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

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

Part No: 820–5816–11
November 14, 2008

081117@21288

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 Web Agents* provides an overview of how web agents work in the Sun OpenSSO Enterprise Policy Agent 3.0 release. This guide focuses on the features and tasks that apply to all web 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 web agents, such as installation information and agent-specific configuration, see the individual web agent guide for that agent.

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

Contents of this Chapter

Who Should Use This Book

This *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for Web 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)

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.

TABLE P–1 OpenSSO Enterprise Documentation Set *(Continued)*

Title	Description
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 Web Agents*
- *Sun OpenSSO Enterprise Policy Agent 3.0 User's Guide for J2EE 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.

Typographic Conventions

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

TABLE P–3 Typographic Conventions

Typeface	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*. A *cache* is a copy that is stored locally. Do *not* save the file. **Note:** Some emphasized items appear bold online.

Shell Prompts in Command Examples

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

TABLE P–4 Shell Prompts

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

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 web agent directory is agent specific, it merely serves as an example: *Agent-HomeDirectory*`/web_agents/sjsws_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.
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`

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 Send comments. 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 Web Agents*, and the part number is 820-5816.

◆ ◆ ◆ 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 50.

- 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 Web 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 29.
  - **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 30.

- 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.

- 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 37.

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 web agents in the Policy Agent 3.0 software set with previous releases of both Access Manager and Policy Agent.

Web 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 Web Agents in the Policy Agent 3.0 Release

This guide focuses on web agents. This chapter provides more information about how web agents function generally.

Note – You can gain a stronger understanding of web agents by reviewing the appendix Appendix A, "Comparing Web Agents and J2EE Agents in Policy Agent 3.0." The comparison provided in that appendix is helpful in that it provides an abundance of information about general Policy Agent functionality.

Uses of Web Agents

Web agents function with OpenSSO Enterprise to protect content on web servers and web proxy servers from unauthorized intrusions. They control access to services and web resources based on the policies configured by an administrator. Web agents perform these tasks while providing single sign-on (SSO) and, in most cases, cross domain single sign-on (CDSSO) capabilities as well as URL protection.

Web agents are installed on deployment containers for a variety of reasons. Here are three examples:

- A web agent on a human resources server prevents non-human resources personnel from viewing confidential salary information and other sensitive data.
- A web agent on an operations deployment container allows only network administrators to view network status reports or to modify network administration records.
- A web agent on an engineering deployment container allows authorized personnel from many internal segments of a company to publish and share research and development information. At the same time, the web agent restricts external partners from gaining access to the proprietary information.

In each of these situations, a system administrator must set up policies that allow or deny users access to content on a deployment container. For information on setting policies and for assigning roles and policies to users, see the *Sun Java System Access Manager 7.1 Administration Guide*.

How Web Agents Work

When a user points a browser to a particular URL on a protected deployment container, a variety of interactions take place as explained in the following numbered list. See the terminology list immediately following this numbered list for a description of terms.

1. The web agent intercepts the request and checks information in the request against not-enforced lists. If specific criteria are met, the authentication process is bypassed and access is granted to the resource.
2. If authentication is required, the web agent validates the existing authentication credentials. If the existing authentication level is insufficient, the appropriate OpenSSO Enterprise Authentication Service will present a login page. The login page prompts the user for credentials such as username and password.
3. The authentication service verifies that the user credentials are valid. For example, the default LDAP authentication service verifies that the username and password are stored in the user data store. You might use other authentication modules such as RADIUS and Certificate modules.
4. If the user's credentials are properly authenticated, the web agent checks if the users is authorized to access the resource.
5. Based on the aggregate of all policies assigned to the user, the individual is either allowed or denied access to the URL.

Terminology: How Web Agents Work

Authentication Level	The ability to access resources can be divided into levels. Therefore, different resources on a deployment container (such as a web server or proxy server) might require different levels of authentication
Service	OpenSSO Enterprise is made of many components. A service is a certain type of component that performs specific tasks. Some of the OpenSSO Enterprise services available are Authentication Service, Session Service, Logging Service, and Policy Service.
Authentication Module	An authentication interface, also referred to as an authentication module, is used to authenticate a user on OpenSSO Enterprise.
Policy	A policy defines rules that specify access privileges to protected resources on a deployment container, such as a web server.

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. The chart is useful in that it demonstrates to some degree how web 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 web 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.

You can see how this web agent-specific flow chart compares to the J2EE agent flow chart as illustrated in "Examples of the Policy Decision Process by Agent Type" on page 87. The comparison gives a sense of how the two agent types differ in how they handle requests for resources.

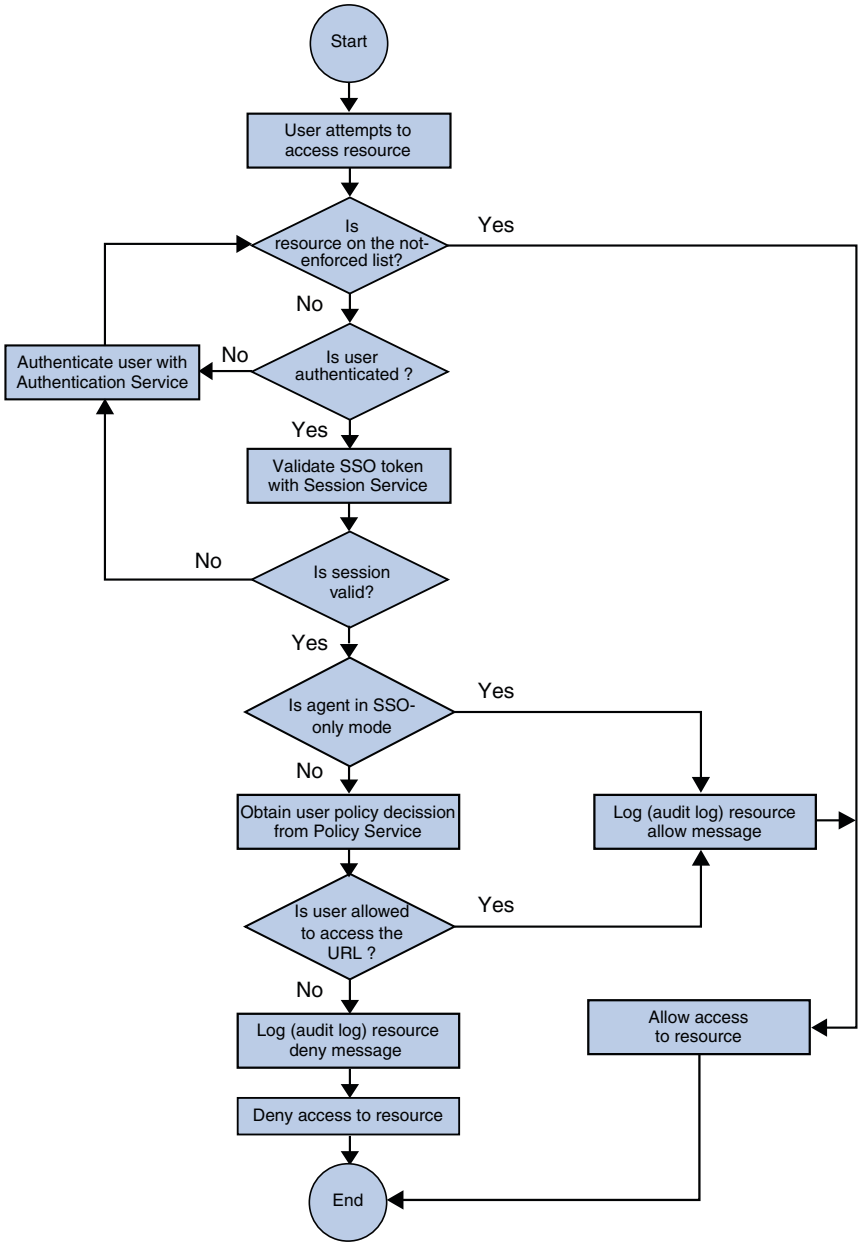

FIGURE 2–1 Policy Agent and the Policy Decision Process

◆ ◆ ◆ CHAPTER 3

Vital Installation Information for a Web Agent in Policy Agent 3.0

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

The information for this chapter is presented in the following sections:

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.

Web 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 Web Agent Base Directory in Policy Agent 3.0

Unpacking the web agent binaries creates a directory named `web_agents`, within which an agent-specific directory is created. Therefore, this directory name is different for each agent. For example, `sjsws_agent` is the directory name applicable to Agent for Sun Java System Web Server.

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, which follows.

EXAMPLE 3–1 Policy Agent Base Directory

The directory you choose in which to unpack the web agent binaries is referred to here as *Agent-HomeDirectory*. For illustration purposes, the following example uses Policy Agent 3.0 for Sun Java System Web Server 7.0 to demonstrate the path to the *PolicyAgent-base*. Be aware, that the directory `sjsws_agent` is only an example. This directory will vary from agent to agent:

Agent-HomeDirectory`/web_agents/sjsws_agent`

References in this book to the *PolicyAgent-base* directory are references to the preceding path.

Inside the Web Agent Base Directory in Policy Agent 3.0

After you finish installing an agent by issuing the `agentadmin ---install` command and interacting with the installer, you will need to access web 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 web 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 this guide 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 Web Agent

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

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:

`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 28.

