J2EE agent can only protect resources that are packaged within a web or enterprise application. Certain application servers provide an embedded web server that can be used to host non-packaged web content such as HTML files and images. Such content cannot be protected by a J2EE agent unless it is redeployed as a part of a web application.

Modes of Operation

J2EE agents provide more modes of operation than do web agents. These modes are basically methods for evaluating and enforcing access to resources. You can set the mode according to your site's security requirements. For example, the SSO_ONLY mode is a less restrictive mode. This mode uses only OpenSSO Enterprise Authentication Service to authenticate users who attempt to access a protected resource.

Some of the modes such as SSO_ONLY and URL_POLICY are also achievable with web agents, whereas other modes of operation such as J2EE_POLICY and ALL modes do not apply to web agents. For web agents, by default, SSO_ONLY and URL_POLICY modes are both on. If you want only SSO set, enable the property labeled SSO Only Mode (Tab: Global, Name: `com.sun.identity.agents.config.sso.only`).

For both J2EE agents and web agents, the modes can be set in OpenSSO Enterprise Console.

In the J2EE_POLICY and ALL modes of operation, J2EE agents enforce J2EE declarative policies as applicable for the protected application.

APPENDIX B

Silent Installation and Uninstallation of a J2EE Agent in Policy Agent 3.0

In addition to a standard installation and uninstallation of J2EE agents, you can perform a silent installation or uninstallation. This appendix provides a description of how to use the silent option for the installation and uninstallation of J2EE agents.

About Silent Installation and Uninstallation of a J2EE Agent in Policy Agent 3.0

A silent installation or uninstallation refers to installing or uninstalling a program by implementing a script. The script is part of a state file. The script provides all the answers that you would normally supply to the installation or uninstallation program interactively. Running the script saves time and is useful when you want to install or uninstall multiple instances of Policy Agent using the same parameters in each instance.

Silent installation is a simple two-step process of generating a state file and then using that state file. To generate a state file, you record the installation or uninstallation process, entering all the required information that you would enter during a standard installation or uninstallation. Then you run the installation or uninstallation program with the state file as the input source.

Generating a State File for a J2EE Agent Installation

This section describes how to generate a state file for installing a J2EE agent. This task requires you to issue a command that records the information you will enter as you follow the agent installation steps. Enter all the necessary installation information in order to create a complete state file.

▼ To Generate a State File for a J2EE Agent Installation

To generate a state file for a J2EE agent installation, perform the following:

1. **Change to the following directory:**

   PolicyAgent-base/`bin`

   This directory contains the `agentadmin` program, which is used for installing a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Role of the `agentadmin` Program in Policy Agent 3.0" on page 35.

2. **Issue the following command:**

   `./agentadmin --install --saveResponse` *filename*

`-saveResponse`	An option that saves all of your responses to installation prompts in a state file.
filename	Represents the name that you choose for the state file.

3. **Answer the prompts to install the agent.**

   Your answers to the prompts are recorded in the state file. When the installation is complete, the state file is created in the same directory where the installation program is located.

Using a State File for a J2EE Agent Silent Installation

The installation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent installation.

▼ To Install a J2EE Agent Using a State File

To perform a silent installation of a J2EE agent using a state file, perform the following:

1. **Change to the following directory:**

   PolicyAgent-base/`bin`

   At this point, this `bin` directory should contain the `agentadmin` program and the J2EE agent installation state file.

2. **Issue the following command:**

   `./agentadmin --install --useResponse` *filename*

`-useResponse`	An option that directs the installer to run in non-interactive mode as it obtains all responses to prompts from the named state file.

filename	Represents the name of the state file from which the installer obtains all responses.

The installation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

Generating a State File for a J2EE Agent Uninstallation

This section describes how to generate a state file for uninstalling a J2EE agent. This task requires you to issue a command that records the information you will enter as you follow the agent uninstallation steps. Enter all the necessary uninstallation information in order to create a complete state file.