`installer-logs` — This directory contains installation-related log files, for example log files created after you issue the `agentadmin` command.

Log information is stored in the installation log file after you install a web 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 web agent. Once you have successfully executed one run of the agentadmin --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 a web server instance. Depending on the number of times the agentadmin --install command is run, the number that replaces the x in the Agent_00x directory name will vary.

Furthermore, the string "Agent" in Agent_00x is configurable. You can change this string by editing the following file: *PolicyAgent-base*/config/OpenSSOInstallerConfig.properties. Search for the following setting:

```
com.sun.identity.install.tools.product.shortname=Agent
```

After you successfully install the first instance of a web 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 web 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.

`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

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

assigns the Policy Agent profile name and encrypted password in the respective agent instance. 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:

```
com.sun.identity.agents.config.key
```

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:

```
com.sun.identity.agents.config.key = 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.

Policy Agent 3.0: Web Agent Properties

The web agent properties changed names from the 2.2 release to the 3.0 release. This section lists those name changes. Furthermore, when applicable, this section provides the property label used with the property names. In prior releases, only property names were used for the properties. However, in Policy Agent 3.0 you can centralize the properties on the OpenSSO Enterprise Console, where labels are more useful.

Web Agent Properties in the `OpenSSOAgentBootstrap.properties` File

The properties listed in the table that follows can be configured by accessing the `OpenSSOAgentBootstrap.properties` file. This properties file, which is new for the 3.0 release, resides locally on the system where the agent is installed and stores the properties required for the agent to start up and initialize itself.

The properties listed in the web agent `OpenSSOAgentBootstrap.properties` file are either new for 3.0 or their property names have changed as indicated in the table. Labels are not assigned for the properties in this file.

Former Web Agent Property Name	Web Agent 3.0 Property Name
com.sun.am.naming.url	com.sun.identity.agents.config.naming.url
com.sun.am.log.level	com.sun.identity.agents.config.debug.level
com.sun.am.policy.agents.config.local.log.file	com.sun.identity.agents.config.local.logfile
com.sun.am.policy.am.username	com.sun.identity.agents.config.username
com.sun.am.policy.am.password	com.sun.identity.agents.config.password
com.sun.am.sslcert.dir	com.sun.identity.agents.config.sslcert.dir
com.sun.am.certdb.prefix	com.sun.identity.agents.config.certdb.prefix
com.sun.am.certdb.password	com.sun.identity.agents.config.certdb.password
com.sun.am.auth.certificate.alias	com.sun.identity.agents.config.certificate.alias
com.sun.am.trust_server_certs	com.sun.identity.agents.config.trust.server.certs
com.sun.am.receive_timeout	com.sun.identity.agents.config.receive.timeout
com.sun.am.connect_timeout	com.sun.identity.agents.config.connect.timeout
com.sun.am.tcp_nodelay.enable	com.sun.identity.agents.config.tcp.nodelay.enable

Former Web Agent Property Name	Web Agent 3.0 Property Name
New	com.sun.identity.agents.config.organization.name
New	com.sun.identity.agents.config.key
New	com.sun.identity.agents.config.debug.file
New	com.sun.identity.agents.config.forward.proxy.host
New	com.sun.identity.agents.config.forward.proxy.port
New	com.sun.identity.agents.config.forward.proxy.user
New	com.sun.identity.agents.config.forward.proxy.password
New	com.sun.identity.agents.config.profilename

Web Agent Properties Available Using the OpenSSO Enterprise Console or Other Methods

This section does not describe the agent properties. For a description of the agent properties, see the following link: http://wikis.sun.com/display/OpenSSO/agent3properties

The properties listed in the various tables that follow can be configured using any of the three following methods, depending on how agent is deployed:

- Using the OpenSSO Enterprise Console (only available with the centralized agent configuration option)
- Using the `ssoadm` command line (a centralized agent configuration option)
- Editing the `OpenSSOAgentConfiguration.properties` file (only available with the local agent configuration option)

The property names have changed for the 3.0 release as indicated in the various tables that follow in this section. Labels are associated with most of these properties, as indicated. The labels are most useful when using the Console.

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.am.login.url	com.sun.identity.agents.config.login.url **Label:** OpenSSO Login URL
com.sun.am.cookie.name	com.sun.identity.agents.config.cookie.name **Label:** Cookie Name

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.cookie.secure	com.sun.identity.agents.config.cookie.secure **Label:** Cookie Security
com.sun.am.policy.agents.config.local.log.rotate	com.sun.identity.agents.config.local.log.rotate **Label:** Rotate Local Audit Log
com.sun.am.policy.agents.config.local.log.size	com.sun.identity.agents.config.local.log.size **Label:** Local Audit Log Rotation Size
com.sun.am.policy.agents.config.audit.accesstype	com.sun.identity.agents.config.audit.accesstype **Label:** Audit Access Types
com.sun.am.policy.agents.config.remote.log	com.sun.identity.agents.config.remote.logfile **Label:** Remote Log Filename
com.sun.am.notification.enable	com.sun.identity.agents.config.notification.enable **Label:** Enable Notifications
com.sun.am.policy.am.url_comparison.case_ignore	com.sun.identity.agents.config.url.comparison.case.ignore **Label:** URL Comparison Case Sensitivity Check

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.am.polling.interval	com.sun.identity.agents.config. policy.cache.polling.interval **Label:** Policy Cache Polling Period
com.sun.am.sso.polling.period	com.sun.identity.agents.config. sso.cache.polling.interval **Label:** SSO Cache Polling Period
com.sun.am.policy.am.userid.param	com.sun.identity.agents.config.userid.param **Label:** User ID Parameter
com.sun.am.policy.am.userid.param.type	com.sun.identity.agents.config.userid.param.type **Label:** User ID Parameter Type
com.sun.am.policy.agents.config. profile.attribute.fetch.mode	com.sun.identity.agents.config. profile.attribute.fetch.mode **Label:** Profile Attribute Fetch Mode

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.profile.attribute.map	com.sun.identity.agents.config. profile.attribute.mapping **Label:** Profile Attribute Mapping
com.sun.am.policy.agents.config. session.attribute.fetch.mode	com.sun.identity.agents.config.session. attribute.fetch.mode **Label:** Session Attribute Fetch Mode
com.sun.am.policy.agents.config.session.attribute.map	com.sun.identity.agents.config. session.attribute.mapping **Label:** Session Attribute Mapping
com.sun.am.policy.agents.config. response.attribute.fetch.mode	com.sun.identity.agents.config. response.attribute.fetch.mode **Label:** Response Attribute Fetch Mode
com.sun.am.policy.agents.config.response.attribute.map	com.sun.identity.agents.config. response.attribute.mapping **Label:** Response Attribute Mapping

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.load_balancer.enable	com.sun.identity.agents.config.load.balancer.enable **Label:** Load Balancer Setup
com.sun.am.policy.agents.config.agenturi.prefix	com.sun.identity.agents.config.agenturi.prefix **Label:** Agent Deployment URI Prefix
com.sun.am.policy.agents.config.locale	com.sun.identity.agents.config.locale **Label:** Agent Locale
com.sun.am.policy.agents.config.do_sso_only	com.sun.identity.agents.config.sso.only **Label:** SSO Only
com.sun.am.policy.agents.config.accessdenied.url	com.sun.identity.agents.config.access.denied.url **Label:** Resources Access Denied URL
com.sun.am.policy.agents.config.fqdn.check.enable	com.sun.identity.agents.config.fqdn.check.enable **Label:** FQDN Check
com.sun.am.policy.agents.config.fqdn.default	com.sun.identity.agents.config.fqdn.default **Label:** FQDN Default

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.fqdn.map	com.sun.identity.agents.config.fqdn.mapping **Label:** FQDN Virtual Host Map
com.sun.am.policy.agents.config.cookie.reset.enable	com.sun.identity.agents.config.cookie.reset.enable **Label:** Cookie Reset
com.sun.am.policy.agents.config.cookie.reset.list	com.sun.identity.agents.config.cookie.reset **Label:** Cookies Reset Name List

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.cookie.domain.list	com.sun.identity.agents.config.cookie.domain **Label:** Cookies Domain List
com.sun.am.policy.agents.config.anonymous_user	com.sun.identity.agents.config.anonymous.user.id **Label:** Anonymous User Default Value
com.sun.am.policy.agents.config.anonymous_user.enable	com.sun.identity.agents.config.anonymous.user.enable **Label:** Anonymous User
com.sun.am.policy.agents.config.notenforced_list	com.sun.identity.agents.config.notenforced.url **Label:** Not Enforced URLs
com.sun.am.policy.agents.config. notenforced_list.invert	com.sun.identity.agents.config.notenforced.url.invert **Label:** Invert Check for Not Enforced URLs
com.sun.am.policy.agents.config. notenforced_client_ip_list	com.sun.identity.agents.config.notenforced.ip **Label:** Not Enforced Client IP List
com.sun.am.policy.agents.config. ignore_policy_evaluation_if_notenforced	com.sun.identity.agents.config. notenforced.url.attributes.enable **Label:** Fetch Attributes for Notenforced URLs
com.sun.am.policy.agents.config. postdata.preserve.enable	com.sun.identity.agents.config. postdata.preserve.enable **Label:** POST Data Preservation
com.sun.am.policy.agents.config. postcache.entry.lifetime	com.sun.identity.agents.config. postcache.entry.lifetime **Label:** POST Data Entries Cache Period

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents. config.client_ip_validation.enable	com.sun.identity.agents.config. client.ip.validation.enable **Label:** Client IP Validation

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.profile.attribute. cookie.prefix	com.sun.identity.agents.config.profile.attribute. cookie.prefix **Label:** Profile Attributes Cookie Prefix