▼ To Generate a State File for a J2EE Agent Uninstallation

To generate a state file for uninstallation of a J2EE agent, perform the following:

1 Change to the following directory:

PolicyAgent-base`/bin`

This directory contains the `agentadmin` program, which is used for uninstalling a J2EE agent and for performing other tasks. For more information on the `agentadmin` program, see "Role of the `agentadmin` Program in Policy Agent 3.0" on page 35.

2 Issue the following command:

`./agentadmin --uninstall --saveResponse` *filename*

`-saveResponse`	An option that saves all of your responses to uninstallation prompts in a state file.
filename	Represents the name that you choose for the state file.

3 Answer the prompts to uninstall the agent.

Your answers to the prompts are recorded in the state file. When uninstallation is complete, the state file is created in the same directory where the uninstallation program is located.

Using a State File for a J2EE Agent Silent Uninstallation

The uninstallation program does not validate inputs or the state in the silent installation. Ensure that the proper environment exists before performing a silent uninstallation.

To Uninstall a J2EE Agent Using a State File

To perform a silent uninstallation of a J2EE agent using a state file, perform the following:

1. **Change to the following directory:**

   PolicyAgent-base/`bin`

   At this point, this `bin` directory should contain the `agentadmin` program and the J2EE agent uninstallation state file.

2. **Issue the following command:**

   ```
   ./agentadmin --uninstall --useResponse filename
   ```

`-useResponse`	An option that runs the uninstallation process in non-interactive mode as all responses to prompts are obtained from the named state file.
filename	Represents the name of the state file from which the installer obtains all responses.

   The uninstallation takes place hidden from view. After completion, the program exits automatically and displays the prompt.

APPENDIX C

Wildcard Matching in Policy Agent 3.0 J2EE Agents

The OpenSSO Enterprise policy service supports policy definitions that use either of the two following wildcards:

- "The Multi-Level Wildcard: *" on page 118
- "The One-Level Wildcard: -*-" on page 119

These wildcards can be used in policy related situations. For example, when using OpenSSO Enterprise Console or the `ssoadm` utility to create policies or when configuring the Policy Agent property that establishes the not-enforced list.

Caution – When issuing the `ssoadm` command, if you include values that contain wildcards (`*` or `-*-`), then the name/value pair should be enclosed in double quotes to avoid substitution by the shell. For more information about the `ssoadm` command, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."

For creating a policy, the following are feasible examples of the wildcards in use:

```
http://agentHost:8090/agentsample/*
```

```
http://agentHost:8090/agentsample/example-*-/example.html
```

For the not-enforced list, the following are feasible examples of the wildcards in use:`/agentsample.com/*.gif` and `/agentsample.com/-*-/images`.

Note –

- **No Support for Mixing Wildcards**: A policy resource can have either the multi-level wildcard (`*`) or the one-level wildcard (`-*-`), but not both. Using both types of wildcards in the same policy resource is not supported.
- **Handling Resources That Contain Query Strings**: Some resources use a query string, which is the part of a URL that contains data to be passed to web applications. The following is a feasible example of a URL that contains a query string:

  ```
  http://AgentHost/path/app?query-string
  ```

  The question mark (`?`) is the separator. It is not part of the query string. Many scenarios exist in which query strings might be used. They can be used for personalization of the user's session. Sometimes an application might add some locale information for a page request. The following example demonstrates the use of such locale information:

  ```
  http://AgentHost.com:8080/sampleapp/main.jsp?language=en&country=US
  ```

  Starting with OpenSSO Enterprise, neither the multi-level wildcard (`*`) nor the one-level wildcard (`-*-`) match the question mark. This is a change in behavior from Access Manager, where the multi-level wildcard (`*`) and the one-level wildcard (`-*-`) both matched the question mark. Because of this change in behavior, when you want to define a policy resource for OpenSSO Enterprise that can handle the question mark, use the multi-level wildcard on both sides of a question mark, as follows: `*?*` (asterisk-question mark-asterisk).

The Multi-Level Wildcard: *

The following list summarizes the behavior of the multi-level wildcard (the asterisk, `*`):

- Matches zero or more occurrences of any character except for the question mark (`?`).
- Spans across multiple levels in a URL
- Cannot be escaped. Therefore, the backslash character (`\`) or other characters cannot be used to escape the asterisk, as such `\*`.

The following examples show the multi-level wildcard character when used with the forward slash (/) as the delimiter character:

- The multi-level wildcard (`*`) matches zero or more characters, except the question mark, in the resource name, including the forward slash (`/`). For example, `...B-examp/*` matches `...B-examp/b/c/d`, but doesn't match `...B-examp/a?b=1`
- Multiple consecutive forward slash characters (`/`) do not match with a single forward slash character (`/`). For example, `...B-examp/*/A-examp` doesn't match `...B-examp/A-examp`.

- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, `...B-examp/` and `...B-examp//` are treated the same as `...B-examp`.

TABLE C–1 Examples of the Asterisk (*) as the Multi-Level Wildcard

Pattern	Matches	Does Not Match

`http://A-examp.com:8080/*`	`http://A-examp.com:8080`
	`http://A-examp.com:8080/`
	`http://A-examp.com:8080/index.html`
	`http://A-examp.com:8080/x.gif`
		`http://B-examp.com:8080/`
		`http://A-examp.com:8090/index.html`
		`http://A-examp.com:8080/a?b=1`

`http://A-examp.com:8080/*.html`	`http://A-examp.com:8080/index.html`
	`http://A-examp.com:8080/pub/ab.html`
	`http://A-examp.com:8080/pri/xy.html`
		`http://A-examp.com/index.html`
		`http://A-examp.com:8080/x.gif`
		`http://B-examp.com/index.html`

`http://A-examp.com:8080/*/ab`	`http://A-examp.com:8080/pri/xy/ab/xy/ab`
	`http://A-examp.com:8080/xy/ab`
		`http://A-examp.com/ab`
		`http://A-examp.com/ab.html`
		`http://B-examp.com:8080/ab`

`http://A-examp.com:8080/ab/*/de`	`http://A-examp.com:8080/ab/123/de`
	`http://A-examp.com:8080/ab/ab/de`
	`http://A-examp.com:8080/ab/de/ab/de`
	`http://A-examp.com:8080/ab//de`
		`http://A-examp.com:8080/ab/de`
		`http://A-examp.com:8090/ab/de`
		`http://B-examp.com:8080/ab/de/ab/de`

The One-Level Wildcard: -*-

The one-level wildcard (-*-) matches only the defined level starting at the location of the one-level wildcard to the next delimiter boundary. The “defined level” refers to the area between delimiter boundaries. Many of the rules that apply to the multi—level wildcard also apply to the one-level wildcard.

The following list summarizes the behavior of hyphen-asterisk-hyphen (-*-) as a wildcard:

- Matches zero or more occurrences of any character except for the forward slash and the question mark (?).
- Does not span across multiple levels in a URL.
- Cannot be escaped. Therefore, the backslash character (\) or other characters cannot be used to escape the hyphen-asterisk-hyphen, as such `\-*-`.

The following examples show the one-level wildcard when used with the forward slash (/) as the delimiter character:

- The one-level wildcard (-*-) matches zero or more characters (except for the forward slash and the question mark) in the resource name. For example, `...B-examp/-*-` doesn't match `...B-examp/b/c` or `...B-examp/a?b=1`
- Multiple consecutive forward slash characters (/) do not match with a single forward slash character (/). For example, `...B-examp/-*-/A-examp` doesn't match `...B-examp/A-examp`.
- Any number of trailing forward slash characters (/) are not recognized as part of the resource name. For example, `...B-examp/` and `...B-examp//` are treated the same as `...B-examp`.