com.sun.am.policy.agents.config.profile.attribute. cookie.maxage	com.sun.identity.agents.config.profile.attribute. cookie.maxage **Label:** Profile Attributes Cookie Maxage
com.sun.am.policy.agents.config.cdsso.enable	com.sun.identity.agents.config.cdsso.enable **Label:** Cross Domain SSO
com.sun.am.policy.agents.config.cdcservlet.url	com.sun.identity.agents.config.cdsso.cdcservlet.url **Label:** CDSSO Servlet URL
com.sun.am.policy.agents.config.logout.url	com.sun.identity.agents.config.logout.url **Label:** OpenSSO Logout URL
com.sun.am.policy.agents.config.logout.cookie. reset.list	com.sun.identity.agents.config.logout.cookie.reset **Label:** Logout Cookies List for Reset
com.sun.am.policy.am.fetch_from_root_resource	com.sun.identity.agents.config.fetch.from. root.resource **Label:** Fetch Policies from Root Resource
com.sun.am.policy.agents.config.get_client_host_name	com.sun.identity.agents.config.get.client.host.name **Label:** Retrieve Client Hostname
com.sun.am.policy.agents.config.convert_mbyte.enable	com.sun.identity.agents.config.convert.mbyte.enable **Label:** Native Encoding of Profile Attributes
com.sun.am.policy.agents.config. encode_url_special_chars.enable	com.sun.identity.agents.config.encode. url.special.chars.enable **Label:** Encode URL's Special Characters

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.ignore_path_info	com.sun.identity.agents.config.ignore.path.info **Label:** Ignore Path Info in Request URL

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.override_protocol	com.sun.identity.agents.config.override.protocol **Label:** Override Request URL Protocol
com.sun.am.policy.agents.config.override_host	com.sun.identity.agents.config.override.host **Label:** Override Request URL Host
com.sun.am.policy.agents.config.override_port	com.sun.identity.agents.config.override.port **Label:** Override Request URL Port
com.sun.am.policy.agents.config. override_notification.url	com.sun.identity.agents.config. override.notification.url **Label:** Override Notification URL
com.sun.am.policy.agents.config.connection_timeout	com.sun.identity.agents.config.auth.connection.timeout **Label:** Agent Connection Timeout
com.sun.am.ignore_server_check	com.sun.identity.agents.config.ignore.server.check **Label:** Ignore Server Check
com.sun.am.poll_primary_server	com.sun.identity.agents.config.poll.primary.server **Label:** Polling Period for Primary Server
com.sun.am.ignore.preferred_naming_url	com.sun.identity.agents.config.ignore. preferred.naming.url **Label:** Ignore Preferred Naming URL in Naming Request
com.sun.am.policy.agents.config.proxy. override_host_port	com.sun.identity.agents.config.proxy. override.host.port **Label:** Override Proxy Server's Host and Port

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.domino. check_name_database	com.sun.identity.agents.config.domino. check.name.database **Label:** Check User in Domino Database
com.sun.am.policy.agents.config.domino. ltpa.enable	com.sun.identity.agents.config.domino. ltpa.enable **Label:** Use LTPA token
com.sun.am.policy.agents.config.domino. ltpa.cookie_name	com.sun.identity.agents.config.domino. ltpa.cookie.name **Label:** LTPA Token Cookie Name

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.domino.ltpa.config_name	com.sun.identity.agents.config.domino.ltpa.config.name **Label:** LTPA Token Configuration Name
com.sun.am.policy.agents.config.domino.ltpa.org_name	com.sun.identity.agents.config.domino.ltpa.org.name **Label:** LTPA Token Organization Name
com.sun.am.policy.agents.config.iis.auth_type	com.sun.identity.agents.config.iis.auth.type **Label:** Authentication Type
com.sun.am.replaypasswd.key	com.sun.identity.agents.config.replaypasswd.key **Label:** Replay Password Key
com.sun.am.policy.agents.config.iis.filter_priority	com.sun.identity.agents.config.iis.filter.priority **Label:** Filter Priority
com.sun.am.policy.agents.config.iis.owa_enabled	com.sun.identity.agents.config.iis.owa.enable **Label:** Filter configured with OWA
com.sun.am.policy.agents.config.iis.owa_enabled_change_protocol	com.sun.identity.agents.config.iis.owa.enable.change.protocol **Label:** Change URL Protocol to https

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
com.sun.am.policy.agents.config.iis.owa_enabled_session_timeout_url	com.sun.identity.agents.config.iis.owa.enable.session.timeout.url **Label:** Idle Session Timeout Page URL
NEW	com.sun.identity.agents.config.repository.location This product is available in OpenSSO Enterprise Console. However, only the label is provided, not the property name. **Label:** Location of Agent Configuration Repository
NEW	com.sun.identity.agents.config.freeformproperties **Label:** Custom Properties
NEW	com.sun.identity.agents.config.polling.interval **Label:** Configuration Reload Interval

Former Web Agent Property Name	Web Agent 3.0 Property Name and Label
NEW	`com.sun.identity.agents.config.cleanup.interval` **Label:** Configuration Cleanup Interval

Creating a Web Agent Profile in Policy Agent 3.0

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

Web 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 web agent authenticates with OpenSSO Enterprise by supplying an agent profile name and password. During the installation of the web agent, you must provide a valid agent profile name and the respective password to enable authentication attempts to succeed.

Creating a Web Agent Profile in Policy Agent 3.0

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

- Use the OpenSSO Enterprise Console as described in the task that follows, "To Create a Web Agent Profile in Policy Agent 3.0 Using OpenSSO Enterprise Console" on page 48. This method is commonly used when you want to create the agent profile as a pre-installation task.
- 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` utility with the `--custom-install`. For more information on the `agentadmin` utility, see "Role of the `agentadmin` Program in Policy Agent 3.0" on page 28.

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

This task applies when you want to create the web agent profile as a pre-installation task. Perform this task using 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`**.**

   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.**

   The Web tab is selected by default.

5. **Click New in the agent section.**

6. **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.

7. **In the Server URL field, enter the OpenSSO Enterprise server URL.**

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

8. **In the Agent URL field, enter the URL for the agent application.**

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

9. **Click Create.**

   The Console creates the agent profile and displays the Web 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 web agent groups.

Create a group if you want agents to inherit specific properties from the group. web agents can inherit properties from a web 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 Web 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 web 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 Web 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 the OpenSSO Enterprise 8.0 Console to the Web Agent Properties" on page 55.

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.

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

- `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.config.username` to the agent authenticator name.**

      Therefore, the setting would be as such:

      `com.sun.identity.agents.config.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.sun.identity.agents.config.password =` *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 35.

      ---

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

3. **Restart the agent container.**

Web Agent Task Reference for Policy Agent 3.0

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

▼ To Navigate in the OpenSSO Enterprise 8.0 Console to the Web 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.**

   By default the Web tab is selected.

5. **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 web agent properties, moving from tab to tab as necessary.

◆ ◆ ◆ CHAPTER 4

Common Web Agent Tasks and Features in Policy Agent 3.0

After installing the web 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 web agents generally. Therefore, the information in this chapter tends to apply to all web agents in the Policy Agent 3.0 software set.

Interaction with Policy Agent 3.0 is enabled to a great extent by configuring the agent properties. However, some interaction with the agent is performed using the `agentadmin` command as explained in "Role of the `agentadmin` Program in Policy Agent 3.0" on page 28.

This section focuses on features that involve setting web 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 list of every web agent property, see "Policy Agent 3.0: Web Agent Properties" on page 40. 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 web 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 web agent properties. For information about the `ssoadm` utility, see Appendix D, "Using the `ssoadm` Command-Line Utility With Agents."

When the web 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 web 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 "Web Agent Directory Structure in Policy Agent 3.0" on page 24.

The following topics are discussed in this section:

How Web 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 Web agent properties using OpenSSO Enterprise Console, see "To Navigate in the OpenSSO Enterprise 8.0 Console to the Web Agent Properties" on page 55.

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 Web 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 name of 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 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 Web Agents

Most web 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` file), 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.

Web Agent Properties That Are List Constructs

Certain web 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 Web Agent Properties That Are List Constructs

```
com.sun.identity.agents.config.notenforced.url[0] = http://agentHost.com:8080/public/*
com.sun.identity.agents.config.notenforced.url[1] = http://agentHost.com:8080/images/*
com.sun.identity.agents.config.notenforced.url[2] = http://agentHost.com:8080/index.html
```

Web 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 web 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 Web Agent Properties That Are Map Constructs

```
com.sun.identity.agents.config.fqdn.mapping[invalid_hostname] = valid_hostname
com.sun.identity.agents.config.profile.attribute.mapping[mail] = email
```

Providing Failover Protection for a Web Agent

When you install a web agent, you can specify a *failover* or backup deployment container, such as a web server, for running OpenSSO Enterprise. This is essentially a high availability option. It ensures that if the deployment container that runs OpenSSO Enterprise service becomes unavailable, the web agent still processes access requests through a secondary, or failover, deployment container running OpenSSO Enterprise service.

Setting up failover protection for the web agent, requires modifying a web agent property. However, you must first install two different instances of OpenSSO Enterprise on two separate deployment containers.

Then follow the instructions about installing the web agent. The web agent installation program prompts you for the host name and port number of the failover deployment container that you have configured to work with OpenSSO Enterprise. The property labeled OpenSSO Login URL (Tab: OpenSSO Services, Name: `com.sun.identity.agents.config.login.url`) stores the failover deployment container name. Furthermore, whenever you add a value to the OpenSSO Login URL property, you must configure the property named `com.sun.identity.agents.config.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file) as illustrated subsequently.

Set this property in order to store failover server information. Given the values in the following list, the property would be set as shown in Example 4–4.

`host1` Name of the primary OpenSSO Enterprise host.

`host2` Name of the failover OpenSSO Enterprise host.

example Name of the domain.

`58080` Default port number

EXAMPLE 4–4 Configuration Property Setting for Failover Protection of a Web Agent

Failover protection of a web agent is enabled by assigning values, as shown in the following list, to the OpenSSO Login URL property:

```
http://host1.example.com:58080/opensso/UI/Login
http://host2.example.com:58080/opensso/UI/Login
```

EXAMPLE 4–5 Configuration Property Setting for Naming Service of Web Agent

Adding a value to the OpenSSO Login URL property requires that you configure the following property:`com.sun.identity.agents.config.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file). For example, if the OpenSSO Login URL property is set as illustrated in Example 4–4, the following property should be set as shown:

EXAMPLE 4–5 Configuration Property Setting for Naming Service of Web Agent *(Continued)*

```
com.sun.identity.agents.config.naming.url =
http://host1.example.com:58080/opensso/namingservice
http://host2.example.com:58080/opensso/namingservice
```

Changing the Web Agent Caching Behavior

Each web agent maintains a cache that stores the policies for every user's session. The cache can be updated by a cache polling mechanism and a cache notification mechanism.

Cache Updates

A web agent maintains a cache of all active sessions involving content that the agent protects. Once an entry is added to an agent's cache, it remains valid for a period of time after which the entry is considered expired and later purged. This feature relies on a polling mechanism.

The web agent property labeled Policy Cache Polling Period (`com.sun.identity.agents.config.policy.cache.polling.interval`) determines the number of minutes an entry will remain in the web agent cache. Once the interval specified by this property has elapsed, the entry is dropped from the cache. By default, the expiration time is set to three minutes.

Hybrid Cache Updates

In this mode, cache entry expiration still applies through use of the polling mechanism. In addition, the web agent gets notified by the OpenSSO Enterprise service about session changes through use of a notification mechanism. Session changes include events such as session logout or a session timeout. When notified of a session or a policy change, the web agent updates the corresponding entry in the cache. Apart from session updates, web agents can also receive policy change updates. Policy changes include events such as updating, deleting, and creating policies.

Web agents have the hybrid cache update mode switched on by default. This is triggered by the web agent property labeled Enable Notifications `com.sun.identity.agents.config.notification.enable`. When this property is disabled, the web agent updates its cache through the polling mechanism only.

Restrictions due to firewalls, as well as the type of deployment container in use, might not allow notifications to work. In such cases, the notification mechanism is turned off.

The web agent sets a timeout period on its cache entries. After its end of life, the cache entry is purged from the web agent's cache. The web agent does not refetch the cache data. The next attempt to access the same entry from cache fails and the web agent makes a round trip to the server and fetches it again to populate the cache. This lazy method of cache updating keeps the web agent cache performing optimally and reduces network traffic.

In a normal deployment situation, policy changes on the server are frequent, which requires sites to accept a certain amount of latency for web agents to reflect policy changes. Each site decides the amount of latency time that is acceptable for the site's specific needs. When setting the Policy Cache Polling Period property, set it to the lower of the two:

- The session idle timeout period
- Your site's accepted latency time for policy changes

Configuring the Not-Enforced URL List

The *not-enforced URL list* defines the resources that should not have any policies (neither allow nor deny) associated with them.

By default, the web agent denies access to all resources on the deployment container that it protects. However, various resources (such as a web site or an application) available through a deployment container might not need to have any policy enforced. Common examples of such resources include the HTML pages and `.gif` images found in the home pages of web sites and the cascading style sheets (CSS) that apply to these home pages. The user should be able to browse such pages without authenticating. For the home page example, all these resources need to be on the not-enforced URL list or the page will not be displayed properly. The property labeled Not Enforced URLs (Tab: Application, Name: `com.sun.identity.agents.config.notenforced.url`) is used for this purpose. Wild cards can be used to define a pattern of URLs. For more information about the use of wildcards, see Appendix C, "Wildcard Matching in Policy Agent 3.0 Web Agents."

There can be a reverse, or "inverted", scenario when all the resources on the deployment container, except a list of URLs, are open to any user. In that case, the property labeled Invert Not Enforced URLs (Tab: Application, Name: `com.sun.identity.agents.config.notenforced.url.invert`) is used to reverse the meaning of the Not Enforced URLs property. If the Invert Not Enforced URLs property is enabled (by default it is not enabled), then the not-enforced URL list becomes the enforced list.

EXAMPLE 4–6 Configuration Property Settings for Not-Enforced URL List

The following are examples:

Scenario 1: Not-Enforced URL List

EXAMPLE 4–6 Configuration Property Settings for Not-Enforced URL List *(Continued)*

The Invert Not Enforced URLs property is *not* enabled.

The following URLs are listed as values for the Not Enforced URLs property:

```
http://host1.example.com:80/welcome.html
http://host1.example.com:80/banner.html
```

In this case, authentication and policies will not be enforced on the two URLs listed on the not-enforced list. All other resources will be protected by the web agent.

Scenario 2: Inverted Not-Enforced URL List

The Invert Not Enforced URLs property is enabled.

The following URLs are listed as values for the Not Enforced URLs property:

```
http://host1.example.com:80/welcome.html
 http://host1.example.com:80/banner.html
```

In this case, authentication and policies will be enforced by the web agent on the two URLs mentioned in the not-enforced list. All other resources will be accessible to any user.

Caution – If feasible, do not enable the Invert Not Enforced URLs property.

Not enabling this property reduces the chance of unintentionally allowing access to resources.

Configuring the Not-Enforced IP Address List

The property labeled Not Enforced Client IP List (Tab: Application, Name: `com.sun.identity.agents.config.notenforced.ip`) is used to specify a list of IP addresses. No authentication is required for the requests coming from these client IP addresses.

In other words, the web agent will not enforce policies for the requests originating from the IP addresses in the Not-Enforced IP Address list.

Enforcing Authentication Only

The property labeled SSO Only (Tab: Global, Name `com.sun.identity.agents.config.sso.only`) is used to specify if only authentication is enforced for URLs protected by the web agent. If this property is enabled (by default it is not enabled), it indicates that the web agent enforces authentication only, without enforcing policies. After a user logs into OpenSSO Enterprise server successfully, the web agent will not check for policies related to the user and the accessed URLs.

Providing Personalization Capabilities

Web agents in Policy Agent 3.0 can personalize page content for users in three distinct ways as described in the following subsections:

- "Providing Personalization With Session Attributes" on page 67
- "Providing Personalization With Policy-Based Response Attributes" on page 69
- "Providing Personalization With User Profile Attributes Globally" on page 69

Providing Personalization With Session Attributes

Web agents in Policy Agent 3.0 support a feature where a user's session attributes are fetched and set as headers or cookies.

Session attributes are especially effective for transferring information that is dynamic. However, the information transferred only lasts during the current session.

Unlike profile attributes, session attributes are not limited to LDAP attributes retrieved from the user data store. Since session attributes allow non-user profile attributes to be fetched, you can configure the deployment to fetch attributes such as SAML assertion.

The following are examples of session attributes: `UserToken`, `UserId`, `Principal`, `AuthType`, `AuthLevel`, `sun.am.UniversalIdentifier`, `MyProperty`

A good use case for fetching user session attributes presents itself when a post authentication plug-in is involved. A post authentication plug-in performs some tasks right after user authentication. You can configure a post authentication plug-in to fetch data from an external data repository and then set this data as session attributes for that user. These session attributes can be retrieved by the web container and made available to the application.

The web agent property labeled Session Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.session.attribute.fetch.mode`) is responsible for fetching session attributes. This property can be configured using OpenSSO Enterprise Console and can be set to one of the following values:

- NONE
- HTTP_HEADER
- HTTP_COOKIE

When set to NONE, no session attributes are fetched and the property labeled Session Attribute Map (Tab: Application, Name: com.sun.identity.agents.config.session.attribute.mapping) is ignored.

With the Session Attribute Fetch Mode property set to either HTTP_HEADER or HTTP_COOKIE, the web agent fetches session attributes. Use the Session Attribute Map property to configure attributes that are to be forwarded as HTTP headers or cookies.

This section illustrates how the Session Attribute Fetch Mode property maps session attributes to headers or cookies.

Session attributes are added to an HTTP header following this format:

```
session_attribute_name|http_header_name[,...]
```

The value of the attribute being fetched in session is session_attribute_name. This value gets mapped to a header value as follows: http_header_name.

Note – In most cases, in a destination application where http_header_name appears as a request header, it is prefixed with HTTP_ and the following type of conversion takes place:

Lower case letters	convert to upper case letters.
Hyphen "-"	converts to underscore "_"
"common-name"	as an example, converts to "HTTP_COMMON_NAME."

Therefore, the Session Attribute Map property would have the following value:

```
successURL | success-url, contextId | context-id
```

The session attribute is forwarded as a header or a cookie as determined by the end-user applications on the web container that the web agent is protecting. These applications can be considered the consumers of the forwarded header values. The forwarded information is used for the customization and personalization of web pages. You can also write server side plug-ins to put any user session attribute and define the corresponding attribute name and mapping in the preceding property to retrieve the value.

Providing Personalization With Policy-Based Response Attributes

Header attributes can also be determined by OpenSSO Enterprise policy configurations. With policy-based response attributes you can define attribute-value pairs at each policy.

Policy-based response attributes can improve the deployment process, allow greater flexibility in terms of customization, and provide central and hierarchical control of attribute values.

Web agents set policy-based response attributes as headers or cookies based on configuration. All subjects that match this attribute set obtain this attribute. The web agent property labeled Response Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.response.attribute.fetch.mode`) controls this functionality:

The default setting for this property is `HTTP_HEADER`. However, this property can be set to any of the following values:

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

Attribute mapping is available for response attributes. Therefore, the format of policy information can be mapped to the format of a header or a cookie. The property labeled Response Attribute Map (Tab: Application, Name: `com.sun.identity.agents.config.response.attribute.mapping`) is used for this type of mapping:

Unlike profile attributes and session attributes, where only the mapped attributes are displayed as headers or cookies, by default, response attributes are set by the agent as headers or cookies based on the setting of the Response Attribute Fetch Mode property.

If a response attribute map is specified, then the corresponding attribute mapped name is fetched from the map and its corresponding value is displayed as either a header or a cookie based on the setting of the above property.

Providing Personalization With User Profile Attributes Globally

Web agents in Policy Agent 3.0 have the ability to forward user profile attribute values via HTTP headers to end-web applications. The user profile attribute values come from the server side of OpenSSO Enterprise. The web agent behaves like a broker to obtain and relay user

attribute values to the destination servlets, CGI scripts, or ASP pages. These applications can in turn use the attribute values to personalize page content.

This feature is configurable through two web agent properties. To turn this feature on and off, edit the property labeled Profile Attribute Fetch Mode (Tab: Application, Name: `com.sun.identity.agents.config.profile.attribute.fetch.mode`)

This property can be set to one of the following values:

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

When set to `NONE`, the web agent does not fetch LDAP attributes from the server and ignores the web agent property labeled Profile Attribute Map (Tab: Application, Name: `com.sun.identity.agents.config.profile.attribute.mapping`). In the other two cases, the web agent fetches the attribute.

To configure the attributes that are to be forwarded in the HTTP headers, use the Profile Attribute Map property.

By default, some LDAP user attribute names and HTTP header names are set to sample values.

To find the appropriate LDAP user attribute names, check the following XML file on the machine where OpenSSO Enterprise is installed: `amUser.xml`

Note – The `amUser.xml` file is available from the directory within which the `opensso.war` file is deployed. This directory varies according to the web container. The following is an example of a possible location:

OpenSSO-Deploy-base`/WEB-INF/classes/amUser.xml`

The attributes in this file could be either OpenSSO Enterprise user attributes or OpenSSO Enterprise dynamic attributes. For an explanation of these two types of user attributes, see *Sun OpenSSO Enterprise 8.0 Administration Guide*

The attribute and HTTP header names that need to be forwarded must be determined by the end-user applications on the deployment container that the web agent is protecting. Basically, these applications are the consumers of the forwarded header values (the forwarded information is used for the customization and personalization of web pages).

Setting the Fully Qualified Domain Name

To ensure appropriate user experience, it is necessary that users access resources protected by the web agent using valid URLs. The property labeled FQDN Default (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.default`) provides the necessary information needed by the web agent to identify if the user is using a valid URL to access the protected resource. If the web 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 changed by the web agent to a fully qualified domain name (FQDN) as per the value specified in this property.

This is a required configuration property without which the deployment container may not start up correctly. This property is set during the web agent installation and must not be modified unless absolutely necessary to accommodate deployment requirements. An invalid value for this property can result in the deployment container becoming unusable or the resources becoming inaccessible.

The property labeled FQDN Virtual Host Map (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.mapping`) provides another way by which the web agent can resolve partial or malformed access URLs and take corrective action. The web agent gives precedence to the entries defined in this property over the value defined in the FQDN Default property. If none of the entries in this property matches the hostname specified in the user request, the agent uses the value specified for the FQDN Default property.

The FQDN Virtual Host Map property can be used for creating a mapping for more than one hostname. This may be the case when the deployment container protected by this agent is accessible by more than one hostname. However, this feature must be used with caution as it can lead to the deployment container resources becoming inaccessible.

This property can also be used to override the behavior of the web agent in cases where necessary. The format for assigning a value to the FQDN Virtual Host Map property is as follows:

Map Key	`invalid_hostname`
Corresponding Map Value	`valid_hostname`

where:

The `invalid_hostname` value is a possible invalid hostname such as partial hostname or an IP address that the user may provide .

The `valid_hostname` value is the corresponding valid hostname that is fully qualified. For example, the following are possible values assigned to the FQDN Virtual Host Map property in OpenSSO Enterprise Console for `xyz.domain1.com`:

Map Key	`xyz`
Corresponding Map Value	`xyz.domain1.com`
Map Key	`xyz.domain1`
Corresponding Map Value	`xyz.domain1.com`

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:

```
[xyz]=xyz.domain1.com

[xyz.domain1]=xyz.domain1.com
```

This property can also be used in such a way that the web agent uses the name specified in this map instead of the deployment container's actual name.

If you want your server to be addressed as `xyz.hostname.com` whereas the actual name of the server is `abc.hostname.com`. The browser only knows `xyz.hostname.com` and you have specified policies using `xyz.hostname.com` in OpenSSO Enterprise Console. Set the FQDN Virtual Host Map property as such:

Map Key	`valid`
Corresponding Map Value	`xyz.hostname.com`

Turning Off FQDN Mapping

You can disable fully qualified domain name (FQDN) mapping of HTTP requests.

This checking capability is controlled by the web agent properties labeled FQDN default (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.default`) and FQDN Virtual Host Map (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.mapping`).

The web agent property labeled FQDN Check (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.check.enable`) is enabled by default. When this property is enabled, the request URLs that are present in user requests are checked against FQDN values. However, you can turn FQDN checking off if you choose. Turning off FQDN checking might be beneficial when a deployment includes a number of virtual servers for which the agent is configured using FQDN mapping.

Resetting Cookies

The cookie reset feature enables the web agent to reset some cookies in the browser session while redirecting to OpenSSO Enterprise for authentication.

This feature is configurable through two web agent properties: the property labeled Cookie Reset (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset.enable`) and the property labeled Cookies Reset Name List (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.reset`):

- Cookie Reset

  This property must be enabled if the web agent is to reset cookies in the response while redirecting to OpenSSO Enterprise for authentication. By default, this property is not enabled.

- Cookie Reset Name List

  The values for this property are the cookies that need to be reset in the response while redirecting to OpenSSO Enterprise for authentication. This property is used only if the Cookie Reset feature is enabled.

  Cookie details must be specified in the following format:

  ```
  name[=value][;Domain=value]
  ```

  For example,

  ```
  cookie1=value1,
  cookie2=value2;Domain=example.com
  ```

Configuring CDSSO

Note – The cross domain single sign-on (CDSSO) feature does not apply to all web agents. When CDSSO is not supported by a specific web agent, such information is provided in the individual web agent guide.

The CDSSO feature is configurable through three web agent properties. Enable or disable this feature with the property labeled Cross Domain SSO (Tab: SSO, Name: `com.sun.identity.agents.config.cdsso.enable`). By default, this property is not enabled, and the feature is turned off.

Set the URL where the CDC controller is installed by assigning the URL as the value to the property labeled CDSSO Servlet URL (Tab: SSO, Name: `com.sun.identity.agents.config.cdsso.cdcservlet.url`).

The following is an example of the value that could be assigned to the CDSSO Servlet URL property:

```
http://OpenSSOhost.example.com:58080/amserver/cdcservlet
```

The third property involved in configuring CDSSO is labeled Cookies Domain List (Tab: SSO, Name: `com.sun.identity.agents.config.cookie.domain`). This property allows you to specify a list of domains in which cookies have to be set in a CDSSO scenario. This property is used only if CDSSO is enabled. If you leave this property blank, then the fully qualified cookie domain for the web agent server will be used for setting the cookie domain. In such a case, it is a host cookie and not a domain cookie.

For more information on CDSSO, see *Sun OpenSSO Enterprise 8.0 Technical Overview*.

Setting the REMOTE_USER Server Variable

The property labeled User ID Parameter (Tab: OpenSSO Services, Name: `com.sun.identity.agents.config.userid.param`) allows you to configure the user ID parameter passed by the session or user profile information from OpenSSO Enterprise. The user ID value is used by the agent to set the value of the `REMOTE_USER` server variable. By default, this parameter is set to `UserToken` and is fetched from session attributes.

It can be set to any other session attribute or profile attribute. The property labeled User ID Parameter Type (Tab: OpenSSO Services, Name: `com.sun.identity.agents.config.userid.param.type`) determines the location from which to retrieve the value: from user profiles or from session properties.

Note – Be aware that when this value is fetched from session properties, you must write server-side plug-in code in order to add session attributes after authentication.

Example 1: This example lists the values that you can set the User ID Parameter property to for session attributes:

`SESSION` (this is default)

`UserToken` (UserId, Principal, or any other session attribute)

Example 2: This example lists the values that you can set the User ID Parameter property to for LDAP user profile attributes:

LDAP

cn (any profile attribute)

Setting Anonymous User

For resources on the not-enforced list, the default configuration does not allow the REMOTE_USER variable to be set. The not-enforced list refers to values assigned to the property labeled Not Enforced URLs (Tab: Application, Name: com.sun.identity.agents.config.notenforced.url)

To enable the REMOTE_USER variable to be set for not-enforced URLs, you must enable the web agent property labeled Anonymous User (Tab: Miscellaneous, Name: com.sun.identity.agents.config.anonymous.user.enable). By default, this value is not enabled.

When you enable this property, the value of REMOTE_USER is set to the value contained in the web agent property labeled Anonymous User Default Value (Tab: Miscellaneous, Name: com.sun.identity.agents.config.anonymous.user.id). By default, the value assigned to this property is anonymous.

Validating Client IP Addresses

This feature can be used to enhance security by preventing the stealing or *hijacking* of SSO tokens.

By default, the web agent labeled Client IP Validation (Tab: Application, Name: com.sun.identity.agents.config.client.ip.validation.enable) is not enabled.

If you enable this property, client IP address validation is enforced for each incoming request that contains an SSO token. If the IP address from which the request was generated does not match the IP address issued for the SSO token, the request is denied. This is essentially the same as enforcing a deny policy.

This feature should not be used, however, if the client browser uses a web proxy or if a load balancer exists somewhere between the client browser and the agent-protected deployment container. In such cases, the IP address appearing in the request will not reflect the real IP address on which the client browser runs.

Resetting a Web Agent Profile 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 web agent installation process. For more information, see "Creating a Web Agent Profile in Policy Agent 3.0" on page 48. However, you can change this password if you choose.

The agent profile password can be updated 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 either prior to agent installation. However, after you install a web agent, you can update the agent profile password at anytime.

▼ To Update a Web 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 web agent properties of the agent that you want to configure.**

   For the steps to navigate to the web agent properties, see "To Navigate in the OpenSSO Enterprise 8.0 Console to the Web Agent Properties" on page 55.

2. **Update the agent profile password in the web 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 property labeled Password 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 web 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 a text editor, 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.**

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: nMXvXoCgWAAbTomKJ6H5/g==
   ```

   For more information on this command, see "`agentadmin --encrypt`" on page 35.

5. **Copy the encrypted value that is returned.**

6. **Using a text editor of your choice, access the web 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 by pasting the encrypted password, and therefore replacing the original value of the encrypted password, as shown:**

   `com.sun.identity.agents.config.password =` *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.sun.identity.agents.config.password = nMXvXoCgWAAbTomKJ6H5/g==
   ```

8. **Restart the web agent container.**

   The container must be restarted for the changes to the bootstrap file to take effect.

Configuring Web Agent Log Rotation

For web agents, when the current log file reaches a specific size, a new log file is created. Log information is then stored in the new log file until it reaches the size limit. This default behavior is configurable. Therefore, log rotation can be turned off or the size limit can be changed.

Note – The following types of information are logged for Policy Agent 3.0:

- Troubleshooting information
- Access denied information
- Access allowed information

The troubleshooting, or diagnostic, information is stored in log files, locally, with the web agent. The access denied and access allowed information, which is often referred to as audit-related information, can be stored both locally and with OpenSSO Enterprise.

Configuration that relates to the local log files is performed by editing the web agent property labeled Rotate Local Audit Log (Tab: Global, Name: `com.sun.identity.agents.config.local.log.rotate`). The Rotate Local Audit Log property is accessible using the OpenSSO Enterprise Console. Configuration that relates to the audit related logs stored with OpenSSO Enterprise is not controlled by an agent property, but this type of configuration can also be implemented using the Console.

The log rotation described in this section refers to logs that store troubleshooting information locally.

The local logs are rotated automatically since by default, the Rotate Local Audit Log property is enabled. When this property is not enabled, no rotation takes place for the local log file.

The following properties are also related to log rotation:

- The value of the following web agent property, which is available in the `OpenSSOAgentBootstrap.properties`file, indicates the location of the debug file:

  `com.sun.identity.agents.config.local.logfile`

  Be aware that this property is not available in OpenSSO Enterprise Console. Since a local audit file is created during agent installation, the location of that file is assigned to this bootstrap file property at that time.
- The value of the web agent property labeled Local Audit Log Rotation Size (Tab: Global, Name: `com.sun.identity.agents.config.local.log.size`) indicates the maximum number of bytes the debug file holds. You can set this agent property in OpenSSO Enterprise Console.

This property controls the log file size in that a new log file is created when the current log file reaches a specific size. The file size should be a minimum of 3000 bytes. The default size is 10 megabytes.

When a new log file is created an index appends to the name of the log file as such:

amAgent-1
amAgent-2

Where *amAgent* represents the fully qualified path name to the log files excluding the appended number. The numbers *1* and *2* represent the appended number. The appended number indicates the chronological order in which information of a given size was filed away into its respective log file. There is no limit to the number of log files that can be rotated.

Enabling Load Balancing

Various web agent properties influence the enablement of load balancing. Edit the properties that apply, according to the location of the load balancer or load balancers in your deployment, as follows:

- "Load Balancer in Front of OpenSSO Enterprise" on page 79
- "Load Balancer in Front of the Web Agent" on page 80
- "Load Balancers in Front of Both the Web Agent and OpenSSO Enterprise" on page 81

Load Balancer in Front of OpenSSO Enterprise

When a load balancer is deployed in front of OpenSSO Enterprise and a web agent interacts with the load balancer, the following web agent properties must be edited:

- `com.sun.identity.agents.config.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file)
- OpenSSO Login URL (Tab: OpenSSO Services, Name: `com.sun.identity.agents.config.login.url`)
- Load Balancer Setup (Tab: Advanced, Name: `com.sun.identity.agents.config.load.balancer.enable`)

EXAMPLE 4–7 Property Settings: Load Balancer in Front of OpenSSO Enterprise

This example illustrates the web agent property settings that can be used to enable load balancing:

EXAMPLE 4–7 Property Settings: Load Balancer in Front of OpenSSO Enterprise *(Continued)*

Property (name or label)	Setting
`com.sun.identity.agents.config.naming.url` (accessible in the `OpenSSOAgentBootstrap.properties` file)	*LB-url* `/amserver/namingservice`
OpenSSO Login URL	*LB-url* `/amserver/UI/Login`
Load Balancer Setup	Enabled

where *LB-url* represents the load balancer URL. The following example is a conceivable load balancer URL:

```
http://LBhost.example.com:8080
```

Load Balancer in Front of the Web Agent

In many cases, when a load balancer is deployed in front of the web agent only the property labeled FQDN Virtual Host Map (Tab: Global, Name: `com.sun.identity.agents.config.fqdn.mapping`) is required.

Assign the following value to the FQDN Virtual Host Map property:

`valid|`*LB-hostname*

where *LB-hostname* represents the name of the machine on which the load balancer is located.

However, if SSL-termination or a proxy server is used in the deployment, all the following web agent properties should be set in addition to the preceding property:

- Override Request URL Protocol (Tab: Advanced, Name: `com.sun.identity.agents.config.override.protocol`)
- Override Request URL Host (Tab: Advanced, Name: `com.sun.identity.agents.config.override.host`)
- Override Request URL Port (Tab: Advanced, Name: `com.sun.identity.agents.config.override.port`)
- Override Notification URL (Tab: Advanced, Name: `com.sun.identity.agents.config.agenturi.prefix`)

This example illustrates how properties can be set to enable load balancing when the protocol, hostname, and port number of the load balancer differ from that of the web agent. However, if the load balancer and the web agent share one of these characteristics, such as the protocol or hostname, then the respective property would not be enabled.

Property (name or label)	Setting
Override Request URL Protocol	`true`
Override Request URL Host	(Enabled)
Override Request URL Port	(Enabled)
Override Notification URL	*LB-url*`/amagent`

Load Balancers in Front of Both the Web Agent and OpenSSO Enterprise

This scenario is simply a combination of the scenarios described in the preceding sections. See "Load Balancer in Front of OpenSSO Enterprise" on page 79 and "Load Balancer in Front of the Web Agent" on page 80.

Composite Advice

Web agents provide a composite advice feature. This feature allows the policy and authentication services of OpenSSO Enterprise to decouple the advice handling mechanism of the agents. This allows you to introduce and manage custom advices by solely writing OpenSSO Enterprise side plug-ins. You are not required to make changes on the agent side. Such advices are honored automatically by the composite advice handling mechanism.

Malicious Header Attributes Automatically Cleared by Agents

For web agents in the Policy Agent software set, malicious header attributes are automatically cleared. The benefit of this automatic clean up is that security is improved. Header information that is *not* automatically cleared has greater risk of being accessed

◆ ◆ ◆ 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.

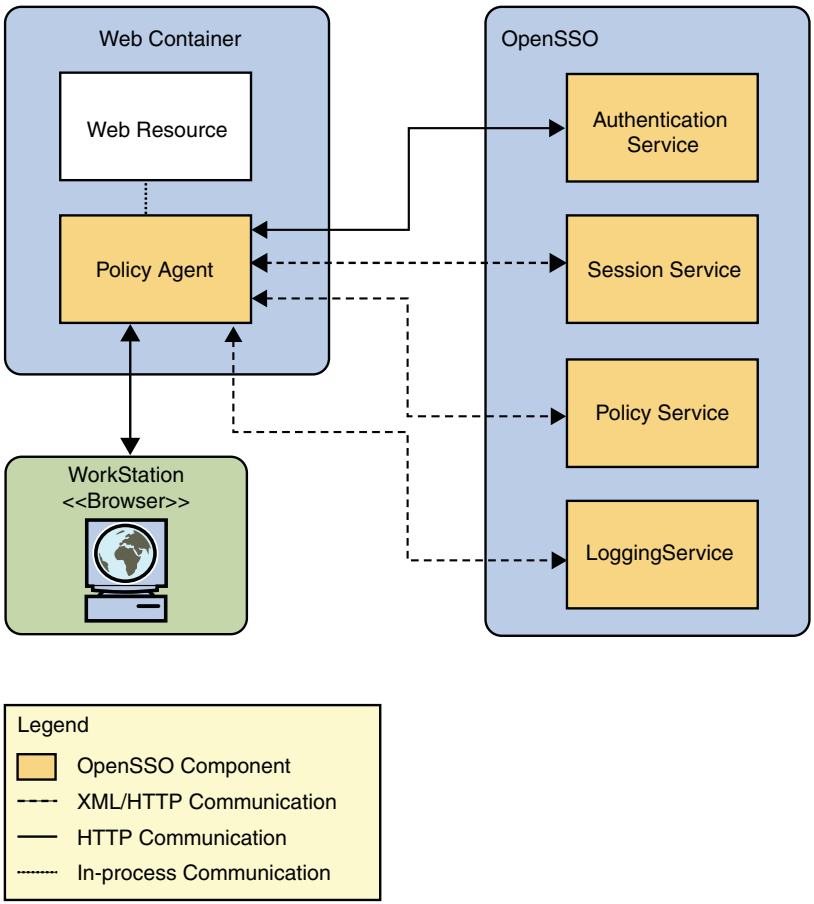

FIGURE A–1 Policy Agent Interaction with OpenSSO Enterprise Services

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.

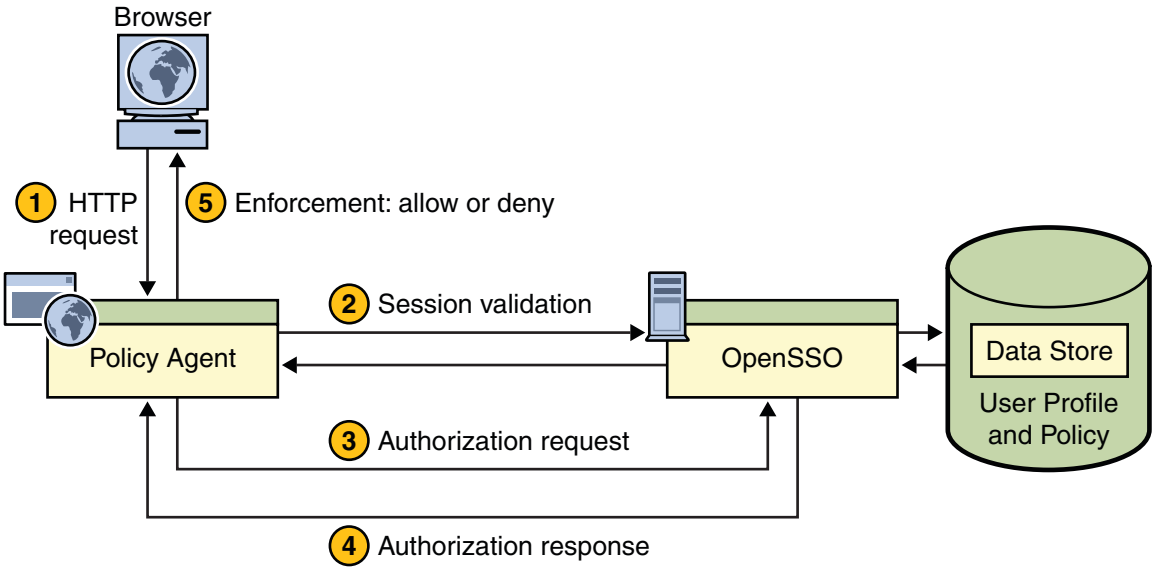

FIGURE A–2 Policy Agent and the Policy Decision Process

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.

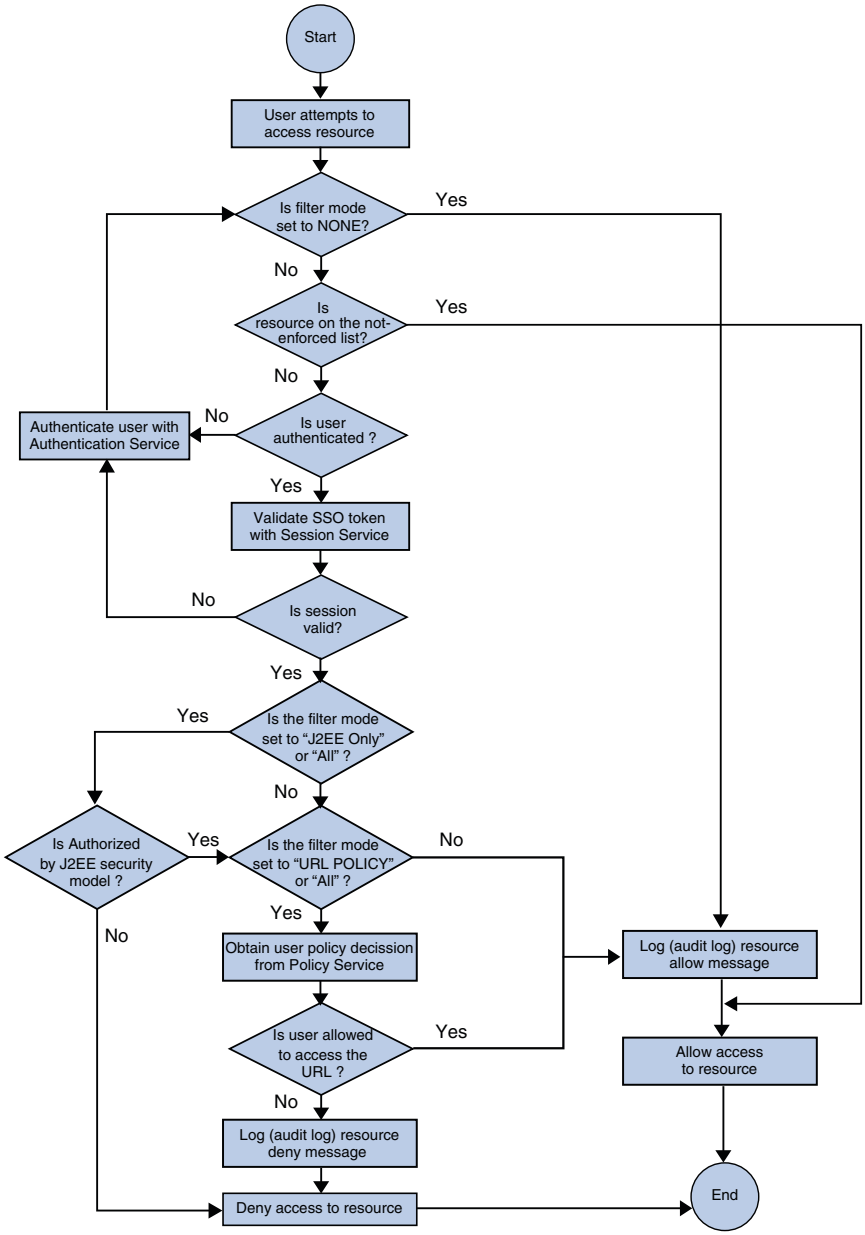

FIGURE A–3 J2EE Agents and the Policy Decision Process

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.

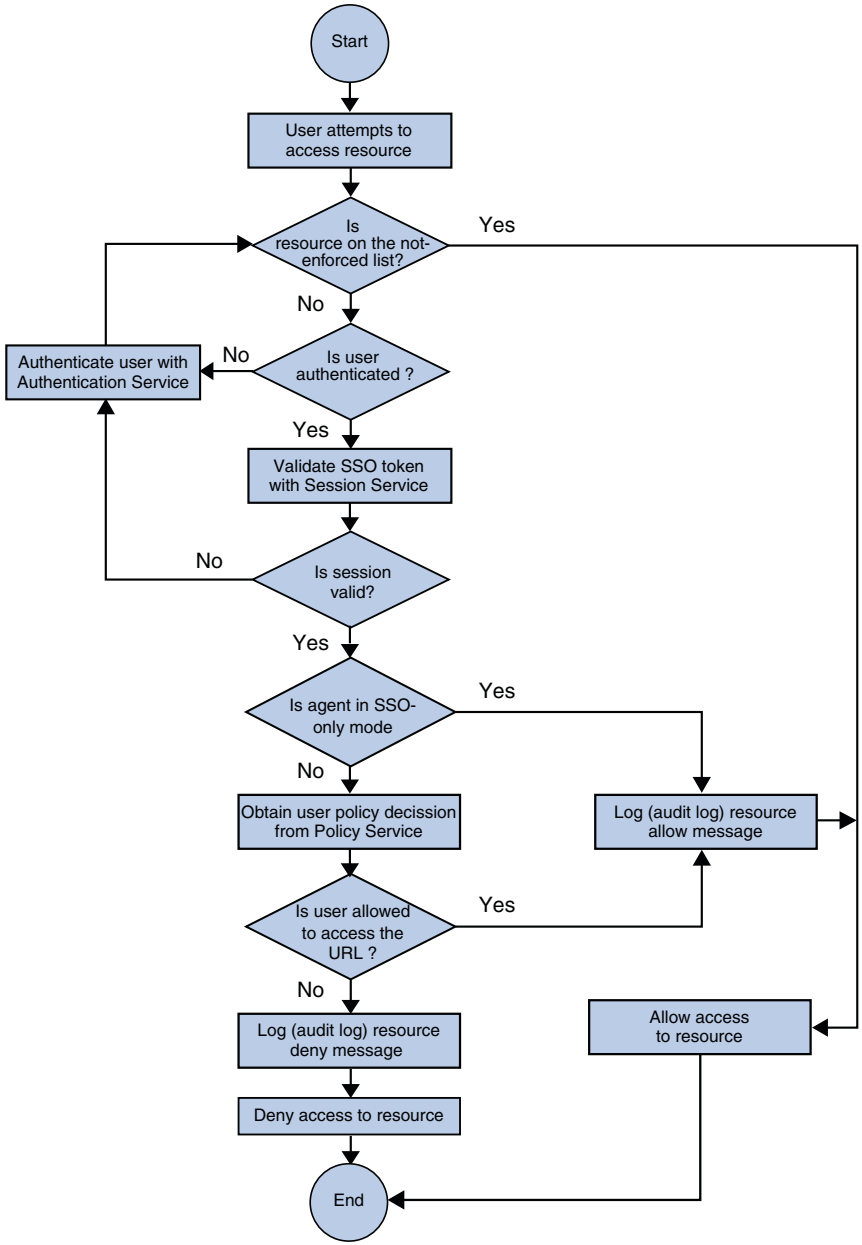

FIGURE A–4 Web Agents and the Policy Decision Process

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 85 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 94

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 Web Agent in Policy Agent 3.0

In addition to a standard installation and uninstallation of web 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 web agents.

About Silent Installation and Uninstallation of a Web 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 Web Agent Installation

This section describes how to generate a state file for installing a web 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 Web Agent Installation

To generate a state file for a web 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 web 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 28.

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 Web 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 Web Agent Using a State File

To perform a silent installation of a web 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 web 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 Web Agent Uninstallation

This section describes how to generate a state file for uninstalling a web 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 Web Agent Uninstallation

To generate a state file for uninstallation of a web 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 web 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 28.

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 Web 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 Web Agent Using a State File

To perform a silent uninstallation of a web 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 web 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 Web Agents

The OpenSSO Enterprise policy service supports policy definitions that use either of the two following wildcards:

- "The Multi-Level Wildcard: *" on page 102
- "The One-Level Wildcard: -*-" on page 103

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:

```
http://agentHost:8090/agentsample.com/*.gif
```

```
http://agentHost:8090/agentsample/-*-/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.url[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 Web 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

```
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.
```

◆ ◆ ◆ APPENDIX E

Web Agent Error Codes

This appendix lists the error codes you might encounter while installing and configuring a web agent. It also provides explanations for the each code item.

Error Code List

This list of error codes includes locations that are reserved for error codes that do not currently exist.

0. AM_SUCCESS	The operation completed successfully.
1. AM_FAILURE	The operation did not complete successfully. Please refer to the log file for more details.
2. AM_INIT_FAILURE	The C SDK initialization routine did not complete successfully. All the other APIs may be used only if the initialization went through successfully.
3. AM_AUTH_FAILURE	The authentication did not go through successfully. This error is returned either by the Authentication API or the Policy Initialization API, which tries to authenticate itself as a client to OpenSSO Enterprise.
4. AM_NAMING_FAILURE	The naming query failed. Please look at the log file for further information.
5. AM_SESSION_FAILURE	The session operation did not succeed. The operation may be any of the operations provided by the session API.
6. AM_POLICY_FAILURE	The policy operation failed. Details of policy failure may be found in the log file.

7. This is a reserved error code.	Currently, no error code exists at this location.
8. AM_INVALID_ARGUMENT	The API was invoked with one or more invalid parameters. Check the input provided to the function.
9. This is a reserved error code.	Currently, no error code exists at this location.
10. This is a reserved error code.	Currently, no error code exists at this location.
11. AM_NO_MEMORY	The operation failed because of a memory allocation problem.
12. AM_NSPR_ERROR	The underlying NSPR layer failed. Please check log for further details.
13. This is a reserved error code.	Currently, no error code exists at this location.
14. AM_BUFFER_TOO_SMALL	The web agent does not have memory allocated to receive data from OpenSSO Enterprise.
15. AM_NO_SUCH_SERVICE_TYPE	The service type input by the user does not exist. This is a more specific version of `AM_INVALID_ARGUMENT` error code. The error can occur in any of the API that take `am_policy_t` as a parameter.
16. AM_SERVICE_NOT_AVAILABLE	Currently, no error code exists at this location.
17. AM_ERROR_PARSING_XML	During communication with OpenSSO Enterprise, there was an error while parsing the incoming XML data.
18. AM_INVALID_SESSION	The session token provided to the API was invalid. The session may have timed out or the token is corrupted.
19. AM_INVALID_ACTION_TYPE	This exception occurs during policy evaluation, if such an action type does not exist for a given policy decision appropriately found for the resource.
20. AM_ACCESS_DENIED	The user is denied access to the resource for the kind of action requested.
21. AM_HTTP_ERROR	There was an HTTP protocol error while contacting OpenSSO Enterprise.
22. AM_INVALID_FQDN_ACCESS	The resource provided by the user is not a fully qualified domain name. This is a web container

	specific error and may be returned by the `am_web_is_access_allowed` function only.
23. AM_FEATURE_UNSUPPORTED	The feature being invoked is not implemented as of now. Only the interfaces have been defined.
24. AM_AUTH_CTX_INIT_FAILURE	The Auth context creation failed. This error is thrown by `am_auth_create_auth_context`.
25. AM_SERVICE_NOT_INITIALIZED	The service is not initialized. This error is thrown by `am_policy` functions if the provided service was not initialized previously using `am_policy_service_init`.
26. AM_INVALID_RESOURCE_FORMAT	This is a plug-in interface error. Implementors of the new resource format may throw this error if the input string does not meet their specified format. This error is thrown by the `am_web` layer, if the resource passed as parameter does not follow the standard URL format.
27. AM_NOTIF_NOT_ENABLED	This error is thrown if the notification registration API is invoked when the notification feature is disabled in the configuration file.
28. AM_ERROR_DISPATCH_LISTENER	Error during notification registration.
29. AM_REMOTE_LOG_FAILURE	This error code indicates that the service that logs messages to OpenSSO Enterprise has failed. The details of this error can be found in the web agent's log file.

◆ ◆ ◆ APPENDIX F

Developing Your Own OpenSSO Enterprise Web Agent

You can develop web agents yourself for OpenSSO Enterprise Policy Agent. By using the available C application programming interfaces (C API) and by following the provided samples, developing your own web agent can be a relatively simple development task, depending on the web container.

A web agent that suits your site's requirements might already exist in the OpenSSO Enterprise Policy Agent software set. If the web agent exists, you can download it from the agents download page. However, if a web agent is not currently available for a particular web container or platform, you can develop that agent yourself using the web agent development toolkit.

The Web Agent Development Toolkit

The web agent development toolkit refers to tools available for developing your own web agent for OpenSSO Enterprise Policy Agent. The basic tools of the toolkit are as follows:

- The C SDK (C software development kit), which comes with the OpenSSO Enterprise download.

  The C SDK includes the C API and sample source files that you can use as a blueprint for creating a web agent.

  This kit is not only for developing agents, though that is the focus of this appendix. This kit can be used to enable external C applications to participate in OpenSSO Enterprise authentication, authorization, single sign-on (SSO), and logging operations. The C SDK is available as a `.zip` file in the following directory:

  zip-root`/opensso/libraries/native/agent-csdk`

  See the `README.TXT` file in the `samples` directory of the C SDK for more information about using this web agent development toolkit.
- *Sun OpenSSO Enterprise 8.0 C API Reference for Application and Web Policy Agent Developers*

This guide describes the C API, providing information to help you create your own web agent.

By using the C SDK and the C API Reference described in the preceding list, you can create your own web agent that is compatible with OpenSSO Enterprise.

Index

R

S

U

W