TABLE C–2 Examples of the One—Level Wildcard (-*-)

Pattern	Matches	Does Not Match

`http://A-examp.com:8080/b/-*-`	`http://A-examp.com:8080/b`
	`http://A-examp.com:8080/b/`
	`http://A-examp.com:8080/b/cd/`
		`http://A-examp.com:8080/b/c?d=e`
		`http://A-examp.com:8080/b/cd/e`
		`http://A-examp.com:8090/b/`

`http://A-examp.com:8080/b/-*-/f`	`http://A-examp.com:8080/b/c/f` `http://A-examp.com:8080/b/cde/f`	`http://A-examp.com:8080/b/c/e/f` `http://A-examp.com:8080/f/`

`http://A-examp.com:8080/b/c-*-/f`	`http://A-examp.com:8080/b/cde/f`
	`http://A-examp.com:8080/b/cd/f`
	`http://A-examp.com:8080/b/c/f`
		`http://A-examp.com:8080/b/c/e/f`
		`http://A-examp.com:8080/b/c/`
		`http://A-examp.com:8080/b/c/fg`

APPENDIX D

Using the ssoadm Command-Line Utility With Agents

When the agent configuration is centralized, you can configure OpenSSO Enterprise through OpenSSO Enterprise Console or through the command line, using the ssoadm utility.

Note – The ssoadm utility cannot be used in scenarios where the agent configuration is stored locally with the agent.

The ssoadm utility has a set of subcommands that allow you to create and configure agents. All agent-related configurations that can be made using OpenSSO Enterprise Console can also be made using the command line. This appendix indicates which ssoadm subcommands are related to agents.

An ssoadm Command-Line Example Specific to Agents

This section provides an example of how you can use the ssoadm command-line for agent-related subcommands. This example highlights the update-agent option. The update-agent option allows you to configure agent properties. The following is an example of how the ssoadm command can be issued with the update-agent option:

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -a "com.sun.identity.agents.config.notenforced.uri[0]=/exampledir/public/*"
```

For the preceding command example, notice that a wildcard was used in the value for this particular property and that the property and value are enclosed in double quotes. The caution that follows addresses this issue. For more information about wildcards, see Appendix C, "Wildcard Matching in Policy Agent 3.0 J2EE Agents."

Caution – When issuing the ssoadm command, if you include values that contain wildcards (* or -*-), then the property name/value pair should be enclosed in double quotes to avoid substitution by the shell. This applies when you use the -a (--attributevalues) option. The double quotes are not necessary when you list the properties in a data file and access them with the -D option.

The format used to assign values to agent properties differs for OpenSSO Enterprise Console and the ssoadm command-line utility. For information about the format to use with the ssoadm utility, refer to the agent property file: OpenSSOAgentConfiguration.properties. This file demonstrates the correct format to use when assigning values to the agent properties using the ssoadm utility. Find this property file on the agent host machine in the following directory:

PolicyAgent-base/*AgentInstance-Dir*/config

For information on the place holders (*PolicyAgent-base* and *AgentInstance-Dir*) used in the preceding path, see "Policy Agent Software: Path and Directory Names" on page 13.

Listing the Options for an ssoadm Subcommand

You can read the options for a subcommand from this guide or you can list the options yourself while using the command. On the machine hosting OpenSSO Enterprise, in the directory containing the ssoadm utility, issue the ssoadm command with the appropriate subcommand. For example:

```
# ./ssoadm update-agent
```

Since the preceding command is missing required options, the utility merely lists all the options available for this subcommand. For example:

```
ssoadm update-agent --options [--global-options]
Update agent configuration.
Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]Global Options:
    --locale, -l
        Name of the locale to display the results.
```

```
    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --set, -s
        Set this flag to overwrite properties values.

    --attributevalues, -a
        properties e.g. homeaddress=here.

    --datafile, -D
        Name of file that contains properties.
```

Analysis of an `ssoadm` Subcommand's Usage Information

By looking at the usage information of a subcommand, you can determine which options are required and which are optional. You can list an option for the command with either a single letter, such as `-e` or with an entire word, such as `--realm`. The following is a list of the usage information for the `update-agent` subcommand:

```
ssoadm update-agent
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]
```

The options not bounded by square brackets are required. Therefore, `realm`, `agentname`, `adminid`, `password-file`. However, even though the three options in brackets (the global options) are considered optional, you must use either `--attributevalues` or `--datafile` to provide a property name and the corresponding value. The `--attributevalues` option is appropriate for assigning values to a single property. The `--datafile` option is appropriate for setting several properties at once. The `realm` and `agentname` options identify the specific agent you are configuring. The `adminid` and `password-file` commands identify you as someone who has the right to configure this agent.

The following command serves as an example of how you can change several agent properties at once. In this scenario the properties and their respective values are stored in a file, `/tmp/testproperties`, to which the command points:

```
# ./ssoadm update-agent -e testRealm1 -b testAgent1 -u amadmin -f
/tmp/testpwd -D /tmp/testproperties
```

Agent-Related Subcommands for the `ssoadm` Command

This sections lists the options available for each of the agent-related subcommands of the `ssoadm` command. The agent-related subcommands are presented as links in the following list:

- “The `ssoadm` Command: add-agent-to-grp subcommand” on page 124
- “The `ssoadm` Command: agent-remove-props subcommand” on page 125
- “The `ssoadm` Command: create-agent subcommand” on page 126
- “The `ssoadm` Command: create-agent-grp subcommand” on page 127
- “The `ssoadm` Command: delete-agent-grps subcommand” on page 128
- “The `ssoadm` Command: delete-agents subcommand” on page 129
- “The `ssoadm` Command: list-agent-grp-members subcommand” on page 130
- “The `ssoadm` Command: list-agent-grps subcommand” on page 131
- “The `ssoadm` Command: list-agents subcommand” on page 131
- “The `ssoadm` Command: remove-agent-from-grp subcommand” on page 132
- “The `ssoadm` Command: show-agent subcommand” on page 133
- “The `ssoadm` Command: show-agent-grp subcommand” on page 134
- “The `ssoadm` Command: show-agent-membership subcommand” on page 135
- “The `ssoadm` Command: show-agent-types subcommand” on page 136
- “The `ssoadm` Command: update-agent subcommand” on page 136
- “The `ssoadm` Command: update-agent-grp subcommand” on page 137

The `ssoadm` Command: add-agent-to-grp subcommand

```
ssoadm add-agent-to-grp--options [--global-options]
Add agents to a agent group.

Usage:
ssoadm
    --realm|-e
```

```
    --agentgroupname|-b
    --agentnames|-s
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.
    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupname, -b
        Name of agent group.

    --agentnames, -s
        Names of agents.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The `ssoadm` Command: agent-remove-props subcommand

```
ssoadm agent-remove-props --options [--global-options]
Remove agent's properties.

Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --attributenames|-a
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
```

```
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --attributenames, -a
        properties name(s).

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The ssoadm Command: create-agent subcommand

```
ssoadm create-agent --options [--global-options]
Create a new agent configuration.

Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --agenttype|-t
    --adminid|-u
    --password-file|-f
    [--attributevalues|-a]
    [--datafile|-D]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
```

```
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --agenttype, -t
        Type of agent. e.g. J2EEAgent, WebAgent

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --attributevalues, -a
        properties e.g. homeaddress=here.

    --datafile, -D
        Name of file that contains properties.
```

The `ssoadm` Command: create-agent-grp subcommand

```
ssoadm create-agent-grp --options [--global-options]
Create a new agent group.

Usage:
ssoadm
    --realm|-e
    --agentgroupname|-b
    --agenttype|-t
    --adminid|-u
    --password-file|-f
    [--attributevalues|-a]
    [--datafile|-D]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.
```

```
    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupname, -b
        Name of agent group.

    --agenttype, -t
        Type of agent group. e.g. J2EEAgent, WebAgent

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --attributevalues, -a
        properties e.g. homeaddress=here.

    --datafile, -D
        Name of file that contains properties.
```

The ssoadm Command: delete-agent-grps subcommand

```
ssoadm delete-agent-grps --options [--global-options]
Delete agent groups.

Usage:
ssoadm
    --realm|-e
    --agentgroupnames|-s
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.
```

```
    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupnames, -s
        Names of agent group.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The ssoadm Command: delete-agents subcommand

```
ssoadm delete-agents --options [--global-options]
Delete agent configurations.

Usage:
ssoadm
    --realm|-e
    --agentnames|-s
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentnames, -s
        Names of agent.

    --adminid, -u
        Administrator ID of running the command.
```

```
    --password-file, -f
        File name that contains password of administrator.
```

The `ssoadm` Command: list-agent-grp-members subcommand

```
ssoadm list-agent-grp-members --options [--global-options]
List agents in agent group.

Usage:
ssoadm
    --realm|-e
    --agentgroupname|-b
    --adminid|-u
    --password-file|-f
    [--filter|-x]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupname, -b
        Name of agent group.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --filter, -x
        Filter (Pattern).
```

The ssoadm Command: list-agent-grps subcommand

```
ssoadm list-agent-grps --options [--global-options]
List agent groups.

Usage:
ssoadm
    --realm|-e
    --adminid|-u
    --password-file|-f
    [--filter|-x]
    [--agenttype|-t]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --filter, -x
        Filter (Pattern).

    --agenttype, -t
        Type of agent. e.g. J2EEAgent, WebAgent
```

The ssoadm Command: list-agents subcommand

```
ssoadm list-agents --options [--global-options]
List agent configurations.

Usage:
ssoadm
    --realm|-e
```

```
    --adminid|-u
    --password-file|-f
    [--filter|-x]
    [--agenttype|-t]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --filter, -x
        Filter (Pattern).

    --agenttype, -t
        Type of agent. e.g. J2EEAgent, WebAgent
```

The `ssoadm` Command: remove-agent-from-grp subcommand

```
ssoadm remove-agent-from-grp --options [--global-options]
Remove agents from a agent group.

Usage:
ssoadm--realm|-e
    --agentgroupname|-b
    --agentnames|-s
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
```

```
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupname, -b
        Name of agent group.

    --agentnames, -s
        Names of agents.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The `ssoadm` Command: show-agent subcommand

```
ssoadm show-agent --options [--global-options]
Show agent profile.

Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--outfile|-o]
    [--inherit|-i]

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.
```

```
Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --outfile, -o
        Filename where configuration is written to.

    --inherit, -i
        Set this to inherit properties from parent group.
```

The ssoadm Command: show-agent-grp subcommand

```
ssoadm show-agent-grp --options [--global-options]
Show agent group profile.

Usage:
ssoadm
    --realm|-e
    --agentgroupname|-b
    --adminid|-u
    --password-file|-f
    [--outfile|-o]

Global Options:
    --locale, -l
    Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.
```

```
    --agentgroupname, -b
        Name of agent group.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --outfile, -o
        Filename where configuration is written to.
```

The `ssoadm` Command: show-agent-membership subcommand

```
ssoadm show-agent-membership --options [--global-options]
List agent?s membership.

Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The ssoadm Command: show-agent-types subcommand

```
ssoadm show-agent-types --options [--global-options]
Show agent types.

Usage:
ssoadm
    --adminid|-u
    --password-file|-f

Global Options:
    --locale, -l
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.
```

The ssoadm Command: update-agent subcommand

```
ssoadm update-agent --options [--global-options]
Update agent configuration.
Usage:
ssoadm
    --realm|-e
    --agentname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]

Global Options:
    --locale, -l
        Name of the locale to display the results.
```

```
    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentname, -b
        Name of agent.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --set, -s
        Set this flag to overwrite properties values.

    --attributevalues, -a
        properties e.g. homeaddress=here.

    --datafile, -D
        Name of file that contains properties.
```

The ssoadm Command: update-agent-grp subcommand

```
ssoadm update-agent-grp --options [--global-options]
Update agent group configuration.
Usage:

ssoadm
    --realm|-e
    --agentgroupname|-b
    --adminid|-u
    --password-file|-f
    [--set|-s]
    [--attributevalues|-a]
    [--datafile|-D]

Global Options:
    --locale, -l
```

```
        Name of the locale to display the results.

    --debug, -d
        Run in debug mode. Results sent to the debug file.

    --verbose, -v
        Run in verbose mode. Results sent to standard output.

Options:
    --realm, -e
        Name of realm.

    --agentgroupname, -b
        Name of agent group.

    --adminid, -u
        Administrator ID of running the command.

    --password-file, -f
        File name that contains password of administrator.

    --set, -s
        Set this flag to overwrite properties values.

    --attributevalues, -a
        properties e.g. homeaddress=here.

    --datafile, -D
        Name of file that contains properties.
```

Index